Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition. Daniel Jurafsky & James H. Martin.

1 INTRODUCTION

Dave Bowman: Open the pod bay doors, HAL.
HAL: I'm sorry Dave, I'm afraid I can't do that.
Stanley Kubrick and Arthur C. Clarke,
screenplay of *2001: A Space Odyssey*

This book is about a new interdisciplinary field variously called **computer speech and language processing** or **human language technology** or **natural language processing** or **computational linguistics**. The goal of this new field is to get computers to perform useful tasks involving human language, tasks like enabling human-machine communication, improving human-human communication, or simply doing useful processing of text or speech.

CONVERSATIONAL AGENT

One example of a useful such task is a **conversational agent**. The HAL 9000 computer in Stanley Kubrick's film *2001: A Space Odyssey* is one of the most recognizable characters in twentieth-century cinema. HAL is an artificial agent capable of such advanced language-processing behavior as speaking and understanding English, and at a crucial moment in the plot, even reading lips. It is now clear that HAL's creator Arthur C. Clarke was a little optimistic in predicting when an artificial agent such as HAL would be available. But just how far off was he? What would it take to create at least the language-related parts of HAL? We call programs like HAL that converse with humans via natural language **conversational agents** or **dialogue systems**. In this text we study the various components that make up modern conversational agents, including language input (**automatic speech recognition** and **natural language understanding**) and language output (**natural language generation** and **speech synthesis**).

CONVERSATIONAL AGENTS

DIALOGUE SYSTEMS

Let's turn to another useful language-related task, that of making available to non-English-speaking readers the vast amount of scientific information on the Web in English. Or translating for English speakers the hundreds of millions of Web pages written in other languages like Chinese. The goal of **machine translation** is to automatically translate a document from one language to another. Machine translation is far from a solved problem; we will cover the algorithms currently used in the field, as well as important component tasks.

MACHINE TRANSLATION

Many other language processing tasks are also related to the Web. Another such task is **Web-based question answering**. This is a generalization of simple web search, where instead of just typing keywords a user might ask complete questions, ranging from easy to hard, like the following:

QUESTION ANSWERING

- What does “divergent” mean?
- What year was Abraham Lincoln born?
- How many states were in the United States that year?
- How much Chinese silk was exported to England by the end of the 18th century?
- What do scientists think about the ethics of human cloning?

Some of these, such as **definition** questions, or simple **factoid** questions like dates and locations, can already be answered by search engines. But answering more complicated questions might require extracting information that is embedded in other text on a Web page, or doing **inference** (drawing conclusions based on known facts), or synthesizing and summarizing information from multiple sources or web pages. In this text we study the various components that make up modern understanding systems of this kind, including **information extraction**, **word sense disambiguation**, and so on.

Although the subfields and problems we’ve described above are all very far from completely solved, these are all very active research areas and many technologies are already available commercially. In the rest of this chapter we briefly summarize the kinds of knowledge that is necessary for these tasks (and others like **spell correction**, **grammar checking**, and so on), as well as the mathematical models that will be introduced throughout the book.

1.1 Knowledge in Speech and Language Processing

What distinguishes language processing applications from other data processing systems is their use of *knowledge of language*. Consider the Unix `wc` program, which is used to count the total number of bytes, words, and lines in a text file. When used to count bytes and lines, `wc` is an ordinary data processing application. However, when it is used to count the words in a file it requires *knowledge about what it means to be a word*, and thus becomes a language processing system.

Of course, `wc` is an extremely simple system with an extremely limited and impoverished knowledge of language. Sophisticated conversational agents like HAL, or machine translation systems, or robust question-answering systems, require much broader and deeper knowledge of language. To get a feeling for the scope and kind of required knowledge, consider some of what HAL would need to know to engage in the dialogue that begins this chapter, or for a question answering system to answer one of the questions above.

HAL must be able to recognize words from an audio signal and to generate an audio signal from a sequence of words. These tasks of **speech recognition** and **speech synthesis** tasks require knowledge about **phonetics and phonology**; how words are pronounced in terms of sequences of sounds, and how each of these sounds is realized acoustically.

Note also that unlike Star Trek’s Commander Data, HAL is capable of producing contractions like *I’m* and *can’t*. Producing and recognizing these and other variations of individual words (e.g., recognizing that *doors* is plural) requires knowledge about **morphology**, the way words break down into component parts that carry meanings like *singular* versus *plural*.

Moving beyond individual words, HAL must use structural knowledge to properly string together the words that constitute its response. For example, HAL must know that the following sequence of words will not make sense to Dave, despite the fact that it contains precisely the same set of words as the original.

> I'm I do, sorry that afraid Dave I'm can't.

The knowledge needed to order and group words together comes under the heading of **syntax**.

Now consider a question answering system dealing with the following question:

- How much Chinese silk was exported to Western Europe by the end of the 18th century?

In order to answer this question we need to know something about **lexical semantics**, the meaning of all the words (*export*, or *silk*) as well as **compositional semantics** (what exactly constitutes *Western Europe* as opposed to Eastern or Southern Europe, what does *end* mean when combined with *the 18th century*. We also need to know something about the relationship of the words to the syntactic structure. For example we need to know that *by the end of the 18th century* is a temporal end-point, and not a description of the agent, as the by-phrase is in the following sentence:

- How much Chinese silk was exported to Western Europe by southern merchants?

We also need the kind of knowledge that lets HAL determine that Dave's utterance is a request for action, as opposed to a simple statement about the world or a question about the door, as in the following variations of his original statement.

REQUEST:	*HAL, open the pod bay door.*
STATEMENT:	*HAL, the pod bay door is open.*
INFORMATION QUESTION:	*HAL, is the pod bay door open?*

Next, despite its bad behavior, HAL knows enough to be polite to Dave. It could, for example, have simply replied *No* or *No, I won't open the door*. Instead, it first embellishes its response with the phrases *I'm sorry* and *I'm afraid*, and then only indirectly signals its refusal by saying *I can't*, rather than the more direct (and truthful) *I won't*.[1] This knowledge about the kind of actions that speakers intend by their use of sentences is **pragmatic** or **dialogue** knowledge.

Another kind of pragmatic or **discourse** knowledge is required to answer the question

- How many states were in the United States *that year*?

What year is *that year*? In order to interpret words like *that year* a question answering system need to examine the the earlier questions that were asked; in this case the previous question talked about the year that Lincoln was born. Thus this task of **coreference resolution** makes use of knowledge about how words like *that* or pronouns like *it* or *she* refer to previous parts of the **discourse**.

To summarize, engaging in complex language behavior requires various kinds of knowledge of language:

[1] For those unfamiliar with HAL, it is neither sorry nor afraid, nor is it incapable of opening the door. It has simply decided in a fit of paranoia to kill its crew.

- Phonetics and Phonology — knowledge about linguistic sounds
- Morphology — knowledge of the meaningful components of words
- Syntax — knowledge of the structural relationships between words
- Semantics — knowledge of meaning
- Pragmatics — knowledge of the relationship of meaning to the goals and intentions of the speaker.
- Discourse — knowledge about linguistic units larger than a single utterance

1.2 Ambiguity

AMBIGUITY
AMBIGUOUS

A perhaps surprising fact about these categories of linguistic knowledge is that most tasks in speech and language processing can be viewed as resolving **ambiguity** at one of these levels. We say some input is **ambiguous** if there are multiple alternative linguistic structures that can be built for it. Consider the spoken sentence *I made her duck.* Here's five different meanings this sentence could have (see if you can think of some more), each of which exemplifies an ambiguity at some level:

(1.1) I cooked waterfowl for her.

(1.2) I cooked waterfowl belonging to her.

(1.3) I created the (plaster?) duck she owns.

(1.4) I caused her to quickly lower her head or body.

(1.5) I waved my magic wand and turned her into undifferentiated waterfowl.

These different meanings are caused by a number of ambiguities. First, the words *duck* and *her* are morphologically or syntactically ambiguous in their part-of-speech. *Duck* can be a verb or a noun, while *her* can be a dative pronoun or a possessive pronoun. Second, the word *make* is semantically ambiguous; it can mean *create* or *cook*. Finally, the verb *make* is syntactically ambiguous in a different way. *Make* can be transitive, that is, taking a single direct object (1.2), or it can be ditransitive, that is, taking two objects (1.5), meaning that the first object (*her*) got made into the second object (*duck*). Finally, *make* can take a direct object and a verb (1.4), meaning that the object (*her*) got caused to perform the verbal action (*duck*). Furthermore, in a spoken sentence, there is an even deeper kind of ambiguity; the first word could have been *eye* or the second word *maid*.

We will often introduce the models and algorithms we present throughout the book as ways to **resolve** or **disambiguate** these ambiguities. For example deciding whether *duck* is a verb or a noun can be solved by **part-of-speech tagging**. Deciding whether *make* means "create" or "cook" can be solved by **word sense disambiguation**. Resolution of part-of-speech and word sense ambiguities are two important kinds of **lexical disambiguation**. A wide variety of tasks can be framed as lexical disambiguation problems. For example, a text-to-speech synthesis system reading the word *lead* needs to decide whether it should be pronounced as in *lead pipe* or as in *lead me on*. By contrast, deciding whether *her* and *duck* are part of the same entity (as in (1.1) or (1.4)) or are different entity (as in (1.2)) is an example of **syntactic disambiguation** and can be addressed by **probabilistic parsing**. Ambiguities that don't arise in this particular example (like whether a given sentence is a statement or a question) will also be resolved, for example by **speech act interpretation**.

1.3 Models and Algorithms

One of the key insights of the last 50 years of research in language processing is that the various kinds of knowledge described in the last sections can be captured through the use of a small number of formal models, or theories. Fortunately, these models and theories are all drawn from the standard toolkits of computer science, mathematics, and linguistics and should be generally familiar to those trained in those fields. Among the most important models are **state machines**, **rule systems**, **logic**, **probabilistic models**, and **vector-space models**. These models, in turn, lend themselves to a small number of algorithms, among the most important of which are **state space search** algorithms such as **dynamic programming**, and machine learning algorithms such as **classifiers** and **EM** and other learning algorithms.

In their simplest formulation, state machines are formal models that consist of states, transitions among states, and an input representation. Some of the variations of this basic model that we will consider are **deterministic** and **non-deterministic finite-state automata** and **finite-state transducers**.

Closely related to these models are their declarative counterparts: formal rule systems. Among the more important ones we will consider are **regular grammars** and **regular relations**, **context-free grammars**, **feature-augmented grammars**, as well as probabilistic variants of them all. State machines and formal rule systems are the main tools used when dealing with knowledge of phonology, morphology, and syntax.

The third model that plays a critical role in capturing knowledge of language is logic. We will discuss **first order logic**, also known as the **predicate calculus**, as well as such related formalisms as lambda-calculus, feature-structures, and semantic primitives. These logical representations have traditionally been used for modeling semantics and pragmatics, although more recent work has focused on more robust techniques drawn from non-logical lexical semantics.

Probabilistic models are crucial for capturing every kind of linguistic knowledge. Each of the other models (state machines, formal rule systems, and logic) can be augmented with probabilities. For example the state machine can be augmented with probabilities to become the **weighted automaton** or **Markov model**. We will spend a significant amount of time on **hidden Markov models** or **HMMs**, which are used everywhere in the field, in part-of-speech tagging, speech recognition, dialogue understanding, text-to-speech, and machine translation. The key advantage of probabilistic models is their ability to to solve the many kinds of ambiguity problems that we discussed earlier; almost any speech and language processing problem can be recast as: "given *N* choices for some ambiguous input, choose the most probable one".

Finally, vector-space models, based on linear algebra, underlie information retrieval and many treatments of word meanings.

Processing language using any of these models typically involves a search through a space of states representing hypotheses about an input. In speech recognition, we search through a space of phone sequences for the correct word. In parsing, we search through a space of trees for the syntactic parse of an input sentence. In machine translation, we search through a space of translation hypotheses for the correct translation of a sentence into another language. For non-probabilistic tasks, such as state machines, we use well-known graph algorithms such as **depth-first search**. For probabilistic tasks, we use heuristic variants such as **best-first** and **A* search**, and rely on dynamic programming algorithms for computational tractability.

For many language tasks, we rely on machine learning tools like **classifiers** and **sequence models**. Classifiers like **decision trees**, **support vector machines**, **Gaussian Mixture Models** and **logistic regression** are very commonly used. A hidden Markov model is one kind of sequence model; other are **Maximum Entropy Markov Models** or **Conditional Random Fields**.

Another tool that is related to machine learning is methodological; the use of distinct training and test sets, statistical techniques like **cross-validation**, and careful evaluation of our trained systems.

1.4 Language, Thought, and Understanding

To many, the ability of computers to process language as skillfully as we humans do will signal the arrival of truly intelligent machines. The basis of this belief is the fact that the effective use of language is intertwined with our general cognitive abilities. Among the first to consider the computational implications of this intimate connection was Alan Turing (1950). In this famous paper, Turing introduced what has come to be known as the **Turing Test**. Turing began with the thesis that the question of what it would mean for a machine to think was essentially unanswerable due to the inherent imprecision in the terms *machine* and *think*. Instead, he suggested an empirical test, a game, in which a computer's use of language would form the basis for determining if it could think. If the machine could win the game it would be judged intelligent.

TURING TEST

In Turing's game, there are three participants: two people and a computer. One of the people is a contestant and plays the role of an interrogator. To win, the interrogator must determine which of the other two participants is the machine by asking a series of questions via a teletype. The task of the machine is to fool the interrogator into believing it is a person by responding as a person would to the interrogator's questions. The task of the second human participant is to convince the interrogator that the other participant is the machine, and that they are human.

The following interaction from Turing's paper illustrates the kind of interactions he had in mind. Clearly a convincing simulation of humanity doesn't require being an expert in all domains:

> Q: Please write me a sonnet on the topic of the Forth Bridge.
> A: Count me out on this one. I never could write poetry.
> Q: Add 34957 to 70764.
> A: (Pause about 30 seconds and then give answer as) 105621.

Turing predicted that roughly by the end of the twentieth century a machine with 10 gigabytes of memory would have around a 30% chance of fooling a human interrogator after 5 minutes of questioning. Given the fact that you can fool some of the people all the time, it is not clear how rigorous this particular standard is. Nevertheless, the critical issue for Turing was that using language as humans do is sufficient, by itself, as an operational test for intelligence.

Surprisingly, many of the issues raised by Turing became relevant as early as 1966 in the context of the ELIZA program (Weizenbaum, 1966). ELIZA was an early natural language processing system capable of carrying on a limited form of conversation with a user. Consider the following session with a version of ELIZA that imitated the responses of a Rogerian psychotherapist.

User$_1$: You are like my father in some ways.
ELIZA$_1$: WHAT RESEMBLANCE DO YOU SEE
User$_2$: You are not very aggressive but I think you don't want me to notice that.
ELIZA$_2$: WHAT MAKES YOU THINK I AM NOT AGGRESSIVE
User$_3$: You don't argue with me.
ELIZA$_3$: WHY DO YOU THINK I DON'T ARGUE WITH YOU
User$_4$: You are afraid of me.
ELIZA$_4$: DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU

As we will discuss in more detail in Ch. 2, ELIZA is a remarkably simple program that makes use of pattern-matching to process the input and translate it into suitable outputs. The success of this simple technique in this domain is due to the fact that ELIZA doesn't actually need to *know* anything to mimic a Rogerian psychotherapist. As Weizenbaum notes, this is one of the few dialogue genres where the listener can act as if they know nothing of the world.

ELIZA's deep relevance to Turing's ideas is that many people who interacted with ELIZA came to believe that it really *understood* them and their problems. Indeed, Weizenbaum (1976) notes that many of these people continued to believe in ELIZA's abilities even after the program's operation was explained to them. In more recent years, Weizenbaum's informal reports have been repeated in a somewhat more controlled setting. Since 1991, an event known as the Loebner Prize competition has attempted to put various computer programs to the Turing test. Although these contests seem to have little scientific interest, a consistent result over the years has been that even the crudest programs can fool some of the judges some of the time (Shieber, 1994). Not surprisingly, these results have done nothing to quell the ongoing debate over the suitability of the Turing test as a test for intelligence among philosophers and AI researchers (Searle, 1980).

Fortunately, for the purposes of this book, the relevance of these results does not hinge on whether or not computers will ever be intelligent, or understand natural language. Far more important is recent related research in the social sciences that has confirmed another of Turing's predictions from the same paper.

> Nevertheless I believe that at the end of the century the use of words and educated opinion will have altered so much that we will be able to speak of machines thinking without expecting to be contradicted.

It is now clear that regardless of what people believe or know about the inner workings of computers, they talk about them and interact with them as social entities. People act toward computers as if they were people; they are polite to them, treat them as team members, and expect among other things that computers should be able to understand their needs, and be capable of interacting with them naturally. For example, Reeves and Nass (1996) found that when a computer asked a human to evaluate how well the computer had been doing, the human gives more positive responses than when a different computer asks the same questions. People seemed to be afraid of being impolite. In a different experiment, Reeves and Nass found that people also give computers higher performance ratings if the computer has recently said something flattering to the human. Given these predispositions, speech and language-based systems may provide many users with the most natural interface for many applications. This fact has led to a long-term focus in the field on the design of **conversational agents**, artificial entities that communicate conversationally.

1.5 The State of the Art

> We can only see a short distance ahead, but we can see plenty there that needs to be done.
>
> Alan Turing.

This is an exciting time for the field of speech and language processing. The startling increase in computing resources available to the average computer user, the rise of the Web as a massive source of information and the increasing availability of wireless mobile access have all placed speech and language processing applications in the technology spotlight. The following are examples of some currently deployed systems that reflect this trend:

- Travelers calling Amtrak, United Airlines and other travel-providers interact with conversational agents that guide them through the process of making reservations and getting arrival and departure information.
- Luxury car makers such as Mercedes-Benz models provide automatic speech recognition and text-to-speech systems that allow drivers to control their environmental, entertainment and navigational systems by voice. A similar spoken dialogue system has been deployed by astronauts on the International Space Station .
- Blinkx, and other video search companies, provide search services for million of hours of video on the Web by using speech recognition technology to capture the words in the sound track.
- Google provides cross-language information retrieval and translation services where a user can supply queries in their native language to search collections in another language. Google translates the query, finds the most relevant pages and then automatically translates them back to the user's native language.
- Large educational publishers such as Pearson, as well as testing services like ETS, use automated systems to analyze thousands of student essays, grading and assessing them in a manner that is indistinguishable from human graders.

- Interactive tutors, based on lifelike animated characters, serve as tutors for children learning to read, and as therapists for people dealing with aphasia and Parkinsons disease. (?, ?)
- Text analysis companies such as Nielsen Buzzmetrics, Umbria, and Collective Intellect, provide marketing intelligence based on automated measurements of user opinions, preferences, attitudes as expressed in weblogs, discussion forums and and user groups.

1.6 Some Brief History

Historically, speech and language processing has been treated very differently in computer science, electrical engineering, linguistics, and psychology/cognitive science. Because of this diversity, speech and language processing encompasses a number of different but overlapping fields in these different departments: **computational linguistics** in linguistics, **natural language processing** in computer science, **speech recognition** in electrical engineering, **computational psycholinguistics** in psychology. This section summarizes the different historical threads which have given rise to the field of speech and language processing. This section will provide only a sketch; see the individual chapters for more detail on each area and its terminology.

1.6.1 Foundational Insights: 1940s and 1950s

The earliest roots of the field date to the intellectually fertile period just after World War II that gave rise to the computer itself. This period from the 1940s through the end of the 1950s saw intense work on two foundational paradigms: the **automaton** and **probabilistic** or **information-theoretic models**.

The automaton arose in the 1950s out of Turing's (1936) model of algorithmic computation, considered by many to be the foundation of modern computer science. Turing's work led first to the **McCulloch-Pitts neuron** (McCulloch and Pitts, 1943), a simplified model of the neuron as a kind of computing element that could be described in terms of propositional logic, and then to the work of Kleene (1951) and (1956) on finite automata and regular expressions. Shannon (1948) applied probabilistic models of discrete Markov processes to automata for language. Drawing the idea of a finite-state Markov process from Shannon's work, Chomsky (1956) first considered finite-state machines as a way to characterize a grammar, and defined a finite-state language as a language generated by a finite-state grammar. These early models led to the field of **formal language theory**, which used algebra and set theory to define formal languages as sequences of symbols. This includes the context-free grammar, first defined by Chomsky (1956) for natural languages but independently discovered by Backus (1959) and Naur et al. (1960) in their descriptions of the ALGOL programming language.

The second foundational insight of this period was the development of probabilistic algorithms for speech and language processing, which dates to Shannon's other contribution: the metaphor of the **noisy channel** and **decoding** for the transmission of language through media like communication channels and speech acoustics. Shannon also borrowed the concept of **entropy** from thermodynamics as a way of measuring the information capacity of a channel, or the information content of a language, and performed the first measure of the entropy of English using probabilistic techniques.

It was also during this early period that the sound spectrograph was developed (Koenig et al., 1946), and foundational research was done in instrumental phonetics that laid the groundwork for later work in speech recognition. This led to the first machine speech recognizers in the early 1950s. In 1952, researchers at Bell Labs built a statistical system that could recognize any of the 10 digits from a single speaker (Davis et al., 1952). The system had 10 speaker-dependent stored patterns roughly representing the first two vowel formants in the digits. They achieved 97–99% accuracy by choosing the pattern which had the highest relative correlation coefficient with the input.

1.6.2 The Two Camps: 1957–1970

By the end of the 1950s and the early 1960s, speech and language processing had split very cleanly into two paradigms: symbolic and stochastic.

The symbolic paradigm took off from two lines of research. The first was the work of Chomsky and others on formal language theory and generative syntax throughout the late 1950s and early to mid 1960s, and the work of many linguistics and computer scientists on parsing algorithms, initially top-down and bottom-up and then via dynamic programming. One of the earliest complete parsing systems was Zelig Harris's Transformations and Discourse Analysis Project (TDAP), which was implemented between June 1958 and July 1959 at the University of Pennsylvania (Harris, 1962).[2] The second line of research was the new field of artificial intelligence. In the summer of 1956 John McCarthy, Marvin Minsky, Claude Shannon, and Nathaniel Rochester brought together a group of researchers for a two-month workshop on what they decided to call artificial intelligence (AI). Although AI always included a minority of researchers focusing on stochastic and statistical algorithms (include probabilistic models and neural nets), the major focus of the new field was the work on reasoning and logic typified by Newell and Simon's work on the Logic Theorist and the General Problem Solver. At this point early natural language understanding systems were built, These were simple systems that worked in single domains mainly by a combination of pattern matching and keyword search with simple heuristics for reasoning and question-answering. By the late 1960s more formal logical systems were developed.

The stochastic paradigm took hold mainly in departments of statistics and of electrical engineering. By the late 1950s the Bayesian method was beginning to be applied to the problem of optical character recognition. Bledsoe and Browning (1959) built a Bayesian system for text-recognition that used a large dictionary and computed the likelihood of each observed letter sequence given each word in the dictionary by multiplying the likelihoods for each letter. Mosteller and Wallace (1964) applied Bayesian methods to the problem of authorship attribution on *The Federalist* papers.

The 1960s also saw the rise of the first serious testable psychological models of

[2] This system was reimplemented recently and is described by Joshi and Hopely (1999) and Karttunen (1999), who note that the parser was essentially implemented as a cascade of finite-state transducers.

human language processing based on transformational grammar, as well as the first on-line corpora: the Brown corpus of American English, a 1 million word collection of samples from 500 written texts from different genres (newspaper, novels, non-fiction, academic, etc.), which was assembled at Brown University in 1963–64 (Kučera and Francis, 1967; Francis, 1979; Francis and Kučera, 1982), and William S. Y. Wang's 1967 DOC (Dictionary on Computer), an on-line Chinese dialect dictionary.

1.6.3 Four Paradigms: 1970–1983

The next period saw an explosion in research in speech and language processing and the development of a number of research paradigms that still dominate the field.

The **stochastic** paradigm played a huge role in the development of speech recognition algorithms in this period, particularly the use of the Hidden Markov Model and the metaphors of the noisy channel and decoding, developed independently by Jelinek, Bahl, Mercer, and colleagues at IBM's Thomas J. Watson Research Center, and by Baker at Carnegie Mellon University, who was influenced by the work of Baum and colleagues at the Institute for Defense Analyses in Princeton. AT&T's Bell Laboratories was also a center for work on speech recognition and synthesis; see Rabiner and Juang (1993) for descriptions of the wide range of this work.

The **logic-based** paradigm was begun by the work of Colmerauer and his colleagues on Q-systems and metamorphosis grammars (Colmerauer, 1970, 1975), the forerunners of Prolog, and Definite Clause Grammars (Pereira and Warren, 1980). Independently, Kay's (1979) work on functional grammar, and shortly later, Bresnan and Kaplan's (1982) work on LFG, established the importance of feature structure unification.

The **natural language understanding** field took off during this period, beginning with Terry Winograd's SHRDLU system, which simulated a robot embedded in a world of toy blocks (Winograd, 1972). The program was able to accept natural language text commands *(Move the red block on top of the smaller green one)* of a hitherto unseen complexity and sophistication. His system was also the first to attempt to build an extensive (for the time) grammar of English, based on Halliday's systemic grammar. Winograd's model made it clear that the problem of parsing was well-enough understood to begin to focus on semantics and discourse models. Roger Schank and his colleagues and students (in what was often referred to as the *Yale School*) built a series of language understanding programs that focused on human conceptual knowledge such as scripts, plans and goals, and human memory organization (Schank and Albelson, 1977; Schank and Riesbeck, 1981; Cullingford, 1981; Wilensky, 1983; Lehnert, 1977). This work often used network-based semantics (Quillian, 1968; Norman and Rumelhart, 1975; Schank, 1972; Wilks, 1975b, 1975a; Kintsch, 1974) and began to incorporate Fillmore's notion of case roles (Fillmore, 1968) into their representations (Simmons, 1973).

The logic-based and natural-language understanding paradigms were unified on systems that used predicate logic as a semantic representation, such as the LUNAR question-answering system (Woods, 1967, 1973).

The **discourse modeling** paradigm focused on four key areas in discourse. Grosz and her colleagues introduced the study of substructure in discourse, and of discourse focus (Grosz, 1977; Sidner, 1983), a number of researchers began to work on automatic reference resolution (Hobbs, 1978), and the **BDI** (Belief-Desire-Intention) framework for logic-based work on speech acts was developed (Perrault and Allen, 1980; Cohen and Perrault, 1979).

1.6.4 Empiricism and Finite State Models Redux: 1983–1993

This next decade saw the return of two classes of models which had lost popularity in the late 1950s and early 1960s, partially due to theoretical arguments against them such as Chomsky's influential review of Skinner's *Verbal Behavior* (Chomsky, 1959). The first class was finite-state models, which began to receive attention again after work on finite-state phonology and morphology by Kaplan and Kay (1981) and finite-state models of syntax by Church (1980). A large body of work on finite-state models will be described throughout the book.

The second trend in this period was what has been called the "return of empiricism"; most notably here was the rise of probabilistic models throughout speech and language processing, influenced strongly by the work at the IBM Thomas J. Watson Research Center on probabilistic models of speech recognition. These probabilistic methods and other such data-driven approaches spread from speech into part-of-speech tagging, parsing and attachment ambiguities, and semantics. This empirical direction was also accompanied by a new focus on model evaluation, based on using held-out data, developing quantitative metrics for evaluation, and emphasizing the comparison of performance on these metrics with previous published research.

This period also saw considerable work on natural language generation.

1.6.5 The Field Comes Together: 1994–1999

By the last five years of the millennium it was clear that the field was vastly changing. First, probabilistic and data-driven models had become quite standard throughout natural language processing. Algorithms for parsing, part-of-speech tagging, reference resolution, and discourse processing all began to incorporate probabilities, and employ evaluation methodologies borrowed from speech recognition and information retrieval. Second, the increases in the speed and memory of computers had allowed commercial exploitation of a number of subareas of speech and language processing, in particular speech recognition and spelling and grammar checking. Speech and language processing algorithms began to be applied to Augmentative and Alternative Communication (AAC). Finally, the rise of the Web emphasized the need for language-based information retrieval and information extraction.

,

1.6.6 The Rise of Machine Learning: 2000–2007

The empiricist trends begun in the latter part of the 1990s accelerated at an astounding pace in the new century. This acceleration was largely driven by three synergistic trends. First, large amounts of spoken and written material became widely available through the auspices of the Linguistic Data Consortium (LDC), and other similar organizations. Importantly, included among these materials were annotated collections such as the Penn Treebank(Marcus et al., 1993), Prague Dependency Treebank(Hajič, 1998), PropBank(Palmer et al., 2005), Penn Discourse Treebank(Miltsakaki et al., 2004), RSTBank(Carlson et al., 2001) and TimeBank(?), all of which layered standard text sources with various forms of syntactic, semantic and pragmatic annotations. The existence of these resources promoted the trend of casting more complex traditional problems, such as parsing and semantic analysis, as problems in supervised machine learning. These resources also promoted the establishment of additional competitive evaluations for parsing (Dejean and Tjong Kim Sang, 2001), information extraction(?, ?), word sense disambiguation(Palmer et al., 2001; Kilgarriff and Palmer, 2000) and question answering(Voorhees and Tice, 1999).

Second, this increased focus on learning led to a more serious interplay with the statistical machine learning community. Techniques such as support vector machines (?; Vapnik, 1995), multinomial logistic regression (MaxEnt) (Berger et al., 1996), and graphical Bayesian models (Pearl, 1988) became standard practice in computational linguistics. Third, the widespread availability of high-performance computing systems facilitated the training and deployment of systems that could not have been imagined a decade earlier.

Finally, near the end of this period, largely unsupervised statistical approaches began to receive renewed attention. Progress on statistical approaches to machine translation(Brown et al., 1990; Och and Ney, 2003) and topic modeling (?) demonstrated that effective applications could be constructed from systems trained on unannotated data alone. In addition, the widespread cost and difficulty of producing reliably annotated corpora became a limiting factor in the use of supervised approaches for many problems. This trend towards the use unsupervised techniques will likely increase.

1.6.7 On Multiple Discoveries

Even in this brief historical overview, we have mentioned a number of cases of multiple independent discoveries of the same idea. Just a few of the "multiples" to be discussed in this book include the application of dynamic programming to sequence comparison by Viterbi, Vintsyuk, Needleman and Wunsch, Sakoe and Chiba, Sankoff, Reichert *et al.*, and Wagner and Fischer (Chapters 3, 5 and 6) the HMM/noisy channel model of speech recognition by Baker and by Jelinek, Bahl, and Mercer (Chapters 6, 9, and 10); the development of context-free grammars by Chomsky and by Backus and Naur (Chapter 12); the proof that Swiss-German has a non-context-free syntax by Huybregts and by Shieber (Chapter 15); the application of unification to language processing by Colmerauer *et al.* and by Kay in (Chapter 16).

Are these multiples to be considered astonishing coincidences? A well-known hypothesis by sociologist of science Robert K. Merton (1961) argues, quite the contrary, that

> all scientific discoveries are in principle multiples, including those that on the surface appear to be singletons.

Of course there are many well-known cases of multiple discovery or invention; just a few examples from an extensive list in Ogburn and Thomas (1922) include the multiple invention of the calculus by Leibniz and by Newton, the multiple development of the theory of natural selection by Wallace and by Darwin, and the multiple invention of the telephone by Gray and Bell.[3] But Merton gives a further array of evidence for the hypothesis that multiple discovery is the rule rather than the exception, including many cases of putative singletons that turn out be a rediscovery of previously unpublished or perhaps inaccessible work. An even stronger piece of evidence is his ethnomethodological point that scientists themselves act under the assumption that multiple invention is the norm. Thus many aspects of scientific life are designed to help scientists avoid being "scooped"; submission dates on journal articles; careful dates in research records; circulation of preliminary or technical reports.

1.6.8 A Final Brief Note on Psychology

Many of the chapters in this book include short summaries of psychological research on human processing. Of course, understanding human language processing is an important scientific goal in its own right and is part of the general field of cognitive science. However, an understanding of human language processing can often be helpful in building better machine models of language. This seems contrary to the popular wisdom, which holds that direct mimicry of nature's algorithms is rarely useful in engineering applications. For example, the argument is often made that if we copied nature exactly, airplanes would flap their wings; yet airplanes with fixed wings are a more successful engineering solution. But language is not aeronautics. Cribbing from nature is sometimes useful for aeronautics (after all, airplanes do have wings), but it is particularly useful when we are trying to solve human-centered tasks. Airplane flight has different goals than bird flight; but the goal of speech recognition systems, for example, is to perform exactly the task that human court reporters perform every day: transcribe spoken dialog. Since people already do this well, we can learn from nature's previous solution. Since an important application of speech and language processing systems is for human-computer interaction, it makes sense to copy a solution that behaves the way people are accustomed to.

1.7 Summary

This chapter introduces the field of speech and language processing. The following are some of the highlights of this chapter.

- A good way to understand the concerns of speech and language processing research is to consider what it would take to create an intelligent agent like HAL from 2001: A Space Odyssey, or build a web-based question answerer, or a machine translation engine.
- Speech and language technology relies on formal models, or representations, of

[3] Ogburn and Thomas are generally credited with noticing that the prevalence of multiple inventions suggests that the cultural milieu and not individual genius is the deciding causal factor in scientific discovery. In an amusing bit of recursion, however, Merton notes that even this idea has been multiply discovered, citing sources from the 19th century and earlier!

knowledge of language at the levels of phonology and phonetics, morphology, syntax, semantics, pragmatics and discourse. A small number of formal models including state machines, formal rule systems, logic, and probabilistic models are used to capture this knowledge.

- The foundations of speech and language technology lie in computer science, linguistics, mathematics, electrical engineering and psychology. A small number of algorithms from standard frameworks are used throughout speech and language processing,
- The critical connection between language and thought has placed speech and language processing technology at the center of debate over intelligent machines. Furthermore, research on how people interact with complex media indicates that speech and language processing technology will be critical in the development of future technologies.
- Revolutionary applications of speech and language processing are currently in use around the world. The creation of the web, as well as significant recent improvements in speech recognition and synthesis, will lead to many more applications.

Bibliographical and Historical Notes

Research in the various subareas of speech and language processing is spread across a wide number of conference proceedings and journals. The conferences and journals most centrally concerned with natural language processing and computational linguistics are associated with the Association for Computational Linguistics (ACL), its European counterpart (EACL), and the International Conference on Computational Linguistics (COLING). The annual proceedings of ACL, NAACL, and EACL, and the biennial COLING conference are the primary forums for work in this area. Related conferences include various proceedings of ACL Special Interest Groups (SIGs) such as the Conference on Natural Language Learning (CoNLL), as well as the conference on Empirical Methods in Natural Language Processing (EMNLP).

Research on speech recognition, understanding, and synthesis is presented at the annual INTERSPEECH conference, which is called the International Conference on Spoken Language Processing (ICSLP) and the European Conference on Speech Communication and Technology (EUROSPEECH) in alternating years, or the annual IEEE International Conference on Acoustics, Speech, and Signal Processing (IEEE ICASSP). Spoken language dialogue research is presented at these or at workshops like SIGDial.

Journals include *Computational Linguistics*, *Natural Language Engineering*, *Speech Communication*, *Computer Speech and Language*, the *IEEE Transactions on Audio, Speech & Language Processing* and the *ACM Transactions on Speech and Language Processing*.

Work on language processing from an Artificial Intelligence perspective can be found in the annual meetings of the American Association for Artificial Intelligence (AAAI), as well as the biennial International Joint Conference on Artificial Intelligence (IJCAI) meetings. Artificial intelligence journals that periodically feature work on speech and language processing include *Machine Learning*, *Journal of Machine Learning Research*, and the *Journal of Artificial Intelligence Research*.

There are a fair number of textbooks available covering various aspects of speech and language processing. Manning and Schütze (1999) (*Foundations of Statistical Language Processing*) focuses on statistical models of tagging, parsing, disambiguation, collocations, and other areas. Charniak (1993) (*Statistical Language Learning*) is an accessible, though older and less-extensive, introduction to similar material. Manning et al. (2008) focuses on information retrieval, text classification, and clustering. NLTK, the Natural Language Toolkit (Bird and Loper, 2004), is a suite of Python modules and data for natural language processing, together with a Natural Language Processing book based on the NLTK suite. Allen (1995) (*Natural Language Understanding*) provides extensive coverage of language processing from the AI perspective. Gazdar and Mellish (1989) (*Natural Language Processing in Lisp/Prolog*) covers especially automata, parsing, features, and unification and is available free online. Pereira and Shieber (1987) gives a Prolog-based introduction to parsing and interpretation. Russell and Norvig (2002) is an introduction to artificial intelligence that includes chapters on natural language processing. Partee et al. (1990) has a very broad coverage of mathematical linguistics. A historically significant collection of foundational papers can be found in Grosz et al. (1986) (*Readings in Natural Language Processing*).

Of course, a wide-variety of speech and language processing resources are now available on the Web. Pointers to these resources are maintained on the home-page for this book at:

`http://www.cs.colorado.edu/~martin/slp.html`.

Allen, J. (1995). *Natural Language Understanding*. Benjamin Cummings, Menlo Park, CA.

Backus, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zurch ACM-GAMM Conference. In *Information Processing: Proceedings of the International Conference on Information Processing, Paris*, pp. 125–132. UNESCO.

Berger, A., Della Pietra, S. A., and Della Pietra, V. J. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, *22*(1), 39–71.

Bird, S. and Loper, E. (2004). NLTK: The Natural Language Toolkit. In *Proceedings of the ACL 2004 demonstration session*, Barcelona, Spain, pp. 214–217.

Bledsoe, W. W. and Browning, I. (1959). Pattern recognition and reading by machine. In *1959 Proceedings of the Eastern Joint Computer Conference*, pp. 225–232. Academic, New York.

Bresnan, J. and Kaplan, R. M. (1982). Introduction: Grammars as mental representations of language. In Bresnan, J. (Ed.), *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. (1990). A statistical approach to machine translation. *Computational Linguistics*, *16*(2), 79–85.

Carlson, L., Marcu, D., and Okurowski, M. E. (2001). Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *Proceedings of SIGDIAL*.

Charniak, E. (1993). *Statistical Language Learning*. MIT Press.

Chomsky, N. (1956). Three models for the description of language. *IRI Transactions on Information Theory*, *2*(3), 113–124.

Chomsky, N. (1959). A review of B. F. Skinner's "Verbal Behavior". *Language*, *35*, 26–58.

Church, K. W. (1980). On memory limitations in natural language processing. Master's thesis, MIT. Distributed by the Indiana University Linguistics Club.

Cohen, P. R. and Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science*, *3*(3), 177–212.

Colmerauer, A. (1970). Les systèmes-q ou un formalisme pour analyser et synthétiser des phrase sur ordinateur. Internal publication 43, Département d'informatique de l'Université de Montréal†.

Colmerauer, A. (1975). Les grammaires de métamorphose GIA. Internal publication, Groupe Intelligence artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, Nov 1975. English version, Metamorphosis grammars. In L. Bolc, (Ed.), *Natural Language Communication with Computers, Lecture Notes in Computer Science 63*, Springer Verlag, Berlin, 1978, pp. 133–189.

Cullingford, R. E. (1981). SAM. In Schank, R. C. and Riesbeck, C. K. (Eds.), *Inside Computer Understanding: Five Programs plus Miniatures*, pp. 75–119. Lawrence Erlbaum, Hillsdale, NJ.

Davis, K. H., Biddulph, R., and Balashek, S. (1952). Automatic recognition of spoken digits. *Journal of the Acoustical Society of America*, *24*(6), 637–642.

Dejean, H. and Tjong Kim Sang, E. F. (2001). Introduction to the CoNLL-2001 shared task: Clause identification. In *Proceedings of CoNLL-2001*.

Fillmore, C. J. (1968). The case for case. In Bach, E. W. and Harms, R. T. (Eds.), *Universals in Linguistic Theory*, pp. 1–88. Holt, Rinehart & Winston, New York.

Francis, W. N. (1979). A tagged corpus – problems and prospects. In Greenbaum, S., Leech, G., and Svartvik, J. (Eds.), *Studies in English linguistics for Randolph Quirk*, pp. 192–209. Longman, London and New York.

Francis, W. N. and Kučera, H. (1982). *Frequency Analysis of English Usage*. Houghton Mifflin, Boston.

Gazdar, G. and Mellish, C. (1989). *Natural Language Processing in LISP*. Addison Wesley.

Grosz, B. J. (1977). The representation and use of focus in a system for understanding dialogs. In *IJCAI-77*, Cambridge, MA, pp. 67–76. Morgan Kaufmann. Reprinted in Grosz et al. (1986).

Grosz, B. J., Jones, K. S., and Webber, B. L. (Eds.). (1986). *Readings in Natural Language Processing*. Morgan Kaufmann, Los Altos, Calif.

Hajič, J. (1998). *Building a Syntactically Annotated Corpus: The Prague Dependency Treebank*, pp. 106–132. Karolinum, Prague/Praha.

Harris, Z. S. (1962). *String Analysis of Sentence Structure*. Mouton, The Hague.

Hobbs, J. R. (1978). Resolving pronoun references. *Lingua*, *44*, 311–338. Reprinted in Grosz et al. (1986).

Joshi, A. K. and Hopely, P. (1999). A parser from antiquity. In Kornai, A. (Ed.), *Extended Finite State Models of Language*, pp. 6–15. Cambridge University Press, Cambridge.

Kaplan, R. M. and Kay, M. (1981). Phonological rules and finite-state transducers. Paper presented at the Annual meeting of the Linguistics Society of America. New York.

Karttunen, L. (1999). Comments on Joshi. In Kornai, A. (Ed.), *Extended Finite State Models of Language*, pp. 16–18. Cambridge University Press, Cambridge.

Kay, M. (1979). Functional grammar. In *BLS-79*, Berkeley, CA, pp. 142–158.

Kilgarriff, A. and Palmer, M. (Eds.). (2000). *Computing and the Humanities: Special Issue on SENSEVAL*, Vol. 34. Kluwer.

Kintsch, W. (1974). *The Representation of Meaning in Memory*. Wiley, New York.

Kleene, S. C. (1951). Representation of events in nerve nets and finite automata. Tech. rep. RM-704, RAND Corporation. RAND Research Memorandum†.

Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In Shannon, C. and McCarthy, J. (Eds.), *Automata Studies*, pp. 3–41. Princeton University Press, Princeton, NJ.

Koenig, W., Dunn, H. K., Y., L., and Lacy (1946). The sound spectrograph. *Journal of the Acoustical Society of America*, *18*, 19–49.

Kučera, H. and Francis, W. N. (1967). *Computational analysis of present-day American English*. Brown University Press, Providence, RI.

Lehnert, W. G. (1977). A conceptual theory of question answering. In *IJCAI-77*, Cambridge, MA, pp. 158–164. Morgan Kaufmann.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, *19*(2), 313–330.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115–133. Reprinted in *Neurocomputing: Foundations of Research, ed. by J. A. Anderson and E Rosenfeld. MIT Press 1988*.

Merton, R. K. (1961). Singletons and multiples in scientific discovery. *American Philosophical Society Proceedings*, *105*(5), 470–486.

Miltsakaki, E., Prasad, R., Joshi, A. K., and Webber, B. L. (2004). The Penn Discourse Treebank. In *LREC-04*.

Mosteller, F. and Wallace, D. L. (1964). *Inference and Disputed Authorship: The Federalist*. Springer-Verlag, New York. 2nd Edition appeared in 1984 and was called *Applied Bayesian and Classical Inference*.

Naur, P., Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., van Wijnagaarden, A., and Woodger, M. (1960). Report on the algorithmic language ALGOL 60. *Communications of the ACM*, *3*(5), 299–314. Revised in CACM 6:1, 1-17, 1963.

Norman, D. A. and Rumelhart, D. E. (1975). *Explorations in Cognition*. Freeman, San Francisco, CA.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, *29*(1), 19–51.

Ogburn, W. F. and Thomas, D. S. (1922). Are inventions inevitable? A note on social evolution. *Political Science Quarterly*, *37*, 83–98.

Palmer, M., Fellbaum, C., Cotton, S., Delfs, L., and Dang, H. T. (2001). English tasks: All-words and verb lexical sample. In *Proceedings of SENSEVAL-2: Second International Workshop on Evaluating Word Sense Disambiguation Systems*, Toulouse, France.

Palmer, M., Kingsbury, P., and Gildea, D. (2005). The proposition bank: An annotated corpus of semantic roles.. *Computational Linguistics*, *31*(1), 71–106.

Partee, B. H., ter Meulen, A., and Wall, R. E. (1990). *Mathematical Methods in Linguistics*. Kluwer, Dordrecht.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, San Mateo, Ca.

Pereira, F. C. N. and Shieber, S. M. (1987). *Prolog and Natural-Language Analysis*, Vol. 10 of *CSLI Lecture Notes*. Chicago University Press, Chicago.

Pereira, F. C. N. and Warren, D. H. D. (1980). Definite clause grammars for language analysis— a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, *13*(3), 231–278.

Perrault, C. R. and Allen, J. (1980). A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics*, *6*(3-4), 167–182.

Quillian, M. R. (1968). Semantic memory. In Minsky, M. (Ed.), *Semantic Information Processing*, pp. 227–270. MIT Press, Cambridge, MA.

Rabiner, L. R. and Juang, B. (1993). *Fundamentals of Speech Recognition*. Prentice Hall, Englewood Cliffs, NJ.

Reeves, B. and Nass, C. (1996). *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*. Cambridge University Press, Cambridge.

Russell, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ. Second edition.

Schank, R. C. (1972). Conceptual dependency: A theory of natural language processing. *Cognitive Psychology*, *3*, 552–631.

Schank, R. C. and Albelson, R. P. (1977). *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum, Hillsdale, NJ.

Schank, R. C. and Riesbeck, C. K. (Eds.). (1981). *Inside Computer Understanding: Five Programs plus Miniatures*. Lawrence Erlbaum, Hillsdale, NJ.

Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, *3*, 417–457.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, *27*(3), 379–423. Continued in following volume.

Shieber, S. M. (1994). Lessons from a restricted Turing test. *Communications of the ACM*, *37*(6), 70–78.

Sidner, C. L. (1983). Focusing in the comprehension of definite anaphora. In Brady, M. and Berwick, R. C. (Eds.), *Computational Models of Discourse*, pp. 267–330. MIT Press, Cambridge, MA.

Simmons, R. F. (1973). Semantic networks: Their computation and use for understanding English sentences. In Schank, R. C. and Colby, K. M. (Eds.), *Computer Models of Thought and Language*, pp. 61–113. W.H. Freeman and Co., San Francisco.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *42*, 230–265. Read to the Society in 1936, but published in 1937. Correction in volume 43, 544–546.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*, 433–460.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.

Voorhees, E. M. and Tice, D. M. (1999). The TREC-8 question answering track evaluation. Proceedings of the TREC-8 Workshop.

Weizenbaum, J. (1966). ELIZA – A computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36–45.

Weizenbaum, J. (1976). *Computer Power and Human Reason: From Judgement to Calculation*. W.H. Freeman and Company, San Francisco.

Wilensky, R. (1983). *Planning and Understanding*. Addison-Wesley, Reading, MA.

Wilks, Y. (1975a). Preference semantics. In Keenan, E. L. (Ed.), *The Formal Semantics of Natural Language*, pp. 329–350. Cambridge Univ. Press, Cambridge.

Wilks, Y. (1975b). A preferential, pattern-seeking, semantics for natural language inference. *Artificial Intelligence*, *6*(1), 53–74.

Winograd, T. (1972). Understanding natural language. *Cognitive Psychology*, *3*(1), 1–191. Reprinted as a book by Academic Press, 1972.

Woods, W. A. (1967). *Semantics for a Question-Answering System*. Ph.D. thesis, Harvard University.

Woods, W. A. (1973). Progress in natural language understanding. In *Proceedings of AFIPS National Conference*, pp. 441–450.

Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition. Daniel Jurafsky & James H. Martin. Copyright © 2007, All rights reserved. Draft of October 11, 2007. Do not cite without permission.

2 REGULAR EXPRESSIONS AND AUTOMATA

User$_1$:	*Men are all alike.*
ELIZA$_1$:	*IN WHAT WAY*
User$_2$:	*They're always bugging us about something or other.*
ELIZA$_2$:	*CAN YOU THINK OF A SPECIFIC EXAMPLE*
User$_3$:	*Well, my boyfriend made me come here.*
ELIZA$_3$:	*YOUR BOYFRIEND MADE YOU COME HERE*
User$_4$:	*He says I'm depressed much of the time.*
ELIZA$_4$:	*I AM SORRY TO HEAR YOU ARE DEPRESSED.*

Weizenbaum (1966)

Imagine that you have become a passionate fan of woodchucks. Desiring more information on this celebrated woodland creature, you turn to your favorite Web browser and type in *woodchuck*. Your browser returns a few sites. You have a flash of inspiration and type in *woodchucks*. This time you discover "interesting links to woodchucks and lemurs" and "all about Vermont's unique, endangered species". Instead of having to do this search twice, you would have rather typed one search command specifying something like *woodchuck with an optional final s*. Or perhaps you might want to search for all the prices in some document; you might want to see all strings that look like *$199* or *$25* or *$24.99*. In this chapter we introduce the **regular expression**, the standard notation for characterizing text sequences. The regular expression is used for specifying text strings in situations like this Web-search example, and in other information retrieval applications, but also plays an important role in word-processing, computation of frequencies from corpora, and other such tasks.

After we have defined regular expressions, we show how they can be implemented via the **finite-state automaton**. The finite-state automaton is not only the mathematical device used to implement regular expressions, but also one of the most significant tools of computational linguistics. Variations of automata such as finite-state transducers, Hidden Markov Models, and *N*-gram grammars are important components of applications that we will introduce in later chapters, including speech recognition and synthesis, machine translation, spell-checking, and information-extraction.

2.1 Regular Expressions

SIR ANDREW: Her C's, her U's and her T's: why that?
Shakespeare, *Twelfth Night*

REGULAR EXPRESSION

One of the unsung successes in standardization in computer science has been the **regular expression** (**RE**), a language for specifying text search strings. The regular expression languages used for searching texts in UNIX (vi, Perl, Emacs, grep), Microsoft Word (version 6 and beyond), and WordPerfect are almost identical, and many RE features exist in the various Web search engines. Besides this practical use, the regular expression is an important theoretical tool throughout computer science and linguistics.

STRINGS

A regular expression (first developed by Kleene (1956) but see the History section for more details) is a formula in a special language that is used for specifying simple classes of **strings**. A string is a sequence of symbols; for the purpose of most text-based search techniques, a string is any sequence of alphanumeric characters (letters, numbers, spaces, tabs, and punctuation). For these purposes a space is just a character like any other, and we represent it with the symbol ␣.

Formally, a regular expression is an algebraic notation for characterizing a set of strings. Thus they can be used to specify search strings as well as to define a language in a formal way. We will begin by talking about regular expressions as a way of specifying searches in texts, and proceed to other uses. Section 2.3 shows that the use of just three regular expression operators is sufficient to characterize strings, but we use the more convenient and commonly-used regular expression syntax of the Perl language throughout this section. Since common text-processing programs agree on most of the syntax of regular expressions, most of what we say extends to all UNIX, Microsoft Word, and WordPerfect regular expressions. Appendix A shows the few areas where these programs differ from the Perl syntax.

CORPUS

Regular expression search requires a **pattern** that we want to search for, and a **corpus** of texts to search through. A regular expression search function will search through the corpus returning all texts that contain the pattern. In an information retrieval (IR) system such as a Web search engine, the texts might be entire documents or Web pages. In a word-processor, the texts might be individual words, or lines of a document. In the rest of this chapter, we will use this last paradigm. Thus when we give a search pattern, we will assume that the search engine returns the *line of the document* returned. This is what the UNIX `grep` command does. We will underline the exact part of the pattern that matches the regular expression. A search can be designed to return all matches to a regular expression or only the first match. We will show only the first match.

2.1.1 Basic Regular Expression Patterns

The simplest kind of regular expression is a sequence of simple characters. For example, to search for *woodchuck*, we type `/woodchuck/`. So the regular expression `/Buttercup/` matches any string containing the substring *Buttercup*, for example the line *I'm called little Buttercup*) (recall that we are assuming a search application that returns entire lines). From here on we will put slashes around each regular expression to make it clear what is a regular expression and what is a pattern. We use the slash since this is the notation used by Perl, but the slashes are *not* part of the regular expressions.

The search string can consist of a single character (like `/!/`) or a sequence of characters (like `/urgl/`); The *first* instance of each match to the regular expression is underlined below (although a given application might choose to return more than just the first instance):

RE	Example Patterns Matched
`/woodchucks/`	"interesting links to woodchucks and lemurs"
`/a/`	"Mary Ann stopped by Mona's"
`/Claire␣says,/`	"Dagmar, my gift please," Claire says,"
`/DOROTHY/`	"SURRENDER DOROTHY"
`/!/`	"You've left the burglar behind again!" said Nori

Regular expressions are **case sensitive**; lowercase `/s/` is distinct from uppercase `/S/` (`/s/` matches a lower case *s* but not an uppercase *S*). This means that the pattern `/woodchucks/` will not match the string *Woodchucks*. We can solve this problem with the use of the square braces `[` and `]`. The string of characters inside the braces specify a **disjunction** of characters to match. For example Fig. 2.1 shows that the pattern `/[wW]/` matches patterns containing either *w* or *W*.

RE	Match	Example Patterns
`/[wW]oodchuck/`	Woodchuck or woodchuck	"Woodchuck"
`/[abc]/`	'a', 'b', *or* 'c'	"In uomini, in soldati"
`/[1234567890]/`	any digit	"plenty of 7 to 5"

Figure 2.1 The use of the brackets `[]` to specify a disjunction of characters.

The regular expression `/[1234567890]/` specified any single digit. While classes of characters like digits or letters are important building blocks in expressions, they can get awkward (e.g., it's inconvenient to specify

`/[ABCDEFGHIJKLMNOPQRSTUVWXYZ]/`

RANGE

to mean "any capital letter"). In these cases the brackets can be used with the dash (`-`) to specify any one character in a **range**. The pattern `/[2-5]/` specifies any one of the characters *2*, *3*, *4*, or *5*. The pattern `/[b-g]/` specifies one of the characters *b*, *c*, *d*, *e*, *f*, or *g*. Some other examples:

RE	Match	Example Patterns Matched
`/[A-Z]/`	an uppercase letter	"we should call it 'Drenched Blossoms'"
`/[a-z]/`	a lowercase letter	"my beans were impatient to be hoed!"
`/[0-9]/`	a single digit	"Chapter 1: Down the Rabbit Hole"

Figure 2.2 The use of the brackets `[]` plus the dash `-` to specify a range.

The square braces can also be used to specify what a single character *cannot* be, by use of the caret `^`. If the caret `^` is the first symbol after the open square brace `[`, the resulting pattern is negated. For example, the pattern /[^a]/ matches any single character (including special characters) except *a*. This is only true when the caret is the first symbol after the open square brace. If it occurs anywhere else, it usually stands for a caret; Fig. 2.3 shows some examples.

RE	Match (single characters)	Example Patterns Matched
[^A-Z]	not an uppercase letter	"Oyfn pripetchik"
[^Ss]	neither 'S' nor 's'	"I have no exquisite reason for't"
[^\.]	not a period	"our resident Djinn"
[e^]	either 'e' or '^'	"look up ^ now"
a^b	the pattern 'a^b'	"look up a^ b now"

Figure 2.3 Uses of the caret ^ for negation or just to mean ^ .

The use of square braces solves our capitalization problem for *woodchucks*. But we still haven't answered our original question; how do we specify both *woodchuck* and *woodchucks*? We can't use the square brackets, because while they allow us to say "s or S", they don't allow us to say "s or nothing". For this we use the question-mark /?/, which means "the preceding character or nothing", as shown in Fig. 2.4.

RE	Match	Example Patterns Matched
woodchucks?	woodchuck or woodchucks	"woodchuck"
colou?r	color or colour	"colour"

Figure 2.4 The question-mark ? marks optionality of the previous expression.

We can think of the question-mark as meaning "zero or one instances of the previous character". That is, it's a way of specifying how many of something that we want. So far we haven't needed to specify that we want more than one of something. But sometimes we need regular expressions that allow repetitions of things. For example, consider the language of (certain) sheep, which consists of strings that look like the following:

baa!
baaa!
baaaa!
baaaaa!
baaaaaa!
...

KLEENE *

This language consists of strings with a *b*, followed by at least two *a*s, followed by an exclamation point. The set of operators that allow us to say things like "some number of *a*s" are based on the asterisk or *, commonly called the **Kleene *** (pronounced "cleany star"). The Kleene star means "zero or more occurrences of the immediately previous character or regular expression". So /a*/ means "any string of zero or more *a*s". This will match *a* or *aaaaaa* but it will also match *Off Minor*, since the string *Off Minor* has zero *a*s. So the regular expression for matching one or more *a* is /aa*/, meaning one *a* followed by zero or more *a*s. More complex patterns can also be repeated. So `/[ab]*/` means “zero or more *a*s or *b*s” (not “zero or more right square braces”). This will match strings like *aaaa* or *ababab* or *bbbb*.

We now know enough to specify part of our regular expression for prices: multiple digits. Recall that the regular expression for an individual digit was `/[0-9]/`. So the regular expression for an integer (a string of digits) is `/[0-9][0-9]*/`. (Why isn’t it just `/[0-9]*/`?)

KLEENE +

Sometimes it’s annoying to have to write the regular expression for digits twice, so there is a shorter way to specify “at least one” of some character. This is the **Kleene +**, which means “one or more of the previous character”. Thus the expression `/[0-9]+/` is the normal way to specify “a sequence of digits”. There are thus two ways to specify the sheep language: `/baaa*!/` or `/baa+!/`.

One very important special character is the period (`/./`), a **wildcard** expression that matches any single character (*except* a carriage return):

RE	Match	Example Patterns
`/beg.n/`	any character between *beg* and *n*	begin, beg’n, begun

Figure 2.5 The use of the period `.` to specify any character.

The wildcard is often used together with the Kleene star to mean “any string of characters”. For example suppose we want to find any line in which a particular word, for example *aardvark*, appears twice. We can specify this with the regular expression `/aardvark.*aardvark/`.

ANCHORS

Anchors are special characters that anchor regular expressions to particular places in a string. The most common anchors are the caret `^` and the dollar-sign `$`. The caret `^` matches the start of a line. The pattern `/^The/` matches the word *The* only at the start of a line. Thus there are three uses of the caret `^`: to match the start of a line, as a negation inside of square brackets, and just to mean a caret. (What are the contexts that allow Perl to know which function a given caret is supposed to have?) The dollar sign `$` matches the end of a line. So the pattern `␣$` is a useful pattern for matching a space at the end of a line, and `/^The dog\.$/` matches a line that contains only the phrase *The dog.* (We have to use the backslash here since we want the `.` to mean “period” and not the wildcard.)

There are also two other anchors: `\b` matches a word boundary, while `\B` matches a non-boundary. Thus `/\bthe\b/` matches the word *the* but not the word *other*. More technically, Perl defines a word as any sequence of digits, underscores or letters; this is based on the definition of “words” in programming languages like Perl or C. For example, `/\b99\b/` will match the string *99* in *There are 99 bottles of beer on the wall* (because 99 follows a space) but not *99* in *There are 299 bottles of beer on the wall* (since 99 follows a number). But it will match *99* in *$99* (since *99* follows a dollar sign ($), which is not a digit, underscore, or letter).

2.1.2 Disjunction, Grouping, and Precedence

DISJUNCTION

Suppose we need to search for texts about pets; perhaps we are particularly interested in cats and dogs. In such a case we might want to search for either the string *cat* or the string *dog*. Since we can't use the square-brackets to search for "cat or dog" (why not?) we need a new operator, the **disjunction** operator, also called the **pipe** symbol `|`. The pattern `/cat|dog/` matches either the string `cat` or the string `dog`.

PRECEDENCE

Sometimes we need to use this disjunction operator in the midst of a larger sequence. For example, suppose I want to search for information about pet fish for my cousin David. How can I specify both *guppy* and *guppies*? We cannot simply say `/guppy|ies/`, because that would match only the strings *guppy* and *ies*. This is because sequences like `guppy` take **precedence** over the disjunction operator `|`. In order to make the disjunction operator apply only to a specific pattern, we need to use the parenthesis operators `(` and `)`. Enclosing a pattern in parentheses makes it act like a single character for the purposes of neighboring operators like the pipe `|` and the Kleene`*`. So the pattern `/gupp(y|ies)/` would specify that we meant the disjunction only to apply to the suffixes `y` and `ies`.

The parenthesis operator `(` is also useful when we are using counters like the Kleene`*`. Unlike the `|` operator, the Kleene`*` operator applies by default only to a single character, not a whole sequence. Suppose we want to match repeated instances of a string. Perhaps we have a line that has column labels of the form *Column 1 Column 2 Column 3*. The expression `/Column␣[0-9]+␣*/` will not match any column; instead, it will match a column followed by any number of spaces! The star here applies only to the space `␣` that precedes it, not the whole sequence. With the parentheses, we could write the expression `/(Column␣[0-9]+␣*)*/` to match the word *Column*, followed by a number and optional spaces, the whole pattern repeated any number of times.

OPERATOR PRECEDENCE

This idea that one operator may take precedence over another, requiring us to sometimes use parentheses to specify what we mean, is formalized by the **operator precedence hierarchy** for regular expressions. The following table gives the order of RE operator precedence, from highest precedence to lowest precedence:

Parenthesis	`()`
Counters	`* + ? {}`
Sequences and anchors	`the ^my end$`
Disjunction	`|`

Thus, because counters have a higher precedence than sequences, `/the*/` matches *theeeee* but not *thethe*. Because sequences have a higher precedence than disjunction, `/the|any/` matches *the* or *any* but not *theny*.

GREEDY

Patterns can be ambiguous in another way. Consider the expression `/[a-z]*/` when matching against the text *once upon a time*. Since `/[a-z]*/` matches zero or more letters, this expression could match nothing, or just the first letter *o*, or *on*, or *onc*, or *once*. In these cases regular expressions always match the *largest* string they can; we say that patterns are **greedy**, expanding to cover as much of a string as they can.

2.1.3 A Simple Example

Suppose we wanted to write a RE to find cases of the English article *the*. A simple (but incorrect) pattern might be:

```
/the/
```

One problem is that this pattern will miss the word when it begins a sentence and hence is capitalized (i.e., *The*). This might lead us to the following pattern:

```
/[tT]he/
```

But we will still incorrectly return texts with `the` embedded in other words (e.g., *other* or *theology*). So we need to specify that we want instances with a word boundary on both sides:

```
/\b[tT]he\b/
```

Suppose we wanted to do this without the use of `/\b/`? We might want this since `/\b/` won't treat underscores and numbers as word boundaries; but we might want to find *the* in some context where it might also have underlines or numbers nearby (*the_* or *the25*). We need to specify that we want instances in which there are no alphabetic letters on either side of the *the*:

```
/[^a-zA-Z][tT]he[^a-zA-Z]/
```

But there is still one more problem with this pattern: it won't find the word *the* when it begins a line. This is because the regular expression `[^a-zA-Z]`, which we used to avoid embedded *the*s, implies that there must be some single (although non-alphabetic) character before the `the`. We can avoid this by specifying that before the *the* we require *either* the beginning-of-line or a non-alphabetic character, and the same at the end of the line:

```
/(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/
```

FALSE POSITIVES

FALSE NEGATIVES

The process we just went through was based on fixing two kinds of errors: **false positives**, strings that we incorrectly matched like *other* or *there*, and **false negatives**, strings that we incorrectly missed, like *The*. Addressing these two kinds of errors comes up again and again in building and improving speech and language processing systems. Reducing the error rate for an application thus involves two antagonistic efforts:

- Increasing **accuracy** (minimizing false positives)
- Increasing **coverage** (minimizing false negatives).

2.1.4 A More Complex Example

Let's try out a more significant example of the power of REs. Suppose we want to build an application to help a user buy a computer on the Web. The user might want "any PC with more than 500 MHz and 32 Gb of disk space for less than $1000". In order to do this kind of retrieval we will first need to be able to look for expressions like *500 MHz* or *32 Gb* or *Compaq* or *Mac* or *$999.99*. In the rest of this section we'll work out some simple regular expressions for this task.

First, let's complete our regular expression for prices. Here's a regular expression for a dollar sign followed by a string of digits. Note that Perl is smart enough to realize that $ here doesn't mean end-of-line; how might it know that?

```
/$[0-9]+/
```

Now we just need to deal with fractions of dollars. We'll add a decimal point and two digits afterwards:

```
/$[0-9]+\.[0-9][0-9]/
```

This pattern only allows *$199.99* but not *$199*. We need to make the cents optional, and make sure we're at a word boundary:

```
/\b$[0-9]+(\.[0-9][0-9])?\b/
```

How about specifications for processor speed (in megahertz = MHz or gigahertz = GHz)? Here's a pattern for that:

```
/\b[0-9]+␣*(MHz|[Mm]egahertz|GHz|[Gg]igahertz)\b/
```

Note that we use `/␣*/` to mean "zero or more spaces", since there might always be extra spaces lying around. Dealing with disk space (in Gb = gigabytes), or memory size (in Mb = megabytes or Gb = gigabytes), we need to allow for optional gigabyte fractions again (*5.5 Gb*). Note the use of `?` for making the final `s` optional:

```
/\b[0-9]+␣*(Mb|[Mm]egabytes?)\b/
/\b[0-9](\.[0-9]+)?␣*(Gb|[Gg]igabytes?)\b/
```

Finally, we might want some simple patterns to specify operating systems and vendors:

```
/\b(Win95|Win98|WinNT|Windows␣*(NT|95|98|2000)?)\b/
/\b(Mac|Macintosh|Apple)\b/
```

2.1.5 Advanced Operators

RE	Expansion	Match	Example Patterns
`\d`	`[^0-9]` → `[0-9]`	any digit	Party␣of␣5
`\D`	`[^0-9]`	any non-digit	Blue␣moon
`\w`	`[a-zA-Z0-9_]`	any alphanumeric or underscore	Daiyu
`\W`	`[^\w]`	a non-alphanumeric	!!!!
`\s`	`[␣\r\t\n\f]`	whitespace (space, tab)	
`\S`	`[^\s]`	Non-whitespace	in␣Concord

Figure 2.6 Aliases for common sets of characters.

There are also some useful advanced regular expression operators. Fig. 2.6 shows some useful aliases for common ranges, which can be used mainly to save typing. Besides the Kleene * and Kleene +, we can also use explicit numbers as counters, by enclosing them in curly brackets. The regular expression `/{3}/` means "exactly 3 occurrences of the previous character or expression". So `/a\.{24}z/` will match *a* followed by 24 dots followed by *z* (but not *a* followed by 23 or 25 dots followed by a *z*).

A range of numbers can also be specified; so `/{n,m}/` specifies from n to m occurrences of the previous char or expression, while `/{n,}/` means at least n occurrences of the previous expression. REs for counting are summarized in Figure 2.7.

RE	Match
`*`	zero or more occurrences of the previous char or expression
`+`	one or more occurrences of the previous char or expression
`?`	exactly zero or one occurrence of the previous char or expression
`{n}`	*n* occurrences of the previous char or expression
`{n,m}`	from *n* to *m* occurrences of the previous char or expression
`{n,}`	at least *n* occurrences of the previous char or expression

Figure 2.7 Regular expression operators for counting.

NEWLINE

Finally, certain special characters are referred to by special notation based on the backslash (\). The most common of these are the **newline** character `\n` and the **tab** character `\t`. To refer to characters that are special themselves (like `.`, `*`, `[`, and `\`), precede them with a backslash, (i.e., `/\./`, `/\*/`, `/\[/`, and `/\\/`).

RE	Match	Example Patterns Matched
`\*`	an asterisk "*"	"K*A*P*L*A*N"
`\.`	a period "."	"Dr. Livingston, I presume"
`\?`	a question mark	"Why don't they come and lend a hand?"
`\n`	a newline	
`\t`	a tab	

Figure 2.8 Some characters that need to be backslashed.

The reader should consult Appendix A for further details of regular expressions, and especially for the differences between regular expressions in Perl, UNIX, and Microsoft Word.

2.1.6 Regular Expression Substitution, Memory, and ELIZA

SUBSTITUTION

An important use of regular expressions is in **substitutions**. For example, the Perl substitution operator `s/regexp1/pattern/` allows a string characterized by a regular expression to be replaced by another string:

```
s/colour/color/
```

It is often useful to be able to refer to a particular subpart of the string matching the first pattern. For example, suppose we wanted to put angle brackets around all integers in a text, changing e.g., *the 35 boxes* to *the <35> boxes*. We'd like a way to refer back to the integer we've found so that we can easily add the brackets. To do this, we put parentheses `(` and `)` around the first pattern, and use the **number** operator `\1` in the second pattern to refer back. Here's how it looks:

```
s/([0-9]+)/<\1>/
```

The parenthesis and number operators can also be used to specify that a certain string or expression must occur twice in the text. For example, suppose we are looking for the pattern "the Xer they were, the Xer they will be", where we want to constrain the two X's to be the same string. We do this by surrounding the first X with the parenthesis operator, and replacing the second X with the number operator `\1`, as follows:

```
/the (.*)er they were, the \1er they will be/
```

Here the `\1` will be replaced by whatever string matched the first item in parentheses. So this will match *The bigger they were, the bigger they will be* but not *The bigger they were, the faster they will be*.

The number operator can be used with other numbers: if you match two different sets of parenthesis, `\2` means whatever matched the *second* set. For example

```
/the (.*)er they (.*), the \1er they \2/
```

REGISTERS

will match *The bigger they were, the bigger they were* but not *The bigger they were, the bigger they will be*. These numbered memories are called **registers** (e.g. register 1, register 2, register 3, etc). This memory feature is not part of every regular expression language and is often considered an "extended" feature of regular expressions.

Substitutions using memory are very useful in implementing a simple natural-language understanding program like ELIZA (Weizenbaum, 1966). Recall that ELIZA simulated a Rogerian psychologist and could carry on conversations with the user like the following:

User$_1$: Men are all alike.
ELIZA$_1$: IN WHAT WAY
User$_2$: They're always bugging us about something or other.
ELIZA$_2$: CAN YOU THINK OF A SPECIFIC EXAMPLE
User$_3$: Well, my boyfriend made me come here.
ELIZA$_3$: YOUR BOYFRIEND MADE YOU COME HERE
User$_4$: He says I'm depressed much of the time.
ELIZA$_4$: I AM SORRY TO HEAR YOU ARE DEPRESSED.

Eliza worked by having a cascade of regular expression substitutions that each matched some part of the input lines and changed them. The first substitutions changed all instances of *my* to *YOUR*, and *I'm* to *YOU ARE*, and so on. The next set of substitutions looked for relevant patterns in the input and created an appropriate output; here are some examples:

```
s/.* YOU ARE (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* YOU ARE (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
```

```
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

Since multiple substitutions could apply to a given input, substitutions were assigned a rank and were applied in order. Creation of such patterns is addressed in Exercise 2.2.

2.2 Finite-State Automata

FINITE-STATE AUTOMATON

FSA

REGULAR LANGUAGE

The regular expression is more than just a convenient metalanguage for text searching. First, a regular expression is one way of describing a **finite-state automaton** (**FSA**). Finite-state automata are the theoretical foundation of a good deal of the computational work we will describe in this book. Any regular expression can be implemented as a finite-state automaton (except regular expressions that use the memory feature; more on this later). Symmetrically, any finite-state automaton can be described with a regular expression. Second, a regular expression is one way of characterizing a particular kind of formal language called a **regular language**. Both regular expressions and finite-state automata can be used to describe regular languages. A third equivalent method of characterizing the regular languages, the **regular grammar**, will be introduced in Ch. 15. The relation among these four theoretical constructions is sketched out in Fig. 2.9.

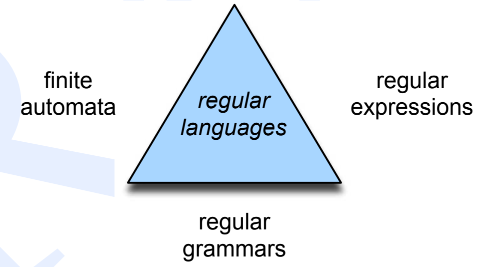


Figure 2.9 Finite automata, regular expressions, and regular grammars are all equivalent ways of describing regular languages.

This section will begin by introducing finite-state automata for some of the regular expressions from the last section, and then suggest how the mapping from regular expressions to automata proceeds in general. Although we begin with their use for implementing regular expressions, FSAs have a wide variety of other uses that we will explore in this chapter and the next.

2.2.1 Using an FSA to Recognize Sheeptalk

After a while, with the parrot's help, the Doctor got to learn the language of the animals so well that he could talk to them himself and understand everything they said.

Hugh Lofting, *The Story of Doctor Dolittle*

Let's begin with the "sheep language" we discussed previously. Recall that we defined the sheep language as any string from the following (infinite) set:

baa!
baaa!
baaaa!
baaaaa!
baaaaaa!
...

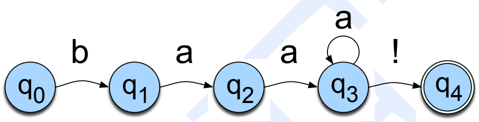


Figure 2.10 A finite-state automaton for talking sheep.

AUTOMATON

STATES

START STATE

The regular expression for this kind of "sheeptalk" is `/baa+!/`. Fig. 2.10 shows an **automaton** for modeling this regular expression. The automaton (i.e., machine, also called **finite automaton**, **finite-state automaton**, or **FSA**) recognizes a set of strings, in this case the strings characterizing sheep talk, in the same way that a regular expression does. We represent the automaton as a directed graph: a finite set of vertices (also called nodes), together with a set of directed links between pairs of vertices called arcs. We'll represent vertices with circles and arcs with arrows. The automaton has five **states**s, which are represented by nodes in the graph. State 0 is the **start state**. In our examples state 0 will generally be the start state; to mark another state as the start state we can add an incoming arrow to the start state. State 4 is the **final state** or **accepting state**, which we represent by the double circle. It also has four **transitions**, which we represent by arcs in the graph.

The FSA can be used for recognizing (we also say **accepting**) strings in the following way. First, think of the input as being written on a long tape broken up into cells, with one symbol written in each cell of the tape, as in Fig. 2.11.

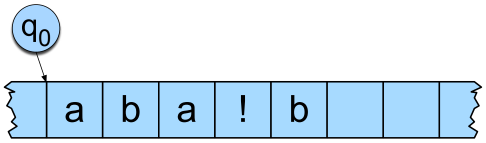


Figure 2.11 A tape with cells.

The machine starts in the start state (q_0), and iterates the following process: Check the next letter of the input. If it matches the symbol on an arc leaving the current state, then cross that arc, move to the next state, and also advance one symbol in the input. If we are in the accepting state (q_4) when we run out of input, the machine has successfully recognized an instance of sheeptalk. If the machine never gets to the final state, either because it runs out of input, or it gets some input that doesn't match an arc (as in Fig. 2.11), or if it just happens to get stuck in some non-final state, we say the machine **rejects** or fails to accept an input.

REJECTS

STATE-TRANSITION TABLE

We can also represent an automaton with a **state-transition table**. As in the graph notation, the state-transition table represents the start state, the accepting states, and what transitions leave each state with which symbols. Here's the state-transition table for the FSA of Figure 2.10.

	Input		
State	b	a	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4:	∅	∅	∅

Figure 2.12 The state-transition table for the FSA of Figure 2.10.

We've marked state 4 with a colon to indicate that it's a final state (you can have as many final states as you want), and the ∅ indicates an illegal or missing transition. We can read the first row as "if we're in state 0 and we see the input **b** we must go to state 1. If we're in state 0 and we see the input **a** or **!**, we fail".

More formally, a finite automaton is defined by the following five parameters:

$Q = q_0 q_1 q_2 \ldots q_{N-1}$	a finite set of N **states**
Σ	a finite **input alphabet** of symbols
q_0	the **start state**
F	the set of **final states**, $F \subseteq Q$
$\delta(q,i)$	the **transition function** or transition matrix between states. Given a state $q \in Q$ and an input symbol $i \in \Sigma$, $\delta(q,i)$ returns a new state $q' \in Q$. δ is thus a relation from $Q \times \Sigma$ to Q;

For the sheeptalk automaton in Fig. 2.10, $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b, !\}$, $F = \{q_4\}$, and $\delta(q,i)$ is defined by the transition table in Fig. 2.12.

DETERMINISTIC

Figure 2.13 presents an algorithm for recognizing a string using a state-transition table. The algorithm is called D-RECOGNIZE for "deterministic recognizer". A **deterministic** algorithm is one that has no choice points; the algorithm always knows what to do for any input. The next section will introduce non-deterministic automata that must make decisions about which states to move to.

D-RECOGNIZE takes as input a tape and an automaton. It returns *accept* if the string it is pointing to on the tape is accepted by the automaton, and *reject* otherwise. Note that since D-RECOGNIZE assumes it is already pointing at the string to be checked, its task is only a subpart of the general problem that we often use regular expressions for, finding a string in a corpus. (The general problem is left as an exercise to the reader in Exercise 2.9.)

D-RECOGNIZE begins by setting the variable *index* to the beginning of the tape, and *current-state* to the machine's initial state. D-RECOGNIZE then enters a loop that drives the rest of the algorithm. It first checks whether it has reached the end of its input. If so, it either accepts the input (if the current state is an accept state) or rejects the input (if not).

If there is input left on the tape, D-RECOGNIZE looks at the transition table to decide which state to move to. The variable *current-state* indicates which row of the table to consult, while the current symbol on the tape indicates which column of the table to consult. The resulting transition-table cell is used to update the variable *current-state* and *index* is incremented to move forward on the tape. If the transition-table cell is empty then the machine has nowhere to go and must reject the input.

```
function D-RECOGNIZE(tape, machine) returns accept or reject

  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
   if End of input has been reached then
     if current-state is an accept state then
       return accept
     else
       return reject
   elsif transition-table[current-state,tape[index]] is empty then
       return reject
   else
       current-state ← transition-table[current-state,tape[index]]
       index ← index + 1
  end
```

Figure 2.13 An algorithm for deterministic recognition of FSAs. This algorithm returns *accept* if the entire string it is pointing at is in the language defined by the FSA, and *reject* if the string is not in the language.

Figure 2.14 traces the execution of this algorithm on the sheep language FSA given the sample input string *baaa!*.

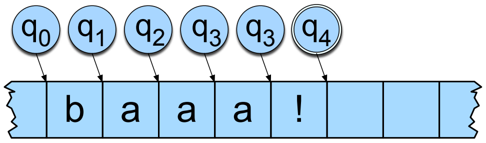


Figure 2.14 Tracing the execution of FSA #1 on some sheeptalk.

Before examining the beginning of the tape, the machine is in state q_0. Finding a *b* on input tape, it changes to state q_1 as indicated by the contents of *transition-table*$[q_0,b]$ in Fig. 2.12 on page 13. It then finds an *a* and switches to state q_2, another *a* puts it in state q_3, a third *a* leaves it in state q_3, where it reads the "!", and switches to state q_4. Since there is no more input, the `End of input` condition at the beginning of the loop is satisfied for the first time and the machine halts in q_4. State q_4 is an accepting state, and so the machine has accepted the string *baaa!* as a sentence in the sheep language.

FAIL STATE

The algorithm will fail whenever there is no legal transition for a given combination of state and input. The input *abc* will fail to be recognized since there is no legal transition out of state q_0 on the input a, (i.e., this entry of the transition table in Fig. 2.12 on page 13 has a $\emptyset$). Even if the automaton had allowed an initial *a* it would have certainly failed on *c*, since *c* isn't even in the sheeptalk alphabet! We can think of these "empty" elements in the table as if they all pointed at one "empty" state, which we might call the **fail state** or **sink state**. In a sense then, we could view any machine with empty transitions *as if* we had augmented it with a fail state, and drawn in all the extra arcs, so we always had somewhere to go from any state on any possible input. Just for completeness, Fig. 2.15 shows the FSA from Figure 2.10 with the fail state q_F filled in.

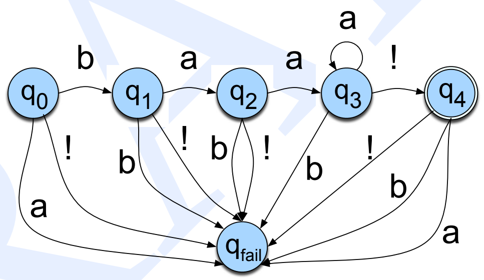


Figure 2.15 Adding a fail state to Fig. 2.10.

2.2.2 Formal Languages

We can use the same graph in Fig. 2.10 as an automaton for GENERATING sheeptalk. If we do, we would say that the automaton starts at state q_0, and crosses arcs to new states, printing out the symbols that label each arc it follows. When the automaton gets to the final state it stops. Notice that at state 3, the automaton has to chose between printing out a **!** and going to state 4, or printing out an **a** and returning to state 3. Let's say for now that we don't care how the machine makes this decision; maybe it flips a coin. For now, we don't care which exact string of sheeptalk we generate, as long as it's a string captured by the regular expression for sheeptalk above.

> **Formal Language:** A model which can both generate and recognize all and only the strings of a formal language acts as a *definition* of the formal language.

FORMAL LANGUAGE
ALPHABET

A **formal language** is a set of strings, each string composed of symbols from a finite symbol-set called an **alphabet** (the same alphabet used above for defining an automaton!). The alphabet for the sheep language is the set $\Sigma = \{a, b, !\}$. Given a model m (such as a particular FSA), we can use $L(m)$ to mean "the formal language characterized by m". So the formal language defined by our sheeptalk automaton m in Fig. 2.10 (and Fig. 2.12) is the infinite set:

$$L(m) = \{baa!, baaa!, baaaa!, baaaaa!, baaaaaa!, \ldots\} \tag{2.1}$$

NATURAL LANGUAGES

The usefulness of an automaton for defining a language is that it can express an infinite set (such as this one above) in a closed form. Formal languages are not the same as **natural languages**, which are the kind of languages that real people speak. In fact, a formal language may bear no resemblance at all to a real language (e.g., a formal language can be used to model the different states of a soda machine). But we often use a formal language to model part of a natural language, such as parts of the phonology, morphology, or syntax. The term **generative grammar** is sometimes used in linguistics to mean a grammar of a formal language; the origin of the term is this use of an automaton to define a language by generating all possible strings.

2.2.3 Another Example

In the previous examples our formal alphabet consisted of letters; but we can also have a higher level alphabet consisting of words. In this way we can write finite-state automata that model facts about word combinations. For example, suppose we wanted to build an FSA that modeled the subpart of English dealing with amounts of money. Such a formal language would model the subset of English consisting of phrases like *ten cents*, *three dollars*, *one dollar thirty-five cents* and so on.

We might break this down by first building just the automaton to account for the numbers from 1 to 99, since we'll need them to deal with cents. Fig. 2.16 shows this.

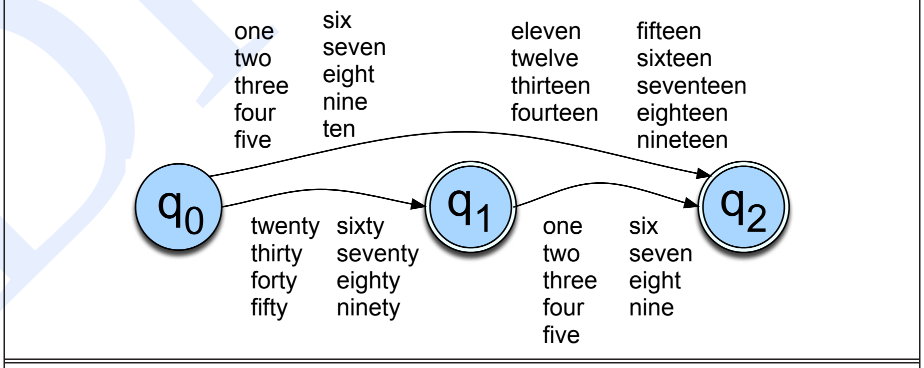


Figure 2.16 An FSA for the words for English numbers 1–99.

We could now add *cents* and *dollars* to our automaton. Fig. 2.17 shows a simple version of this, where we just made two copies of the automaton in Fig. 2.16 and appended the words *cents* and *dollars*.

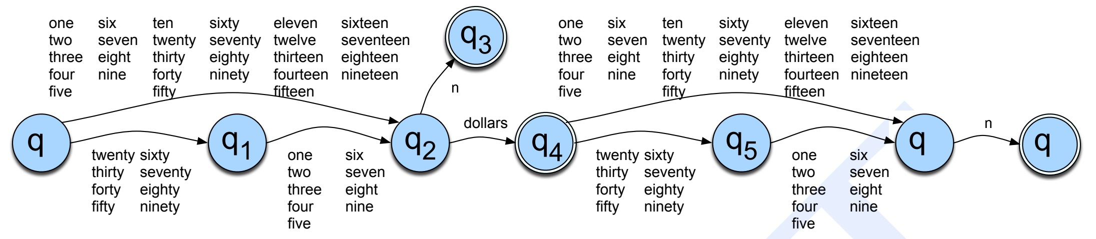


Figure 2.17 FSA for the simple dollars and cents.

We would now need to add in the grammar for different amounts of dollars; including higher numbers like *hundred*, *thousand*. We'd also need to make sure that the nouns like *cents* and *dollars* are singular when appropriate (*one cent*, *one dollar*), and plural when appropriate (*ten cents*, *two dollars*). This is left as an exercise for the reader (Exercise 2.3). We can think of the FSAs in Fig. 2.16 and Fig. 2.17 as simple grammars of parts of English. We will return to grammar-building in Part II of this book, particularly in Ch. 12.

2.2.4 Non-Deterministic FSAs

Let's extend our discussion now to another class of FSAs: **non-deterministic FSAs** (or **NFSAs**). Consider the sheeptalk automaton in Figure 2.18, which is much like our first automaton in Figure 2.10:

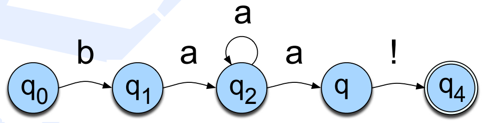


Figure 2.18 A non-deterministic finite-state automaton for talking sheep (NFSA #1). Compare with the deterministic automaton in Fig. 2.10.

NON-DETERMINISTIC
NFSA
DFSA

The only difference between this automaton and the previous one is that here in Figure 2.18 the self-loop is on state 2 instead of state 3. Consider using this network as an automaton for recognizing sheeptalk. When we get to state 2, if we see an **a** we don't know whether to remain in state 2 or go on to state 3. Automata with decision points like this are called **non-deterministic FSAs** (or **NFSAs**). Recall by contrast that Figure 2.10 specified a **deterministic** automaton, i.e., one whose behavior during recognition is fully *determined* by the state it is in and the symbol it is looking at. A deterministic automaton can be referred to as a **DFSA**. That is not true for the machine in Figure 2.18 (NFSA #1).

ε-TRANSITION

There is another common type of non-determinism, caused by arcs that have no symbols on them (called **ε-transitions**). The automaton in Fig. 2.19 defines the exact same language as the last one, or our first one, but it does it with an ε-transition.

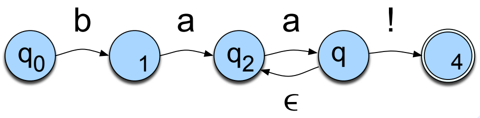


Figure 2.19 Another NFSA for the sheep language (NFSA #2). It differs from NFSA #1 in Fig. 2.18 in having an ε-transition.

We interpret this new arc as follows: If we are in state 3, we are allowed to move to state 2 *without* looking at the input, or advancing our input pointer. So this introduces another kind of non-determinism — we might not know whether to follow the ε-transition or the **!** arc.

2.2.5 Using an NFSA to Accept Strings

If we want to know whether a string is an instance of sheeptalk or not, and if we use a non-deterministic machine to recognize it, we might follow the wrong arc and reject it when we should have accepted it. That is, since there is more than one choice at some point, we might take the wrong choice. This problem of choice in non-deterministic models will come up again and again as we build computational models, particularly for parsing. There are three standard **solutions to the problem of non-determinism**:

BACKUP

- **Backup:** Whenever we come to a choice point, we could put a *marker* to mark where we were in the input, and what state the automaton was in. Then if it turns out that we took the wrong choice, we could back up and try another path.

LOOK-AHEAD

- **Look-ahead:** We could look ahead in the input to help us decide which path to take.

PARALLELISM

- **Parallelism:** Whenever we come to a choice point, we could look at every alternative path in parallel.

We will focus here on the backup approach and defer discussion of the look-ahead and parallelism approaches to later chapters.

The backup approach suggests that we should blithely make choices that might lead to deadends, knowing that we can always return to unexplored alternative choices. There are two keys to this approach: we need to remember all the alternatives for each choice point, and we need to store sufficient information about each alternative so that we can return to it when necessary. When a backup algorithm reaches a point in its processing where no progress can be made (because it runs out of input, or has no legal transitions), it returns to a previous choice point, selects one of the unexplored alternatives, and continues from there. Applying this notion to our non-deterministic recognizer, we need only remember two things for each choice point: the state, or node, of the machine that we can go to and the corresponding position on the tape. We will

SEARCH-STATE

call the combination of the node and position the **search-state** of the recognition algo-

	Input			
State	b	a	!	ε
0	1	∅	∅	∅
1	∅	2	∅	∅
2	∅	2,3	∅	∅
3	∅	∅	4	∅
4:	∅	∅	∅	∅

Figure 2.20 The transition table from NFSA #1 in Fig. 2.18.

rithm. To avoid confusion, we will refer to the state of the automaton (as opposed to the state of the search) as a **node** or a **machine-state**. Figure 2.21 presents a recognition algorithm based on this approach.

Before going on to describe the main part of this algorithm, we should note two changes to the transition table that drives it. First, in order to represent nodes that have outgoing ε-transitions, we add a new **ε-column** to the transition table. If a node has an ε-transition, we list the destination node in the ε-column for that node's row. The second addition is needed to account for multiple transitions to different nodes from the same input symbol. We let each cell entry consist of a list of destination nodes rather than a single node. Fig. 2.20 shows the transition table for the machine in Figure 2.18 (NFSA #1). While it has no ε-transitions, it does show that in machine-state q_2 the input *a* can lead back to q_2 or on to q_3.

Fig. 2.21 shows the algorithm for using a non-deterministic FSA to recognize an input string. The function ND-RECOGNIZE uses the variable *agenda* to keep track of all the currently unexplored choices generated during the course of processing. Each choice (search state) is a tuple consisting of a node (state) of the machine and a position on the tape. The variable *current-search-state* represents the branch choice being currently explored.

ND-RECOGNIZE begins by creating an initial search-state and placing it on the agenda. For now we don't specify what order the search-states are placed on the agenda. This search-state consists of the initial machine-state of the machine and a pointer to the beginning of the tape. The function NEXT is then called to retrieve an item from the agenda and assign it to the variable *current-search-state*.

As with D-RECOGNIZE, the first task of the main loop is to determine if the entire contents of the tape have been successfully recognized. This is done via a call to ACCEPT-STATE?, which returns *accept* if the current search-state contains both an accepting machine-state and a pointer to the end of the tape. If we're not done, the machine generates a set of possible next steps by calling GENERATE-NEW-STATES, which creates search-states for any ε-transitions and any normal input-symbol transitions from the transition table. All of these search-state tuples are then added to the current agenda.

Finally, we attempt to get a new search-state to process from the agenda. If the agenda is empty we've run out of options and have to reject the input. Otherwise, an unexplored option is selected and the loop continues.

It is important to understand why ND-RECOGNIZE returns a value of reject only when the agenda is found to be empty. Unlike D-RECOGNIZE, it does not return reject when it reaches the end of the tape in a non-accept machine-state or when it finds itself unable to advance the tape from some machine-state. This is because, in the non-deterministic case, such roadblocks only indicate failure down a given path, not overall failure. We can only be sure we can reject a string when all possible choices have been examined and found lacking.

```
function ND-RECOGNIZE(tape, machine) returns accept or reject

 agenda ← {(Initial state of machine, beginning of tape)}
 current-search-state ← NEXT(agenda)
 loop
   if ACCEPT-STATE?(current-search-state) returns true then
      return accept
   else
      agenda ← agenda ∪ GENERATE-NEW-STATES(current-search-state)
   if agenda is empty then
      return reject
   else
      current-search-state ← NEXT(agenda)
 end

function GENERATE-NEW-STATES(current-state) returns a set of search-states

 current-node ← the node the current search-state is in
 index ← the point on the tape the current search-state is looking at
 return a list of search states from transition table as follows:
   (transition-table[current-node,ε], index)
   ∪
   (transition-table[current-node, tape[index]], index + 1)

function ACCEPT-STATE?(search-state) returns true or false

 current-node ← the node search-state is in
 index ← the point on the tape search-state is looking at
 if index is at the end of the tape and current-node is an accept state of machine
 then
   return true
 else
   return false
```

Figure 2.21 An algorithm for NFSA recognition. The word *node* means a state of the FSA, while *state* or *search-state* means "the state of the search process", i.e., a combination of *node* and *tape-position*.

Figure 2.22 illustrates the progress of ND-RECOGNIZE as it attempts to handle the input `baaa!`. Each strip illustrates the state of the algorithm at a given point in its processing. The *current-search-state* variable is captured by the solid bubbles representing the machine-state along with the arrow representing progress on the tape. Each strip lower down in the figure represents progress from one *current-search-state* to the

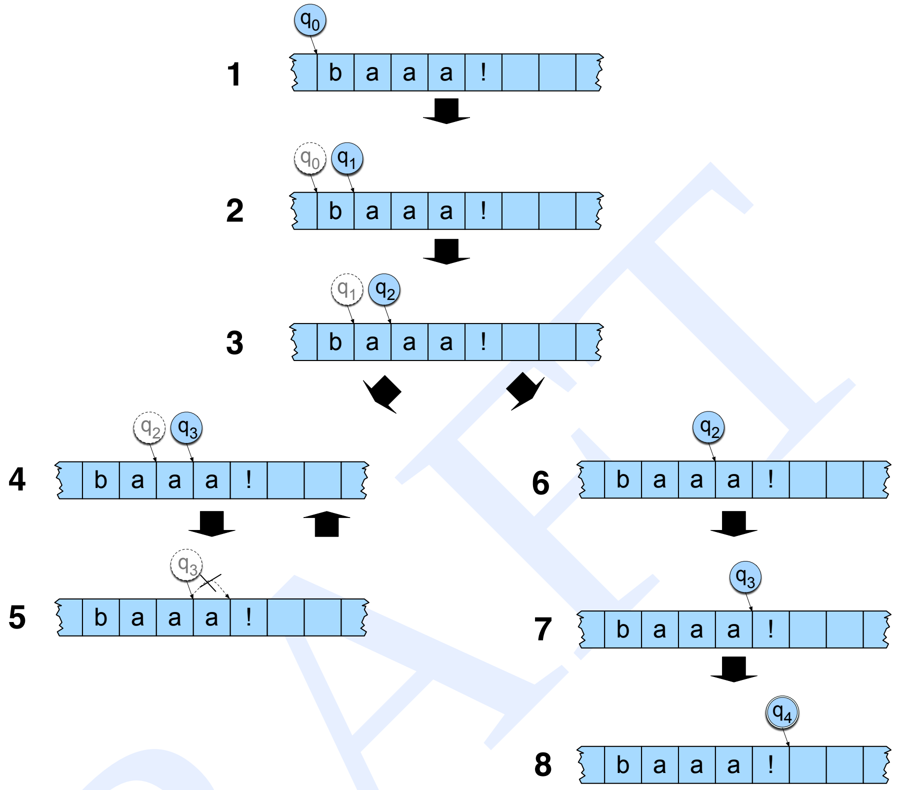


Figure 2.22 Tracing the execution of NFSA #1 (Fig. 2.18) on some sheeptalk.

next.

Little of interest happens until the algorithm finds itself in state q_2 while looking at the second a on the tape. An examination of the entry for transition-table[q_2,a] returns both q_2 and q_3. Search states are created for each of these choices and placed on the agenda. Unfortunately, our algorithm chooses to move to state q_3, a move that results in neither an accept state nor any new states since the entry for transition-table[q_3, a] is empty. At this point, the algorithm simply asks the agenda for a new state to pursue. Since the choice of returning to q_2 from q_2 is the only unexamined choice on the agenda it is returned with the tape pointer advanced to the next a. Somewhat diabolically, ND-RECOGNIZE finds itself faced with the same choice. The entry for transition-table[q_2,a] still indicates that looping back to q_2 or advancing to q_3 are valid choices. As before, states representing both are placed on the agenda. These search states are not the same as the previous ones since their tape index values have advanced. This time the agenda provides the move to q_3 as the next move. The move to q_4, and success, is then uniquely determined by the tape and the transition-table.

2.2.6 Recognition as Search

ND-RECOGNIZE accomplishes the task of recognizing strings in a regular language by providing a way to systematically explore all the possible paths through a machine. If this exploration yields a path ending in an accept state, it accepts the string, otherwise it rejects it. This systematic exploration is made possible by the agenda mechanism, which on each iteration selects a partial path to explore and keeps track of any remaining, as yet unexplored, partial paths.

STATE-SPACE SEARCH

Algorithms such as ND-RECOGNIZE, which operate by systematically searching for solutions, are known as **state-space search** algorithms. In such algorithms, the problem definition creates a space of possible solutions; the goal is to explore this space, returning an answer when one is found or rejecting the input when the space has been exhaustively explored. In ND-RECOGNIZE, search states consist of pairings of machine-states with positions on the input tape. The state-space consists of all the pairings of machine-state and tape positions that are possible given the machine in question. The goal of the search is to navigate through this space from one state to another looking for a pairing of an accept state with an end of tape position.

The key to the effectiveness of such programs is often the *order* in which the states in the space are considered. A poor ordering of states may lead to the examination of a large number of unfruitful states before a successful solution is discovered. Unfortunately, it is typically not possible to tell a good choice from a bad one, and often the best we can do is to insure that each possible solution is eventually considered.

DEPTH-FIRST

Careful readers may have noticed that the ordering of states in ND-RECOGNIZE has been left unspecified. We know only that unexplored states are added to the agenda as they are created and that the (undefined) function NEXT returns an unexplored state from the agenda when asked. How should the function NEXT be defined? Consider an ordering strategy where the states that are considered next are the most recently created ones. Such a policy can be implemented by placing newly created states at the front of the agenda and having NEXT return the state at the front of the agenda when called. Thus the agenda is implemented by a **stack**. This is commonly referred to as a **depth-first search** or **Last In First Out** (**LIFO**) strategy.

Such a strategy dives into the search space following newly developed leads as they are generated. It will only return to consider earlier options when progress along a current lead has been blocked. The trace of the execution of ND-RECOGNIZE on the string `baaa!` as shown in Fig. 2.22 illustrates a depth-first search. The algorithm hits the first choice point after seeing `ba` when it has to decide whether to stay in q_2 or advance to state q_3. At this point, it chooses one alternative and follows it until it is sure it's wrong. The algorithm then backs up and tries another older alternative.

Depth first strategies have one major pitfall: under certain circumstances they can enter an infinite loop. This is possible either if the search space happens to be set up in such a way that a search-state can be accidentally re-visited, or if there are an infinite number of search states. We will revisit this question when we turn to more complicated search problems in parsing in Ch. 13.

The second way to order the states in the search space is to consider states in the order in which they are created. Such a policy can be implemented by placing newly created states at the back of the agenda and still have NEXT return the state at the

BREADTH-FIRST

front of the agenda. Thus the agenda is implemented via a **queue**. This is commonly referred to as a **breadth-first search** or **First In First Out** (**FIFO**) strategy. Consider a different trace of the execution of ND-RECOGNIZE on the string `baaa!` as shown in Fig. 2.23. Again, the algorithm hits its first choice point after seeing `ba` when it had to decide whether to stay in q_2 or advance to state q_3. But now rather than picking one choice and following it up, we imagine examining all possible choices, expanding one ply of the search tree at a time.

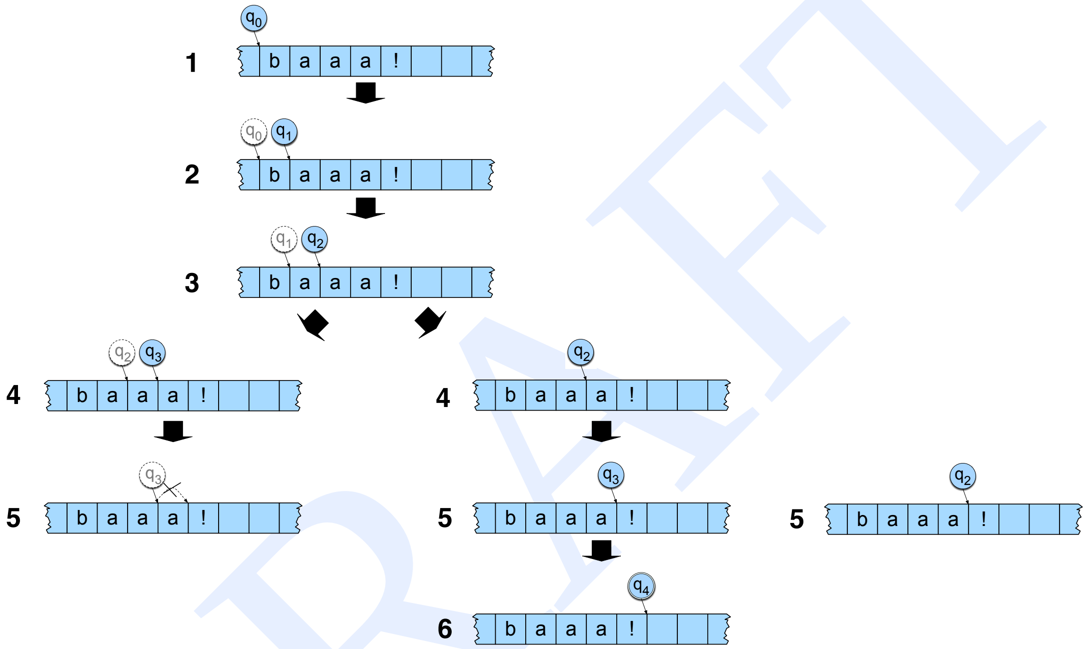


Figure 2.23 A breadth-first trace of FSA #1 on some sheeptalk.

Like depth-first search, breadth-first search has its pitfalls. As with depth-first if the state-space is infinite, the search may never terminate. More importantly, due to growth in the size of the agenda if the state-space is even moderately large, the search may require an impractically large amount of memory. For small problems, either depth-first or breadth-first search strategies may be adequate, although depth-first is normally preferred for its more efficient use of memory. For larger problems, more complex search techniques such as **dynamic programming** or **A*** must be used, as we will see in Chapters 7 and 10.

2.2.7 Relating Deterministic and Non-Deterministic Automata

It may seem that allowing NFSAs to have non-deterministic features like ε-transitions would make them more powerful than DFSAs. In fact this is not the case; for any NFSA, there is an exactly equivalent DFSA. In fact there is a simple algorithm for converting an NFSA to an equivalent DFSA, although the number of states in this equivalent deterministic automaton may be much larger. See Lewis and Papadimitriou (1988) or Hopcroft and Ullman (1979) for the proof of the correspondence. The basic intuition of the proof is worth mentioning, however, and builds on the way NFSAs parse their input. Recall that the difference between NFSAs and DFSAs is that in an NFSA a state q_i may have more than one possible next state given an input i (for example q_a and q_b). The algorithm in Figure 2.21 dealt with this problem by choosing either q_a or q_b and then *backtracking* if the choice turned out to be wrong. We mentioned that a parallel version of the algorithm would follow both paths (toward q_a and q_b) simultaneously.

The algorithm for converting a NFSA to a DFSA is like this parallel algorithm; we build an automaton that has a deterministic path for every path our parallel recognizer might have followed in the search space. We imagine following both paths simultaneously, and group together into an equivalence class all the states we reach on the same input symbol (i.e., q_a and q_b). We now give a new state label to this new equivalence class state (for example q_{ab}). We continue doing this for every possible input for every possible group of states. The resulting DFSA can have as many states as there are distinct sets of states in the original NFSA. The number of different subsets of a set with N elements is 2^N, hence the new DFSA can have as many as 2^N states.

2.3 REGULAR LANGUAGES AND FSAS

REGULAR LANGUAGES

As we suggested above, the class of languages that are definable by regular expressions is exactly the same as the class of languages that are characterizable by finite-state automata (whether deterministic or non-deterministic). Because of this, we call these languages the **regular languages**. In order to give a formal definition of the class of regular languages, we need to refer back to two earlier concepts: the alphabet Σ, which is the set of all symbols in the language, and the *empty string* ε, which is conventionally not included in Σ. In addition, we make reference to the *empty set* $\emptyset$ (which is distinct from ε). The class of regular languages (or **regular sets**) over Σ is then formally defined as follows: [1]

1. $\emptyset$ is a regular language
2. $\forall a \in \Sigma \cup \varepsilon$, $\{a\}$ is a regular language
3. If L_1 and L_2 are regular languages, then so are:
   (a) $L_1 \cdot L_2 = \{xy | x \in L_1, y \in L_2\}$, the **concatenation** of L_1 and L_2
   (b) $L_1 \cup L_2$, the **union** or **disjunction** of L_1 and L_2
   (c) L_1^*, the **Kleene closure** of L_1

Only languages which meet the above properties are regular languages. Since the regular languages are the languages characterizable by regular expressions, all the regular expression operators introduced in this chapter (except memory) can be implemented by the three operations which define regular languages: concatenation, disjunction/union (also called "|"), and Kleene closure. For example all the counters (*,+,

[1] Following van Santen and Sproat (1998), Kaplan and Kay (1994), and Lewis and Papadimitriou (1988).

`{n,m}`) are just a special case of repetition plus Kleene *. All the anchors can be thought of as individual special symbols. The square braces `[]` are a kind of disjunction (i.e., `[ab]` means "*a* or *b*", or the disjunction of *a* and *b*). Thus it is true that any regular expression can be turned into a (perhaps larger) expression which only makes use of the three primitive operations.

Regular languages are also closed under the following operations (Σ^* means the infinite set of all possible strings formed from the alphabet Σ):

- **intersection**: if L_1 and L_2 are regular languages, then so is $L_1 \cap L_2$, the language consisting of the set of strings that are in both L_1 and L_2.
- **difference**: if L_1 and L_2 are regular languages, then so is $L_1 - L_2$, the language consisting of the set of strings that are in L_1 but not L_2.
- **complementation**: If L_1 is a regular language, then so is $\Sigma^* - L_1$, the set of all possible strings that aren't in L_1.
- **reversal**: If L_1 is a regular language, then so is L_1^R, the language consisting of the set of reversals of all the strings in L_1.

The proof that regular expressions are equivalent to finite-state automata can be found in Hopcroft and Ullman (1979), and has two parts: showing that an automaton can be built for each regular language, and conversely that a regular language can be built for each automaton.

We won't give the proof, but we give the intuition by showing how to do the first part: take any regular expression and build an automaton from it. The intuition is inductive on the number of operators: for the base case we build an automaton to correspond to the regular expressions with no operators, i.e. the regular expressions $\emptyset$, ϵ, or any single symbol $a \in \Sigma$. Fig. 2.24 shows the automata for these three base cases.

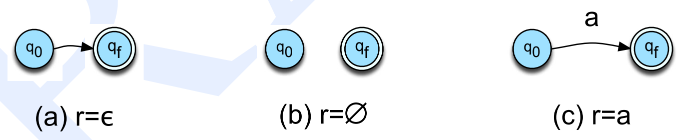


Figure 2.24 Automata for the base case (no operators) for the induction showing that any regular expression can be turned into an equivalent automaton.

Now for the inductive step, we show that each of the primitive operations of a regular expression (concatenation, union, closure) can be imitated by an automaton:

- **concatenation**: We just string two FSAs next to each other by connecting all the final states of FSA_1 to the initial state of FSA_2 by an ϵ-transition.
- **closure**: We create a new final and initial state, connect the original final states of the FSA back to the initial states by ϵ-transitions (this implements the repetition part of the Kleene *), and then put direct links between the new initial and final states by ϵ-transitions (this implements the possibility of having *zero* occurrences). We'd leave out this last part to implement Kleene-plus instead.
- **union**: We add a single new initial state q_0', and add new ϵ-transitions from it to the former initial states of the two machines to be joined.

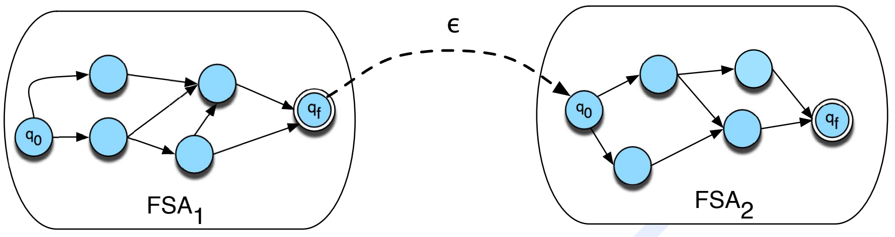


Figure 2.25 The concatenation of two FSAs.

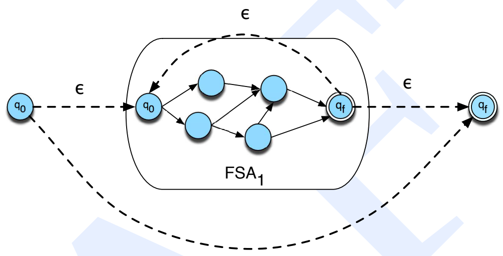


Figure 2.26 The closure (Kleene *) of an FSA.

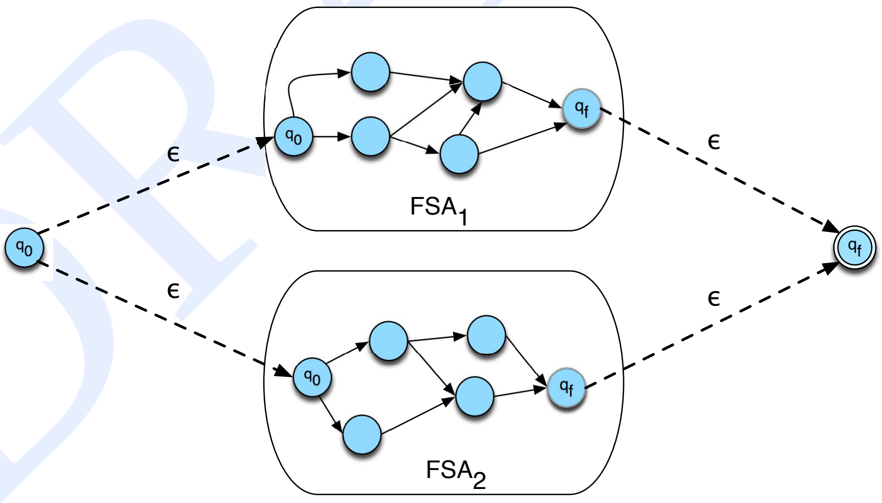


Figure 2.27 The union (|) of two FSAs.

We will return to regular languages and their relationship to regular grammars in Ch. 15.

2.4 Summary

This chapter introduced the most important fundamental concept in language processing, the **finite automaton**, and the practical tool based on automaton, the **regular expression**. Here's a summary of the main points we covered about these ideas:

- The **regular expression** language is a powerful tool for pattern-matching.
- Basic operations in regular expressions include **concatenation** of symbols, **disjunction** of symbols (`[]`, `|`, and `.`), **counters** (`*`, `+`, and `{n,m}`), **anchors** (`^`, `$`) and precedence operators (`(`,`)`).
- Any regular expression can be realized as a **finite state automaton** (**FSA**).
- Memory (`\1` together with `()`) is an advanced operation that is often considered part of regular expressions, but which cannot be realized as a finite automaton.
- An automaton implicitly defines a **formal language** as the set of strings the automaton **accepts**.
- An automaton can use any set of symbols for its vocabulary, including letters, words, or even graphic images.
- The behavior of a **deterministic** automaton (**DFSA**) is fully determined by the state it is in.
- A **non-deterministic** automaton (**NFSA**) sometimes has to make a choice between multiple paths to take given the same current state and next input.
- Any **NFSA** can be converted to a **DFSA**.
- The order in which a **NFSA** chooses the next state to explore on the agenda defines its **search strategy**. The **depth-first search** or **LIFO** strategy corresponds to the agenda-as-stack; the **breadth-first search** or **FIFO** strategy corresponds to the agenda-as-queue.
- Any regular expression can be automatically compiled into a **NFSA** and hence into a **FSA**.

Bibliographical and Historical Notes

Finite automata arose in the 1950s out of Turing's (1936) model of algorithmic computation, considered by many to be the foundation of modern computer science. The Turing machine was an abstract machine with a finite control and an input/output tape. In one move, the Turing machine could read a symbol on the tape, write a different symbol on the tape, change state, and move left or right. Thus the Turing machine differs from a finite-state automaton mainly in its ability to change the symbols on its tape.

MCCULLOCH-PITTS NEURON

Inspired by Turing's work, McCulloch and Pitts built an automata-like model of the neuron (see von Neumann, 1963, p. 319). Their model, which is now usually called the **McCulloch-Pitts neuron** (McCulloch and Pitts, 1943), was a simplified model of the neuron as a kind of "computing element" that could be described in terms of propositional logic. The model was a binary device, at any point either active or not, which took excitatory and inhibitatory input from other neurons and fired if its activation passed some fixed threshold. Based on the McCulloch-Pitts neuron, Kleene (1951) and (1956) defined the finite automaton and regular expressions, and proved their equivalence. Non-deterministic automata were introduced by Rabin and Scott (1959), who also proved them equivalent to deterministic ones.

Ken Thompson was one of the first to build regular expressions compilers into editors for text searching (Thompson, 1968). His editor *ed* included a command "g/regular expression/p", or Global Regular Expression Print, which later became the UNIX `grep` utility.

There are many general-purpose introductions to the mathematics underlying automata theory, such as Hopcroft and Ullman (1979) and Lewis and Papadimitriou (1988). These cover the mathematical foundations of the simple automata of this chapter, as well as the finite-state transducers of Ch. 3, the context-free grammars of Ch. 12, and the Chomsky hierarchy of Ch. 15. Friedl (1997) is a very useful comprehensive guide to the advanced use of regular expressions.

The metaphor of problem-solving as search is basic to Artificial Intelligence (AI); more details on search can be found in any AI textbook such as Russell and Norvig (2002).

EXERCISES

2.1 Write regular expressions for the following languages: You may use either Perl notation or the minimal "algebraic" notation of Sec. 2.3, but make sure to say which one you are using. By "word", we mean an alphabetic string separated from other words by white space, any relevant punctuation, line breaks, and so forth.

a. the set of all alphabetic strings.

b. the set of all lowercase alphabetic strings ending in a *b*.

c. the set of all strings with two consecutive repeated words (e.g., "Humbert Humbert" and "the the" but not "the bug" or "the big bug").

d. the set of all strings from the alphabet a, b such that each *a* is immediately preceded and immediately followed by a *b*.

e. all strings which start at the beginning of the line with an integer (i.e., 1,2,3,...,10,...,10000,...) and which end at the end of the line with a word.

f. all strings which have both the word *grotto* and the word *raven* in them. (but not, for example, words like *grottos* that merely *contain* the word *grotto*).

g. write a pattern which places the first word of an English sentence in a register. Deal with punctuation.

2.2 Implement an ELIZA-like program, using substitutions such as those described on page 10. You may choose a different domain than a Rogerian psychologist, if you wish, although keep in mind that you would need a domain in which your program can legitimately do a lot of simple repeating-back.

2.3 Complete the FSA for English money expressions in Fig. 2.16 as suggested in the text following the figure. You should handle amounts up to $100,000, and make sure that “cent” and “dollar” have the proper plural endings when appropriate.

2.4 Design an FSA that recognizes simple date expressions like *March 15*, *the 22nd of November*, *Christmas*. You should try to include all such “absolute” dates, (e.g. not “deictic” ones relative to the current day like *the day before yesterday*). Each edge of the graph should have a word or a set of words on it. You should use some sort of shorthand for classes of words to avoid drawing too many arcs (e.g., furniture → desk, chair, table).

2.5 Now extend your date FSA to handle deictic expressions like *yesterday*, *tomorrow*, *a week from tomorrow*, *the day before yesterday*, *Sunday*, *next Monday*, *three weeks from Saturday*.

2.6 Write an FSA for time-of-day expressions like *eleven o’clock*, *twelve-thirty*, *midnight*, or *a quarter to ten* and others.

2.7 (Due to Pauline Welby; this problem probably requires the ability to knit.) Write a regular expression (or draw an FSA) which matches all knitting patterns for scarves with the following specification: *32 stitches wide, K1P1 ribbing on both ends, stockinette stitch body, exactly two raised stripes*. All knitting patterns must include a cast-on row (to put the correct number of stitches on the needle) and a bind-off row (to end the pattern and prevent unraveling). Here’s a sample pattern for one possible scarf matching the above description:[2]

1.	Cast on 32 stitches.	*cast on; puts stitches on needle*
2.	K1 P1 across row (i.e. do (K1 P1) 16 times).	*K1P1 ribbing*
3.	Repeat instruction 2 seven more times.	*adds length*
4.	K32, P32.	*stockinette stitch*
5.	Repeat instruction 4 an additional 13 times.	*adds length*
6.	P32, P32.	*raised stripe stitch*
7.	K32, P32.	*stockinette stitch*
8.	Repeat instruction 7 an additional 251 times.	*adds length*
9.	P32, P32.	*raised stripe stitch*
10.	K32, P32.	*stockinette stitch*
11.	Repeat instruction 10 an additional 13 times.	*adds length*
12.	K1 P1 across row.	*K1P1 ribbing*
13.	Repeat instruction 12 an additional 7 times.	*adds length*
14.	Bind off 32 stitches.	*binds off row: ends pattern*

[2] *Knit* and *purl* are two different types of stitches. The notation K*n* means do *n* knit stitches. Similarly for purl stitches. Ribbing has a striped texture—most sweaters have ribbing at the sleeves, bottom, and neck. Stockinette stitch is a series of knit and purl rows that produces a plain pattern— socks or stockings are knit with this basic pattern, hence the name.

2.8 Write a regular expression for the language accepted by the NFSA in Fig. 2.28.

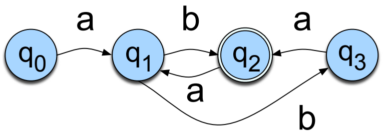


Figure 2.28 A mystery language

2.9 Currently the function D-RECOGNIZE in Fig. 2.13 only solves a subpart of the important problem of finding a string in some text. Extend the algorithm to solve the following two deficiencies: (1) D-RECOGNIZE currently assumes that it is already pointing at the string to be checked, and (2) D-RECOGNIZE fails if the string it is pointing includes as a proper substring a legal string for the FSA. That is, D-RECOGNIZE fails if there is an extra character at the end of the string.

2.10 Give an algorithm for negating a deterministic FSA. The negation of an FSA accepts exactly the set of strings that the original FSA rejects (over the same alphabet), and rejects all the strings that the original FSA accepts.

2.11 Why doesn't your previous algorithm work with NFSAs? Now extend your algorithm to negate an NFSA.

Friedl, J. E. F. (1997). *Master Regular Expressions*. O'Reilly.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

Kaplan, R. M. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, *20*(3), 331–378.

Kleene, S. C. (1951). Representation of events in nerve nets and finite automata. Tech. rep. RM-704, RAND Corporation. RAND Research Memorandum†.

Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In Shannon, C. and McCarthy, J. (Eds.), *Automata Studies*, pp. 3–41. Princeton University Press, Princeton, NJ.

Lewis, H. and Papadimitriou, C. (1988). *Elements of the Theory of Computation*. Prentice-Hall. Second edition.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115–133. Reprinted in *Neurocomputing: Foundations of Research, ed. by J. A. Anderson and E Rosenfeld. MIT Press 1988*.

Rabin, M. O. and Scott, D. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, *3*(2), 114–125.

Russell, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall. Second edition.

Thompson, K. (1968). Regular expression search algorithm. *Communications of the ACM*, *11*(6), 419–422.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *42*, 230–265. Read to the Society in 1936, but published in 1937. Correction in volume 43, 544–546.

van Santen, J. P. H. and Sproat, R. (1998). Methods and tools. In Sproat, R. (Ed.), *Multilingual Text-To-Speech Synthesis: The Bell Labs Approach*, pp. 7–30. Kluwer, Dordrecht.

von Neumann, J. (1963). *Collected Works: Volume V*. Macmillan Company, New York.

Weizenbaum, J. (1966). ELIZA – A computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36–45.

3 WORDS & TRANSDUCERS

How can there be any sin in sincere?
Where is the good in goodbye?
Meredith Willson, *The Music Man*

Ch. 2 introduced the regular expression, showing for example how a single search string could help us find both *woodchuck* and *woodchucks*. Hunting for singular or plural woodchucks was easy; the plural just tacks an *s* on to the end. But suppose we were looking for another fascinating woodland creatures; let's say a *fox*, and a *fish*, that surly *peccary* and perhaps a Canadian *wild goose*. Hunting for the plurals of these animals takes more than just tacking on an *s*. The plural of *fox* is *foxes*; of *peccary*, *peccaries*; and of *goose*, *geese*. To confuse matters further, fish don't usually change their form when they are plural[1].

It takes two kinds of knowledge to correctly search for singulars and plurals of these forms. **Orthographic rules** tell us that English words ending in *-y* are pluralized by changing the *-y* to *-i-* and adding an *-es*. **Morphological rules** tell us that *fish* has a null plural, and that the plural of *goose* is formed by changing the vowel.

The problem of recognizing that a word (like *foxes*) breaks down into component morphemes (*fox* and *-es*) and building a structured representation of this fact is called **morphological parsing**.

MORPHOLOGICAL PARSING

PARSING

Parsing means taking an input and producing some sort of linguistic structure for it. We will use the term parsing very broadly throughout this book, including many kinds of structures that might be produced; morphological, syntactic, semantic, discourse; in the form of a string, or a tree, or a network. Morphological parsing or stemming applies to many affixes other than plurals; for example we might need to take any English verb form ending in *-ing* (*going*, *talking*, *congratulating*) and parse it into its verbal stem plus the *-ing* morpheme. So given the **surface** or **input form** *going*, we might want to produce the parsed form `VERB-go + GERUND-ing`.

SURFACE

Morphological parsing is important throughout speech and language processing. It plays a crucial role in Web search for morphologically complex languages like Russian or German; in Russian the word *Moscow* has different endings in the phrases *Moscow*, *of Moscow*, *from Moscow*, and so on. We want to be able to automatically

[1] (see e.g., Seuss (1960))

search for the inflected forms of the word even if the user only typed in the base form. Morphological parsing also plays a crucial role in part-of-speech tagging for these morphologically complex languages, as we will see in Ch. 5. It is important for producing the large dictionaries that are necessary for robust spell-checking. We will need it in machine translation to realize for example that the French words *va* and *aller* should both translate to forms of the English verb *go*.

PRODUCTIVE

To solve the morphological parsing problem, why couldn't we just store all the plural forms of English nouns and *-ing* forms of English verbs in a dictionary and do parsing by lookup? Sometimes we can do this, and for example for English speech recognition this is exactly what we do. But for many NLP applications this isn't possible because *-ing* is a **productive** suffix; by this we mean that it applies to every verb. Similarly *-s* applies to almost every noun. Productive suffixes even apply to new words; thus the new word *fax* can automatically be used in the *-ing* form: *faxing*. Since new words (particularly acronyms and proper nouns) are created every day, the class of nouns in English increases constantly, and we need to be able to add the plural morpheme *-s* to each of these. Additionally, the plural form of these new nouns depends on the spelling/pronunciation of the singular form; for example if the noun ends in *-z* then the plural form is *-es* rather than *-s*. We'll need to encode these rules somewhere.

Finally, we certainly cannot list all the morphological variants of every word in morphologically complex languages like Turkish, which has words like:

(3.1) uygarlaştıramadıklarımızdanmışsınızcasına

uygar	*+laş*	*+tır*	*+ama*	*+dık*	*+lar*	*+ımız*	*+dan*	*+mış*	*+sınız*	*+casına*
civilized	+BEC	+CAUS	+NABL	+PART	+PL	+P1PL	+ABL	+PAST	+2PL	+AsIf

"(behaving) as if you are among those whom we could not civilize"

The various pieces of this word (the **morphemes**) have these meanings:

+BEC	"become"
+CAUS	the causative verb marker ('cause to X')
+NABL	"not able"
+PART	past participle form
+P1PL	1st person pl possessive agreement
+2PL	2nd person pl
+ABL	ablative (from/among) case marker
+AsIf	derivationally forms an adverb from a finite verb

Not all Turkish words look like this; the average Turkish word has about three morphemes. But such long words do exist; indeed Kemal Oflazer, who came up with this example, notes (p.c.) that verbs in Turkish have 40,000 possible forms not counting derivational suffixes. Adding derivational suffixes, such as causatives, allows a theoretically infinite number of words, since causativization can be repeated in a single word (*You cause X to cause Y to . . . do W*). Thus we cannot store all possible Turkish words in advance, and must do morphological parsing dynamically.

In the next section we survey morphological knowledge for English and some other languages. We then introduce the key algorithm for morphological parsing, the **finite-state transducer**. Finite-state transducers are a crucial technology throughout speech and language processing, so we will return to them again in later chapters.

STEMMING

After describing morphological parsing, we will introduce some related algorithms in this chapter. In some applications we don't need to parse a word, but we do need to map from the word to its root or stem. For example in information retrieval and web search (IR), we might want to map from *foxes* to *fox*; but might not need to also know that *foxes* is plural. Just stripping off such word endings is called **stemming** in IR. We will describe a simple stemming algorithm called the **Porter stemmer**.

LEMMATIZATION

For other speech and language processing tasks, we need to know that two words have a similar root, despite their surface differences. For example the words *sang*, *sung*, and *sings* are all forms of the verb *sing*. The word *sing* is sometimes called the common *lemma* of these words, and mapping from all of these to *sing* is called **lemmatization**.[2]

TOKENIZATION

Next, we will introduce another task related to morphological parsing. **Tokenization** or **word segmentation** is the task of separating out (tokenizing) words from running text. In English, words are often separated from each other by blanks (whitespace), but whitespace is not always sufficient; we'll need to notice that *New York* and *rock 'n' roll* are individual words despite the fact that they contain spaces, but for many applications we'll need to separate *I'm* into the two words *I* and *am*.

Finally, for many applications we need to know how similar two words are orthographically. Morphological parsing is one method for computing this similarity, but another is to just compare the strings of letters to see how similar they are. A common way of doing this is with the **minimum edit distance** algorithm, which is important throughout NLP. We'll introduce this algorithm and also show how it can be used in spell-checking.

3.1 Survey of (Mostly) English Morphology

MORPHEMES

Morphology is the study of the way words are built up from smaller meaning-bearing units, **morphemes**. A morpheme is often defined as the minimal meaning-bearing unit in a language. So for example the word *fox* consists of a single morpheme (the morpheme *fox*) while the word *cats* consists of two: the morpheme *cat* and the morpheme *-s*.

STEMS
AFFIXES

As this example suggests, it is often useful to distinguish two broad classes of morphemes: **stems** and **affixes**. The exact details of the distinction vary from language to language, but intuitively, the stem is the "main" morpheme of the word, supplying the main meaning, while the affixes add "additional" meanings of various kinds.

Affixes are further divided into **prefixes**, **suffixes**, **infixes**, and **circumfixes**. Prefixes precede the stem, suffixes follow the stem, circumfixes do both, and infixes are inserted inside the stem. For example, the word *eats* is composed of a stem *eat* and the suffix *-s*. The word *unbuckle* is composed of a stem *buckle* and the prefix *un-*. English doesn't have any good examples of circumfixes, but many other languages do. In German, for example, the past participle of some verbs is formed by adding *ge-* to the beginning of the stem and *-t* to the end; so the past participle of the verb *sagen* (to say) is *gesagt* (said). Infixes, in which a morpheme is inserted in the middle of a word,

[2] Lemmatization is actually more complex, since it sometimes involves deciding on which sense of a word is present. We return to this issue in Ch. 20.

occur very commonly for example in the Philipine language Tagalog. For example the affix *um*, which marks the agent of an action, is infixed to the Tagalog stem *hingi* "borrow" to produce *humingi*. There is one infix that occurs in some dialects of English in which the taboo morphemes "f**king" or "bl**dy" or others like them are inserted in the middle of other words ("Man-f**king-hattan", "abso-bl**dy-lutely"[3]) (McCawley, 1978).

A word can have more than one affix. For example, the word *rewrites* has the prefix *re-*, the stem *write*, and the suffix *-s*. The word *unbelievably* has a stem (*believe*) plus three affixes (*un-*, *-able*, and *-ly*). While English doesn't tend to stack more than four or five affixes, languages like Turkish can have words with nine or ten affixes, as we saw above. Languages that tend to string affixes together like Turkish does are called **agglutinative** languages.

INFLECTION
DERIVATION
COMPOUNDING
CLITICIZATION

There are many ways to combine morphemes to create words. Four of these methods are common and play important roles in speech and language processing: **inflection**, **derivation**, **compounding**, and **cliticization**.

CLITIC

Inflection is the combination of a word stem with a grammatical morpheme, usually resulting in a word of the same class as the original stem, and usually filling some syntactic function like agreement. For example, English has the inflectional morpheme *-s* for marking the **plural** on nouns, and the inflectional morpheme *-ed* for marking the past tense on verbs. **Derivation** is the combination of a word stem with a grammatical morpheme, usually resulting in a word of a *different* class, often with a meaning hard to predict exactly. For example the verb *computerize* can take the derivational suffix *-ation* to produce the noun *computerization*. **Compounding** is the combination of multiple word stems together. For example the noun *doghouse* is the concatenation of the morpheme *dog* with the morpheme *house*. Finally, **cliticization** is the combination of a word stem with a **clitic**. A clitic is a morpheme that acts syntactically like a word, but is reduced in form and attached (phonologically and sometimes orthographically) to another word. For example the English morpheme *'ve* in the word *I've* is a clitic, as is the French definite article *l'* in the word *l'opera*. In the following sections we give more details on these processes.

3.1.1 Inflectional Morphology

English has a relatively simple inflectional system; only nouns, verbs, and sometimes adjectives can be inflected, and the number of possible inflectional affixes is quite small.

PLURAL
SINGULAR

English nouns have only two kinds of inflection: an affix that marks **plural** and an affix that marks **possessive**. For example, many (but not all) English nouns can either appear in the bare stem or **singular** form, or take a plural suffix. Here are examples of the regular plural suffix *-s* (also spelled *-es*), and irregular plurals:

	Regular Nouns		Irregular Nouns	
Singular	cat	thrush	mouse	ox
Plural	cats	thrushes	mice	oxen

[3] Alan Jay Lerner, the lyricist of My Fair Lady, bowdlerized the latter to *abso-bloomin'lutely* in the lyric to "Wouldn't It Be Loverly?" (Lerner, 1978, p. 60).

While the regular plural is spelled -*s* after most nouns, it is spelled -*es* after words ending in -*s* (*ibis/ibises*), -*z* (*waltz/waltzes*), -*sh* (*thrush/thrushes*), -*ch* (*finch/finches*), and sometimes -*x* (*box/boxes*). Nouns ending in -*y* preceded by a consonant change the -*y* to -*i* (*butterfly/butterflies*).

The possessive suffix is realized by apostrophe + -*s* for regular singular nouns (*llama's*) and plural nouns not ending in -*s* (*children's*) and often by a lone apostrophe after regular plural nouns (*llamas'*) and some names ending in -*s* or -*z* (*Euripides' comedies*).

REGULAR

English verbal inflection is more complicated than nominal inflection. First, English has three kinds of verbs; **main verbs**, (*eat, sleep, impeach*), **modal verbs** (*can, will, should*), and **primary verbs** (*be, have, do*) (using the terms of Quirk et al., 1985). In this chapter we will mostly be concerned with the main and primary verbs, because it is these that have inflectional endings. Of these verbs a large class are **regular**, that is to say all verbs of this class have the same endings marking the same functions. These regular verbs (e.g. *walk*, or *inspect*) have four morphological forms, as follow:

Morphological Form Classes	Regularly Inflected Verbs			
stem	walk	merge	try	map
-*s* form	walks	merges	tries	maps
-*ing* participle	walking	merging	trying	mapping
Past form or -*ed* participle	walked	merged	tried	mapped

These verbs are called regular because just by knowing the stem we can predict the other forms by adding one of three predictable endings and making some regular spelling changes (and as we will see in Ch. 7, regular pronunciation changes). These regular verbs and forms are significant in the morphology of English first because they cover a majority of the verbs, and second because the regular class is **productive**. As discussed earlier, a productive class is one that automatically includes any new words that enter the language. For example the recently-created verb *fax* (*My mom faxed me the note from cousin Everett*) takes the regular endings -*ed*, -*ing*, -*es*. (Note that the -*s* form is spelled *faxes* rather than *faxs*; we will discuss spelling rules below).

IRREGULAR VERBS

PRETERITE

The **irregular verbs** are those that have some more or less idiosyncratic forms of inflection. Irregular verbs in English often have five different forms, but can have as many as eight (e.g., the verb *be*) or as few as three (e.g. *cut* or *hit*). While constituting a much smaller class of verbs (Quirk et al. (1985) estimate there are only about 250 irregular verbs, not counting auxiliaries), this class includes most of the very frequent verbs of the language.[4] The table below shows some sample irregular forms. Note that an irregular verb can inflect in the past form (also called the **preterite**) by changing its vowel (*eat/ate*), or its vowel and some consonants (*catch/caught*), or with no change at all (*cut/cut*).

[4] In general, the more frequent a word form, the more likely it is to have idiosyncratic properties; this is due to a fact about language change; very frequent words tend to preserve their form even if other words around them are changing so as to become more regular.

Morphological Form Classes	Irregularly Inflected Verbs		
stem	eat	catch	cut
-*s* form	eats	catches	cuts
-*ing* participle	eating	catching	cutting
Past form	ate	caught	cut
-*ed*/-*en* participle	eaten	caught	cut

PROGRESSIVE

GERUND

PERFECT

The way these forms are used in a sentence will be discussed in the syntax and semantics chapters but is worth a brief mention here. The -*s* form is used in the "habitual present" form to distinguish the third-person singular ending (*She jogs every Tuesday*) from the other choices of person and number (*I/you/we/they jog every Tuesday*). The stem form is used in the infinitive form, and also after certain other verbs (*I'd rather walk home*, *I want to walk home*). The -*ing* participle is used in the **progressive** construction to mark present or ongoing activity (*It is raining*), or when the verb is treated as a noun; this particular kind of nominal use of a verb is called a **gerund** use: *Fishing is fine if you live near water.* The -*ed*/-*en* participle is used in the **perfect** construction (*He's eaten lunch already*) or the passive construction (*The verdict was overturned yesterday*).

In addition to noting which suffixes can be attached to which stems, we need to capture the fact that a number of regular spelling changes occur at these morpheme boundaries. For example, a single consonant letter is doubled before adding the -*ing* and -*ed* suffixes (*beg/begging/begged*). If the final letter is "c", the doubling is spelled "ck" (*picnic/picnicking/picnicked*). If the base ends in a silent -*e*, it is deleted before adding -*ing* and -*ed* (*merge/merging/merged*). Just as for nouns, the -*s* ending is spelled -*es* after verb stems ending in -*s* (*toss/tosses*) , -*z*, (*waltz/waltzes*) -*sh*, (*wash/washes*) -*ch*, (*catch/catches*) and sometimes -*x* (*tax/taxes*). Also like nouns, verbs ending in -*y* preceded by a consonant change the -*y* to -*i* (*try/tries*).

The English verbal system is much simpler than for example the European Spanish system, which has as many as fifty distinct verb forms for each regular verb. Fig. 3.1 shows just a few of the examples for the verb *amar*, 'to love'. Other languages can have even more forms than this Spanish example.

	Present Indicative	Imperfect Indicative	Future	Preterite	Present Subjnct.	Conditional	Imperfect Subjnct.	Future Subjnct.
1SG	amo	amaba	amaré	amé	ame	amaría	amara	amare
2SG	amas	amabas	amarás	amaste	ames	amarías	amaras	amares
3SG	ama	amaba	amará	amó	ame	amaría	amara	amáreme
1PL	amamos	amábamos	amaremos	amamos	amemos	amaríamos	amáramos	amáremos
2PL	amáis	amabais	amaréis	amasteis	améis	amaríais	amarais	amareis
3PL	aman	amaban	amarán	amaron	amen	amarían	amaran	amaren

Figure 3.1 To love in Spanish. Some of the inflected forms of the verb *amar* in European Spanish. *1SG* stands for "first person singular", 3PL for "third person plural", and so on.

3.1.2 Derivational Morphology

While English inflection is relatively simple compared to other languages, derivation in English is quite complex. Recall that derivation is the combination of a word stem with a grammatical morpheme, usually resulting in a word of a *different* class, often with a meaning hard to predict exactly.

NOMINALIZATION

A very common kind of derivation in English is the formation of new nouns, often from verbs or adjectives. This process is called **nominalization**. For example, the suffix -*ation* produces nouns from verbs ending often in the suffix -*ize* (*computerize* → *computerization*). Here are examples of some particularly productive English nominalizing suffixes.

Suffix	Base Verb/Adjective	Derived Noun
-ation	computerize (V)	computerization
-ee	appoint (V)	appointee
-er	kill (V)	killer
-ness	fuzzy (A)	fuzziness

Adjectives can also be derived from nouns and verbs. Here are examples of a few suffixes deriving adjectives from nouns or verbs.

Suffix	Base Noun/Verb	Derived Adjective
-al	computation (N)	computational
-able	embrace (V)	embraceable
-less	clue (N)	clueless

Derivation in English is more complex than inflection for a number of reasons. One is that it is generally less productive; even a nominalizing suffix like -*ation*, which can be added to almost any verb ending in -*ize*, cannot be added to absolutely every verb. Thus we can't say **eatation* or **spellation* (we use an asterisk (*) to mark "non-examples" of English). Another is that there are subtle and complex meaning differences among nominalizing suffixes. For example *sincerity* has a subtle difference in meaning from *sincereness*.

3.1.3 Cliticization

Recall that a clitic is a unit whose status lies in between that of an affix and a word. The phonological behavior of clitics is like affixes; they tend to be short and unaccented (we will talk more about phonology in Ch. 8). Their syntactic behavior is more like words, often acting as pronouns, articles, conjunctions, or verbs. Clitics preceding a word are called **proclitics**, while those following are **enclitics**.

PROCLITICS
ENCLITICS

English clitics include these auxiliary verbal forms:

Full Form	Clitic	Full Form	Clitic
am	'm	have	've
are	're	has	's
is	's	had	'd
will	'll	would	'd

Note that the clitics in English are ambiguous; Thus *she's* can mean *she is* or *she has*. Except for a few such ambiguities, however, correctly segmenting off clitics in English is simplified by the presence of the apostrophe. Clitics can be harder to parse in other languages. In Arabic and Hebrew, for example, the definite article (*the*; *Al* in Arabic, *ha* in Hebrew) is cliticized on to the front of nouns. It must be segmented off in order to do part-of-speech tagging, parsing, or other tasks. Other Arabic proclitics include prepositions like *b* 'by/with', and conjunctions like *w* 'and'. Arabic also has *enclitics* marking certain pronouns. For example the word *and by their virtues* has clitics meaning *and*, *by*, and *their*, a stem *virtue*, and a plural affix. Note that since Arabic is read right to left, these would actually appear ordered from right to left in an Arabic word.

	proclitic	**proclitic**	**stem**	**affix**	**enclitic**
Arabic	w	b	Hsn	At	hm
Gloss	and	by	virtue	s	their

3.1.4 Non-concatenative Morphology

CONCATENATIVE

The kind of morphology we have discussed so far, in which a word is composed of a string of morphemes concatenated together is often called **concatenative morphology**. A number of languages have extensive **non-concatenative morphology**, in which morphemes are combined in more complex ways. The Tagalog infixation example above is one example of non-concatenative morphology, since two morphemes (*hingi* and *um*) are intermingled.

Another kind of non-concatenative morphology is called **templatic morphology** or **root-and-pattern** morphology. This is very common in Arabic, Hebrew, and other Semitic languages. In Hebrew, for example, a verb (as well as other parts-of-speech) is constructed using two components: a root, consisting usually of three consonants (CCC) and carrying the basic meaning, and a template, which gives the ordering of consonants and vowels and specifies more semantic information about the resulting verb, such as the semantic voice (e.g., active, passive, middle). For example the Hebrew tri-consonantal root *lmd*, meaning 'learn' or 'study', can be combined with the active voice CaCaC template to produce the word *lamad*, 'he studied', or the intensive CiCeC template to produce the word *limed*, 'he taught', or the intensive passive template CuCaC to produce the word *lumad*, 'he was taught'. Arabic and Hebrew combine this templatic morphology with concatenative morphology (like the cliticization example shown in the previous section).

3.1.5 Agreement

AGREE

GENDER

We introduced the plural morpheme above, and noted that plural is marked on both nouns and verbs in English. We say that the subject noun and the main verb in English have to **agree** in number, meaning that the two must either be both singular or both plural. There are other kinds of agreement processes. For example nouns, adjectives, and sometimes verbs in many languages are marked for **gender**. A gender is a kind of equivalence class that is used by the language to categorize the nouns; each noun falls into one class. Many languages (for example Romance languages like French, Spanish, or Italian) have 2 genders, which are referred to as masculine and feminine. Other languages (like most Germanic and Slavic languages) have three (masculine, feminine, neuter). Some languages, for example the Bantu languages of Africa, have as many as 20 genders. When the number of classes is very large, we often refer to them as **noun classes** instead of genders.

NOUN CLASSES

Gender is sometimes marked explicitly on a noun; for example Spanish masculine words often end in *-o* and feminine words in *-a*. But in many cases the gender is not marked in the letters or phones of the noun itself. Instead, it is a property of the word that must be stored in a lexicon. We will see an example of this in Fig. 3.2.

3.2 Finite-State Morphological Parsing

Let's now proceed to the problem of parsing morphology. Our goal will be to take input forms like those in the first and third columns of Fig. 3.2, produce output forms like those in the second and fourth column.

English		Spanish		
Input	Morphologically Parsed Output	Input	Morphologically Parsed Output	Gloss
cats	cat +N +PL	pavos	pavo +N +Masc +Pl	'ducks'
cat	cat +N +SG	pavo	pavo +N +Masc +Sg	'duck'
cities	city +N +Pl	bebo	beber +V +PInd +1P +Sg	'I drink'
geese	goose +N +Pl	canto	cantar +V +PInd +1P +Sg	'I sing'
goose	goose +N +Sg	canto	canto +N +Masc +Sg	'song'
goose	goose +V	puse	poner +V +Perf +1P +Sg	'I was able'
gooses	goose +V +1P +Sg	vino	venir +V +Perf +3P +Sg	'he/she came'
merging	merge +V +PresPart	vino	vino +N +Masc +Sg	'wine'
caught	catch +V +PastPart	lugar	lugar +N +Masc +Sg	'place'
caught	catch +V +Past			

Figure 3.2 Output of a morphological parse for some English and Spanish words. Spanish output modified from the Xerox XRCE finite-state language tools.

FEATURES

The second column contains the stem of each word as well as assorted morphological **features**. These features specify additional information about the stem. For example the feature `+N` means that the word is a noun; `+Sg` means it is singular, `+Pl` that it is plural. Morphological features will be referred to again in Ch. 5 and in more detail in Ch. 16; for now, consider `+Sg` to be a primitive unit that means "singular". Spanish has some features that don't occur in English; for example the nouns *lugar* and *pavo* are marked `+Masc` (masculine). Because Spanish nouns agree in gender with adjectives, knowing the gender of a noun will be important for tagging and parsing.

Note that some of the input forms (like *caught*, *goose*, *canto*, or *vino*) will be ambiguous between different morphological parses. For now, we will consider the goal of morphological parsing merely to list all possible parses. We will return to the task of disambiguating among morphological parses in Ch. 5.

In order to build a morphological parser, we'll need at least the following:

LEXICON

1. **lexicon:** the list of stems and affixes, together with basic information about them (whether a stem is a Noun stem or a Verb stem, etc.).

MORPHOTACTICS

2. **morphotactics:** the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word. For example, the fact that the English plural morpheme follows the noun rather than preceding it is a morphotactic fact.
3. **orthographic rules:** these **spelling rules** are used to model the changes that occur in a word, usually when two morphemes combine (e.g., the $y \rightarrow ie$ spelling rule discussed above that changes *city* + *-s* to *cities* rather than *citys*).

The next section will discuss how to represent a simple version of the lexicon just for the sub-problem of morphological recognition, including how to use FSAs to model morphotactic knowledge.

In following sections we will then introduce the finite-state transducer (FST) as a way of modeling morphological features in the lexicon, and addressing morphological parsing. Finally, we show how to use FSTs to model orthographic rules.

3.3 Building a Finite-State Lexicon

A lexicon is a repository for words. The simplest possible lexicon would consist of an explicit list of every word of the language (*every* word, i.e., including abbreviations ("AAA") and proper names ("Jane" or "Beijing")) as follows:

a, AAA, AA, Aachen, aardvark, aardwolf, aba, abaca, aback, . . .

Since it will often be inconvenient or impossible, for the various reasons we discussed above, to list every word in the language, computational lexicons are usually structured with a list of each of the stems and affixes of the language together with a representation of the morphotactics that tells us how they can fit together. There are many ways to model morphotactics; one of the most common is the finite-state automaton. A very simple finite-state model for English nominal inflection might look like Fig. 3.3.

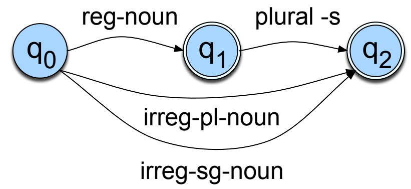


Figure 3.3 A finite-state automaton for English nominal inflection.

The FSA in Fig. 3.3 assumes that the lexicon includes regular nouns (**reg-noun**) that take the regular -*s* plural (e.g., *cat*, *dog*, *fox*, *aardvark*). These are the vast majority of English nouns since for now we will ignore the fact that the plural of words like *fox* have an inserted *e*: *foxes*. The lexicon also includes irregular noun forms that don't take *-s*, both singular **irreg-sg-noun** (*goose*, *mouse*) and plural **irreg-pl-noun** (*geese*, *mice*).

reg-noun	**irreg-pl-noun**	**irreg-sg-noun**	**plural**
fox	geese	goose	-s
cat	sheep	sheep	
aardvark	mice	mouse	

A similar model for English verbal inflection might look like Fig. 3.4.

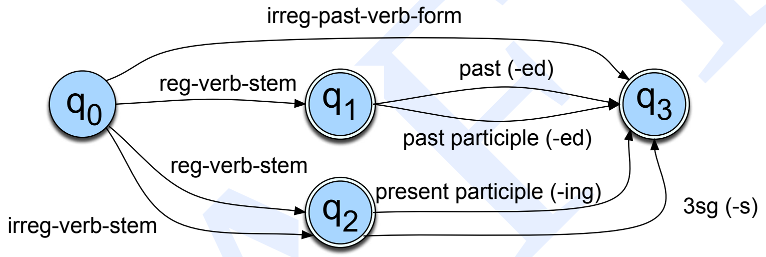


Figure 3.4 A finite-state automaton for English verbal inflection

This lexicon has three stem classes (reg-verb-stem, irreg-verb-stem, and irreg-past-verb-form), plus four more affix classes (*-ed* past, *-ed* participle, *-ing* participle, and third singular *-s*):

reg-verb-stem	**irreg-verb-stem**	**irreg-past-verb**	**past**	**past-part**	**pres-part**	**3sg**
walk	cut	caught	-ed	-ed	-ing	-s
fry	speak	ate				
talk	sing	eaten				
impeach		sang				

English derivational morphology is significantly more complex than English inflectional morphology, and so automata for modeling English derivation tend to be quite complex. Some models of English derivation, in fact, are based on the more complex context-free grammars of Ch. 12 (Sproat, 1993).

Consider a relatively simpler case of derivation: the morphotactics of English adjectives. Here are some examples from Antworth (1990):

big, bigger, biggest,
happy, happier, happiest, happily
unhappy, unhappier, unhappiest, unhappily
clear, clearer, clearest, clearly, unclear, unclearly

cool, cooler, coolest, coolly
red, redder, reddest
real, unreal, really

An initial hypothesis might be that adjectives can have an optional prefix (*un-*), an obligatory root (*big*, *cool*, etc.) and an optional suffix (*-er*, *-est*, or *-ly*). This might suggest the the FSA in Fig. 3.5.

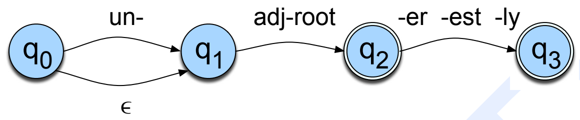


Figure 3.5 An FSA for a fragment of English adjective morphology: Antworth's Proposal #1.

Alas, while this FSA will recognize all the adjectives in the table above, it will also recognize ungrammatical forms like *unbig*, *unfast*, *oranger*, or *smally*. We need to set up classes of roots and specify their possible suffixes. Thus **adj-root**$_1$ would include adjectives that can occur with *un-* and *-ly* (*clear*, *happy*, and *real*) while **adj-root**$_2$ will include adjectives that can't (*big*, *small*), and so on.

This gives an idea of the complexity to be expected from English derivation. As a further example, we give in Figure 3.6 another fragment of an FSA for English nominal and verbal derivational morphology, based on Sproat (1993), Bauer (1983), and Porter (1980). This FSA models a number of derivational facts, such as the well known generalization that any verb ending in *-ize* can be followed by the nominalizing suffix *-ation* (Bauer, 1983; Sproat, 1993). Thus since there is a word *fossilize*, we can predict the word *fossilization* by following states q_0, q_1, and q_2. Similarly, adjectives ending in *-al* or *-able* at q_5 (*equal*, *formal*, *realizable*) can take the suffix *-ity*, or sometimes the suffix *-ness* to state q_6 (*naturalness*, *casualness*). We leave it as an exercise for the reader (Exercise 3.1) to discover some of the individual exceptions to many of these constraints, and also to give examples of some of the various noun and verb classes.

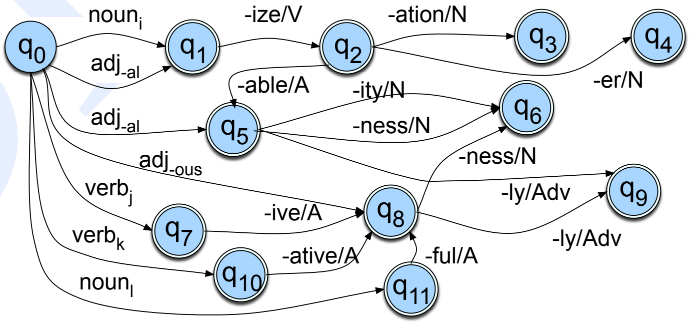


Figure 3.6 An FSA for another fragment of English derivational morphology.

We can now use these FSAs to solve the problem of **morphological recognition**; that is, of determining whether an input string of letters makes up a legitimate English word or not. We do this by taking the morphotactic FSAs, and plugging in each "sub-lexicon" into the FSA. That is, we expand each arc (e.g., the **reg-noun-stem** arc) with all the morphemes that make up the set of **reg-noun-stem**. The resulting FSA can then be defined at the level of the individual letter.

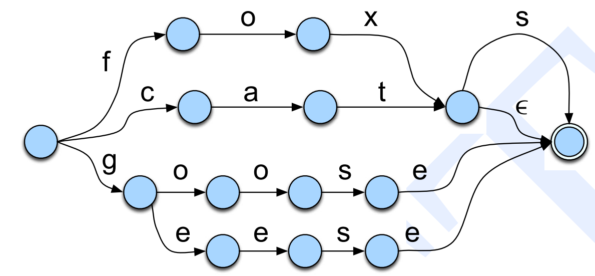


Figure 3.7 Expanded FSA for a few English nouns with their inflection. Note that this automaton will incorrectly accept the input *foxs*. We will see beginning on page 19 how to correctly deal with the inserted *e* in *foxes*.

Fig. 3.7 shows the noun-recognition FSA produced by expanding the Nominal Inflection FSA of Fig. 3.3 with sample regular and irregular nouns for each class. We can use Fig. 3.7 to recognize strings like *aardvarks* by simply starting at the initial state, and comparing the input letter by letter with each word on each outgoing arc, and so on, just as we saw in Ch. 2.

3.4 Finite-State Transducers

We've now seen that FSAs can represent the morphotactic structure of a lexicon, and can be used for word recognition. In this section we introduce the finite-state transducer. The next section will show how transducers can be applied to morphological parsing.

FST

A transducer maps between one representation and another; a **finite-state transducer** or **FST** is a type of finite automaton which maps between two sets of symbols. We can visualize an FST as a two-tape automaton which recognizes or generates *pairs* of strings. Intuitively, we can do this by labeling each arc in the finite-state machine with two symbol strings, one from each tape. Fig. 3.8 shows an example of an FST where each arc is labeled by an input and output string, separated by a colon.

The FST thus has a more general function than an FSA; where an FSA defines a formal language by defining a set of strings, an FST defines a *relation* between sets of strings. Another way of looking at an FST is as a machine that reads one string and generates another. Here's a summary of this four-fold way of thinking about transducers:

- **FST as recognizer:** a transducer that takes a pair of strings as input and outputs *accept* if the string-pair is in the string-pair language, and *reject* if it is not.

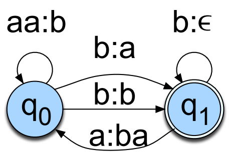


Figure 3.8 A finite-state transducer, modified from Mohri (1997).

- **FST as generator:** a machine that outputs pairs of strings of the language. Thus the output is a yes or no, and a pair of output strings.
- **FST as translator:** a machine that reads a string and outputs another string
- **FST as set relater:** a machine that computes relations between sets.

All of these have applications in speech and language processing. For morphological parsing (and for many other NLP applications), we will apply the FST as translator metaphor, taking as input a string of letters and producing as output a string of morphemes.

Let's begin with a formal definition. An FST can be formally defined with 7 parameters:

Q a finite set of N states $q_0, q_1, \ldots, q_{N-1}$

Σ a finite set corresponding to the input alphabet

Δ a finite set corresponding to the output alphabet

$q_0 \in Q$ the start state

$F \subseteq Q$ the set of final states

$\delta(q,w)$ the transition function or transition matrix between states; Given a state $q \in Q$ and a string $w \in \Sigma^*$, $\delta(q,w)$ returns a set of new states $Q' \in Q$. δ is thus a function from $Q \times \Sigma^*$ to 2^Q (because there are 2^Q possible subsets of Q). δ returns a set of states rather than a single state because a given input may be ambiguous in which state it maps to.

$\sigma(q,w)$ the output function giving the set of possible output strings for each state and input. Given a state $q \in Q$ and a string $w \in \Sigma^*$, $\sigma(q,w)$ gives a set of output strings, each a string $o \in \Delta^*$. σ is thus a function from $Q \times \Sigma^*$ to 2^{Δ^*}

REGULAR RELATIONS

Where FSAs are isomorphic to regular languages, FSTs are isomorphic to **regular relations**. Regular relations are sets of pairs of strings, a natural extension of the regular languages, which are sets of strings. Like FSAs and regular languages, FSTs and regular relations are closed under union, although in general they are not closed under difference, complementation and intersection (although some useful subclasses of FSTs *are* closed under these operations; in general FSTs that are not augmented with the ϵ are more likely to have such closure properties). Besides union, FSTs have two additional closure properties that turn out to be extremely useful:

INVERSION

- **inversion**: The inversion of a transducer T (T^{-1}) simply switches the input and output labels. Thus if T maps from the input alphabet I to the output alphabet O, T^{-1} maps from O to I.

COMPOSITION

- **composition**: If T_1 is a transducer from I_1 to O_1 and T_2 a transducer from O_1 to O_2, then $T_1 \circ T_2$ maps from I_1 to O_2.

Inversion is useful because it makes it easy to convert a FST-as-parser into an FST-as-generator.

Composition is useful because it allows us to take two transducers that run in series and replace them with one more complex transducer. Composition works as in algebra; applying $T_1 \circ T_2$ to an input sequence S is identical to applying T_1 to S and then T_2 to the result; thus $T_1 \circ T_2(S) = T_2(T_1(S))$.

Fig. 3.9, for example, shows the composition of `[a:b]+` with `[b:c]+` to produce `[a:c]+`.

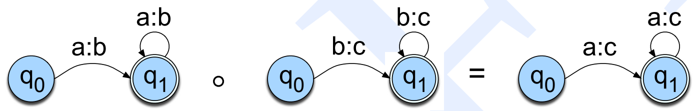


Figure 3.9 The composition of `[a:b]+` with `[b:c]+` to produce `[a:c]+`.

PROJECTION

The **projection** of an FST is the FSA that is produced by extracting only one side of the relation. We can refer to the projection to the left or upper side of the relation as the **upper** or **first** projection and the projection to the lower or right side of the relation as the **lower** or **second** projection.

3.4.1 Sequential Transducers and Determinism

Transducers as we have described them may be nondeterministic, in that a given input may translate to many possible output symbols. Thus using general FSTs requires the kinds of search algorithms discussed in Ch. 2, making FSTs quite slow in the general case. This suggests that it would nice to have an algorithm to convert a nondeterministic FST to a deterministic one. But while every non-deterministic FSA is equivalent to some deterministic FSA, not all finite-state transducers can be determinized.

SEQUENTIAL TRANSDUCERS

Sequential transducers, by contrast, are a subtype of transducers that are deterministic on their input. At any state of a sequential transducer, each given symbol of the input alphabet Σ can label at most one transition out of that state. Fig. 3.10 gives an example of a sequential transducer from Mohri (1997); note that here, unlike the transducer in Fig. 3.8, the transitions out of each state are deterministic based on the state and the input symbol. Sequential transducers can have epsilon symbols in the output string, but not on the input.

Sequential transducers are not necessarily sequential on their output. Mohri's transducer in Fig. 3.10 is not, for example, since two distinct transitions leaving state 0 have the same output (b). Since the inverse of a sequential transducer may thus not be sequential, we always need to specify the direction of the transduction when discussing sequentiality. Formally, the definition of sequential transducers modifies the δ and σ

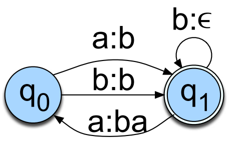


Figure 3.10 A sequential finite-state transducer, from Mohri (1997).

functions slightly; δ becomes a function from $Q \times \Sigma^*$ to Q (rather than to 2^Q), and σ becomes a function from $Q \times \Sigma^*$ to Δ^* (rather than to 2^{Δ^*}).

SUBSEQUENTIAL TRANSDUCER

One generalization of sequential transducers is the **subsequential transducer** (Schützenberger, 1977), which generates an additional output string at the final states, concatenating it onto the output produced so far.

What makes sequential and subsequential transducers important is their efficiency; because they are deterministic on input, they can be processed in time proportional to the number of symbols in the input (they are linear in their input length) rather than proportional to some much larger number which is a function of the number of states. Another advantage of subsequential transducers is that there exist efficient algorithms for their determinization (Mohri, 1997) and minimization (Mohri, 2000), extending the algorithms for determinization and minimization of finite-state automata that we saw in Ch. 2. also an equivalence algorithm.

While both sequential and subsequential transducers are deterministic and efficient, neither of them is able to handle ambiguity, since they transduce each input string to exactly one possible output string. Since ambiguity is a crucial property of natural language, it will be useful to have an extension of subsequential transducers that can deal with ambiguity, but still retain the efficiency and other useful properties of sequential transducers. One such generalization of subsequential transducers is the p-**subsequential** transducer. A p-**subsequential** transducer allows for $p(p \geq 1)$ final output strings to be associated with each final state (Mohri, 1996). They can thus handle a finite amount of ambiguity, which is useful for many NLP tasks. Fig. 3.11 shows an example of a 2-subsequential FST.

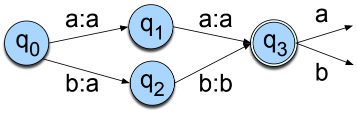


Figure 3.11 A 2-subsequential finite-state transducer, from Mohri (1997).

Mohri (1996, 1997) show a number of tasks whose ambiguity can be limited in this way, including the representation of dictionaries, the compilation of morphological and phonological rules, and local syntactic constraints. For each of these kinds of problems, he and others have shown that they are **p-subsequentializable**, and thus can be determinized and minimized. This class of transducers includes many, although not necessarily all, morphological rules.

3.5 FSTs for Morphological Parsing

Let's now turn to the task of morphological parsing. Given the input *cats*, for instance, we'd like to output *cat +N +Pl*, telling us that *cat* is a plural noun. Given the Spanish input *bebo* ('I drink'), we'd like to output *beber +V +PInd +1P +Sg* telling us that *bebo* is the present indicative first person singular form of the Spanish verb *beber*, 'to drink'.

In the **finite-state morphology** paradigm that we will use, we represent a word as a correspondence between a **lexical level**, which represents a concatenation of morphemes making up a word, and the **surface** level, which represents the concatenation of letters which make up the actual spelling of the word. Fig. 3.12 shows these two levels for (English) *cats*.

SURFACE

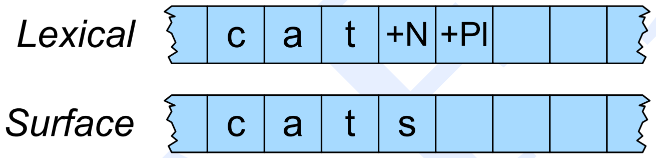


Figure 3.12 Schematic examples of the lexical and surface tapes; the actual transducers will involve intermediate tapes as well.

LEXICAL TAPE

For finite-state morphology it's convenient to view an FST as having two tapes. The **upper** or **lexical tape**, is composed from characters from one alphabet Σ. The **lower** or **surface** tape, is composed of characters from another alphabet Δ. In the **two-level morphology** of Koskenniemi (1983), we allow each arc only to have a single symbol from each alphabet. We can then combine the two symbol alphabets Σ and Δ to create a new alphabet, Σ', which makes the relationship to FSAs quite clear. Σ' is a finite alphabet of complex symbols. Each complex symbol is composed of an input-output pair $i:o$; one symbol i from the input alphabet Σ, and one symbol o from an output alphabet Δ, thus $\Sigma' \subseteq \Sigma \times \Delta$. Σ and Δ may each also include the epsilon symbol ϵ. Thus where an FSA accepts a language stated over a finite alphabet of single symbols, such as the alphabet of our sheep language:

(3.2)
$$\Sigma = \{b, a, !\}$$

an FST defined this way accepts a language stated over *pairs* of symbols, as in:

(3.3)
$$\Sigma' = \{a:a,\ b:b,\ !:!,\ a:!,\ a:\epsilon,\ \epsilon:!\}$$

FEASIBLE PAIRS

In two-level morphology, the pairs of symbols in Σ' are also called **feasible pairs**. Thus each feasible pair symbol $a:b$ in the transducer alphabet Σ' expresses how the symbol a from one tape is mapped to the symbol b on the other tape. For example $a:\epsilon$ means that an a on the upper tape will correspond to *nothing* on the lower tape. Just as for an FSA, we can write regular expressions in the complex alphabet Σ'. Since it's most common for symbols to map to themselves, in two-level morphology we call pairs like $a:a$ **default pairs**, and just refer to them by the single letter a.

DEFAULT PAIRS

We are now ready to build an FST morphological parser out of our earlier morphotactic FSAs and lexica by adding an extra "lexical" tape and the appropriate morphological features. Fig. 3.13 shows an augmentation of Fig. 3.3 with the nominal morphological features (`+Sg` and `+Pl`) that correspond to each morpheme. The symbol ˆ indicates a **morpheme boundary**, while the symbol # indicates a **word boundary**. The morphological features map to the empty string ϵ or the boundary symbols since there is no segment corresponding to them on the output tape.

MORPHEME BOUNDARY

#

WORD BOUNDARY

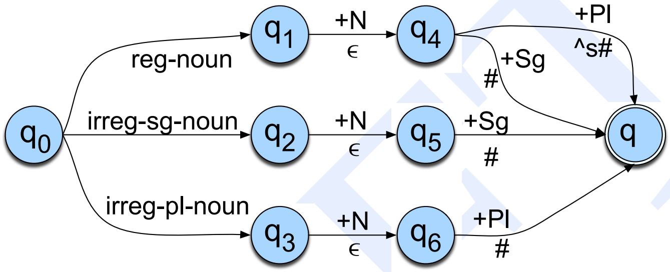


Figure 3.13 A schematic transducer for English nominal number inflection T_{num}. The symbols above each arc represent elements of the morphological parse in the lexical tape; the symbols below each arc represent the surface tape (or the intermediate tape, to be described later), using the morpheme-boundary symbol ˆ and word-boundary marker #. The labels on the arcs leaving q_0 are schematic, and need to be expanded by individual words in the lexicon.

In order to use Fig. 3.13 as a morphological noun parser, it needs to be expanded with all the individual regular and irregular noun stems, replacing the labels **reg-noun** etc. In order to do this we need to update the lexicon for this transducer, so that irregular plurals like *geese* will parse into the correct stem `goose +N +Pl`. We do this by allowing the lexicon to also have two levels. Since surface *geese* maps to lexical `goose`, the new lexical entry will be "`g:g o:e o:e s:s e:e`". Regular forms are simpler; the two-level entry for *fox* will now be "`f:f o:o x:x`", but by relying on the orthographic convention that `f` stands for `f:f` and so on, we can simply refer to it as `fox` and the form for *geese* as "`g o:e o:e s e`". Thus the lexicon will look only slightly more complex:

reg-noun	**irreg-pl-noun**	**irreg-sg-noun**
fox	g o:e o:e s e	goose
cat	sheep	sheep
aardvark	m o:i u:ϵ s:c e	mouse

The resulting transducer, shown in Fig. 3.14, will map plural nouns into the stem plus the morphological marker `+Pl`, and singular nouns into the stem plus the morphological marker `+Sg`. Thus a surface *cats* will map to `cat +N +Pl`. This can be viewed in feasible-pair format as follows:

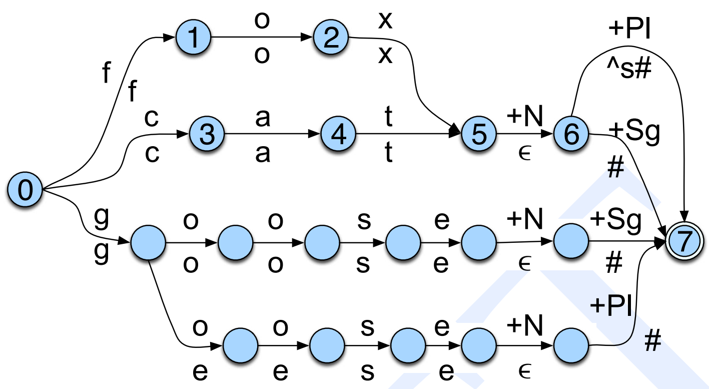


Figure 3.14 A fleshed-out English nominal inflection FST T_{lex}, expanded from T_{num} by replacing the three arcs with individual word stems (only a few sample word stems are shown).

```
c:c a:a t:t +N:ϵ +Pl:^s#
```

Since the output symbols include the morpheme and word boundary markers ˆ and #, the lower labels Fig. 3.14 do not correspond exactly to the surface level. Hence we refer to tapes with these morpheme boundary markers in Fig. 3.15 as **intermediate** tapes; the next section will show how the boundary marker is removed.

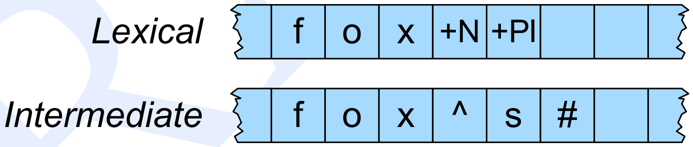


Figure 3.15 A schematic view of the lexical and intermediate tapes.

3.6 Transducers and Orthographic Rules

The method described in the previous section will successfully recognize words like *aardvarks* and *mice*. But just concatenating the morphemes won't work for cases where there is a spelling change; it would incorrectly reject an input like *foxes* and accept an input like *foxs*. We need to deal with the fact that English often requires spelling changes at morpheme boundaries by introducing **spelling rules** (or **orthographic rules**) This section introduces a number of notations for writing such rules and shows how to implement the rules as transducers. In general, the ability to implement rules as a transducer turns out to be useful throughout speech and language processing. Here's some spelling rules:

SPELLING RULES

Name	Description of Rule	Example
Consonant doubling	1-letter consonant doubled before *-ing*/*-ed*	beg/begging
E deletion	Silent e dropped before *-ing* and *-ed*	make/making
E insertion	e added after *-s*,*-z*,*-x*,*-ch*, *-sh* before *-s*	watch/watches
Y replacement	*-y* changes to *-ie* before *-s*, *-i* before *-ed*	try/tries
K insertion	verbs ending with *vowel* + *-c* add *-k*	panic/panicked

We can think of these spelling changes as taking as input a simple concatenation of morphemes (the "intermediate output" of the lexical transducer in Fig. 3.14) and producing as output a slightly-modified (correctly-spelled) concatenation of morphemes. Fig. 3.16 shows in schematic form the three levels we are talking about: lexical, intermediate, and surface. So for example we could write an E-insertion rule that performs the mapping from the intermediate to surface levels shown in Fig. 3.16. Such a rule might say something like "insert an *e* on the surface tape just when the lexical tape has a morpheme ending in *x* (or *z*, etc) and the next morpheme is *-s*". Here's a formalization of the rule:

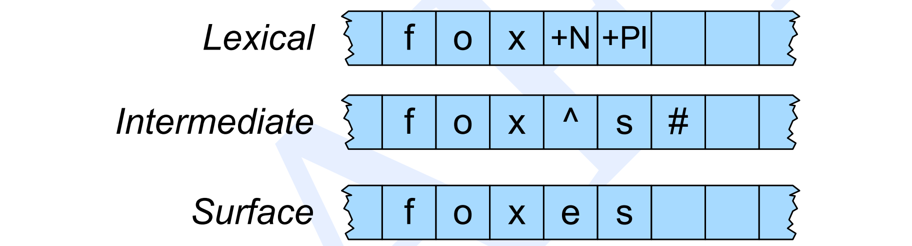


Figure 3.16 An example of the lexical, intermediate, and surface tapes. Between each pair of tapes is a two-level transducer; the lexical transducer of Fig. 3.14 between the lexical and intermediate levels, and the E-insertion spelling rule between the intermediate and surface levels. The E-insertion spelling rule inserts an *e* on the surface tape when the intermediate tape has a morpheme boundary ˆ followed by the morpheme *-s*.

$$\epsilon \rightarrow \text{e} \, / \left\{ \begin{array}{c} x \\ s \\ z \end{array} \right\} \hat{} \underline{\quad} \text{s\#} \qquad (3.4)$$

This is the rule notation of Chomsky and Halle (1968); a rule of the form $a \rightarrow b / c \underline{\quad} d$ means "rewrite *a* as *b* when it occurs between *c* and *d*". Since the symbol ϵ means an empty transition, replacing it means inserting something. Recall that the symbol ˆ indicates a morpheme boundary. These boundaries are deleted by including the symbol ˆ:ϵ in the default pairs for the transducer; thus morpheme boundary markers are deleted on the surface level by default. The # symbol is a special symbol that marks a word boundary. Thus (3.4) means "insert an *e* after a morpheme-final *x*, *s*, or *z*, and before the morpheme *s*". Fig. 3.17 shows an automaton that corresponds to this rule.

The idea in building a transducer for a particular rule is to express only the constraints necessary for that rule, allowing any other string of symbols to pass through

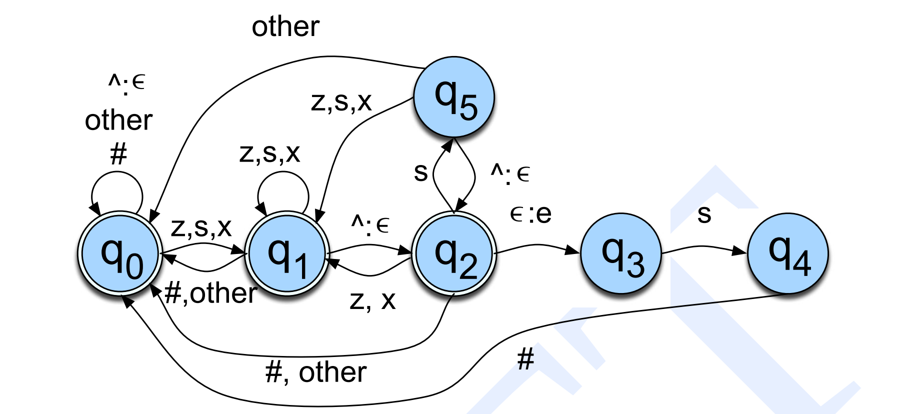


Figure 3.17 The transducer for the E-insertion rule of (3.4), extended from a similar transducer in Antworth (1990). We additionally need to delete the # symbol from the surface string; this can be done either by interpreting the symbol # as the pair #:ϵ, or by postprocessing the output to remove word boundaries.

unchanged. This rule is used to ensure that we can only see the ϵ:*e* pair if we are in the proper context. So state q_0, which models having seen only default pairs unrelated to the rule, is an accepting state, as is q_1, which models having seen a *z*, *s*, or *x*. q_2 models having seen the morpheme boundary after the *z*, *s*, or *x*, and again is an accepting state. State q_3 models having just seen the E-insertion; it is not an accepting state, since the insertion is only allowed if it is followed by the *s* morpheme and then the end-of-word symbol *#*.

The *other* symbol is used in Fig. 3.17 to safely pass through any parts of words that don't play a role in the E-insertion rule. *other* means "any feasible pair that is not in this transducer". So for example when leaving state q_0, we go to q_1 on the *z*, *s*, or *x* symbols, rather than following the *other* arc and staying in q_0. The semantics of *other* depends on what symbols are on other arcs; since # is mentioned on some arcs, it is (by definition) not included in *other*, and thus, for example, is explicitly mentioned on the arc from q_2 to q_0.

A transducer needs to correctly reject a string that applies the rule when it shouldn't. One possible bad string would have the correct environment for the E-insertion, but have no insertion. State q_5 is used to ensure that the *e* is always inserted whenever the environment is appropriate; the transducer reaches q_5 only when it has seen an *s* after an appropriate morpheme boundary. If the machine is in state q_5 and the next symbol is *#*, the machine rejects the string (because there is no legal transition on # from q_5). Fig. 3.18 shows the transition table for the rule which makes the illegal transitions explicit with the "–" symbol.

The next section will show a trace of this E-insertion transducer running on a sample input string.

State\Input	s:s	x:x	z:z	^:ϵ	ϵ:e	#	other
q_0:	1	1	1	0	-	0	0
q_1:	1	1	1	2	-	0	0
q_2:	5	1	1	0	3	0	0
q_3	4	-	-	-	-	-	-
q_4	-	-	-	-	-	0	-
q_5	1	1	1	2	-	-	0

Figure 3.18 The state-transition table for E-insertion rule of Fig. 3.17, extended from a similar transducer in Antworth (1990).

3.7 Combining FST Lexicon and Rules

We are now ready to combine our lexicon and rule transducers for parsing and generating. Fig. 3.19 shows the architecture of a two-level morphology system, whether used for parsing or generating. The lexicon transducer maps between the lexical level, with its stems and morphological features, and an intermediate level that represents a simple concatenation of morphemes. Then a host of transducers, each representing a single spelling rule constraint, all run in parallel so as to map between this intermediate level and the surface level. Putting all the spelling rules in parallel is a design choice; we could also have chosen to run all the spelling rules in series (as a long cascade), if we slightly changed each rule.

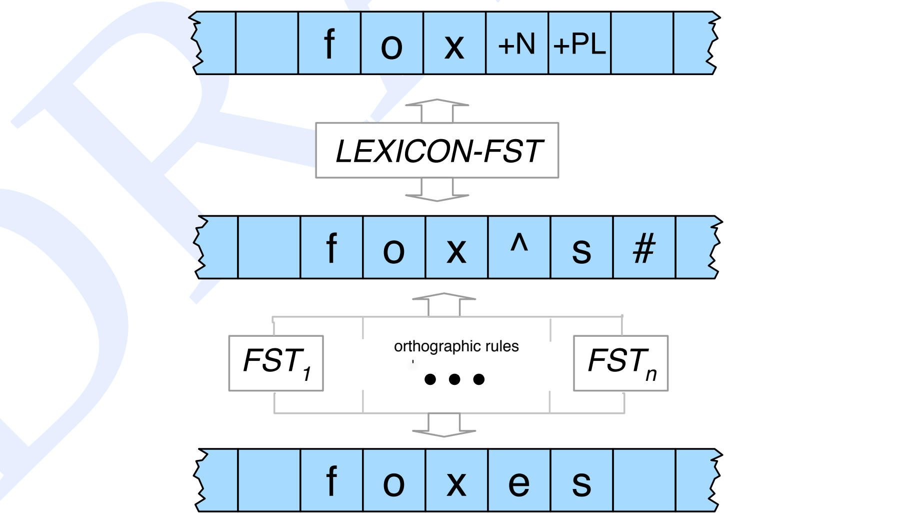


Figure 3.19 Generating or parsing with FST lexicon and rules

CASCADE

The architecture in Fig. 3.19 is a two-level **cascade** of transducers. Cascading two automata means running them in series with the output of the first feeding the input to the second. Cascades can be of arbitrary depth, and each level might be built out of many individual transducers. The cascade in Fig. 3.19 has two transducers in series: the transducer mapping from the lexical to the intermediate levels, and the collection of parallel transducers mapping from the intermediate to the surface level. The cascade can be run top-down to generate a string, or bottom-up to parse it; Fig. 3.20 shows a trace of the system *accepting* the mapping from `fox +N +PL` to *foxes*.

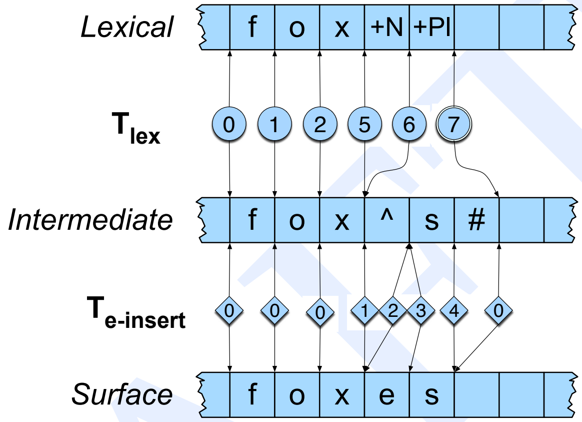


Figure 3.20 Accepting *foxes*: The lexicon transducer T_{lex} from Fig. 3.14 cascaded with the E-insertion transducer in Fig. 3.17.

The power of finite-state transducers is that the exact same cascade with the same state sequences is used when the machine is generating the surface tape from the lexical tape, or when it is parsing the lexical tape from the surface tape. For example, for generation, imagine leaving the Intermediate and Surface tapes blank. Now if we run the lexicon transducer, given `fox +N +PL`, it will produce *fox^s#* on the Intermediate tape via the same states that it accepted the Lexical and Intermediate tapes in our earlier example. If we then allow all possible orthographic transducers to run in parallel, we will produce the same surface tape.

AMBIGUITY

DISAMBIGUATING

Parsing can be slightly more complicated than generation, because of the problem of **ambiguity**. For example, *foxes* can also be a verb (albeit a rare one, meaning “to baffle or confuse”), and hence the lexical parse for *foxes* could be `fox +V +3Sg` as well as `fox +N +PL`. How are we to know which one is the proper parse? In fact, for ambiguous cases of this sort, the transducer is not capable of deciding. **Disambiguating** will require some external evidence such as the surrounding words. Thus *foxes* is likely to be a noun in the sequence *I saw two foxes yesterday*, but a verb in the sequence *That trickster foxes me every time!*. We will discuss such disambiguation algorithms in Ch. 5 and Ch. 20. Barring such external evidence, the best our transducer can do is just enumerate the possible choices; so we can transduce *fox^s#* into both `fox +V +3SG` and `fox +N +PL`.

There is a kind of ambiguity that we need to handle: local ambiguity that occurs during the process of parsing. For example, imagine parsing the input verb *assess*. After seeing *ass*, our E-insertion transducer may propose that the *e* that follows is inserted by the spelling rule (for example, as far as the transducer is concerned, we might have been parsing the word *asses*). It is not until we don't see the # after *asses*, but rather run into another *s*, that we realize we have gone down an incorrect path.

Because of this non-determinism, FST-parsing algorithms need to incorporate some sort of search algorithm. Exercise 3.7 asks the reader to modify the algorithm for non-deterministic FSA recognition in Fig. **??** in Ch. 2 to do FST parsing.

Note that many possible spurious segmentations of the input, such as parsing *assess* as ˆaˆsˆsesˆs will be ruled out since no entry in the lexicon will match this string.

INTERSECTION

Running a cascade, particularly one with many levels, can be unwieldy. Luckily, we've already seen how to compose a cascade of transducers in series into a single more complex transducer. Transducers in parallel can be combined by **automaton intersection**. The automaton intersection algorithm just takes the Cartesian product of the states, i.e., for each state q_i in machine 1 and state q_j in machine 2, we create a new state q_{ij}. Then for any input symbol a, if machine 1 would transition to state q_n and machine 2 would transition to state q_m, we transition to state q_{nm}. Fig. 3.21 sketches how this intersection ($\wedge$) and composition ($\circ$) process might be carried out.

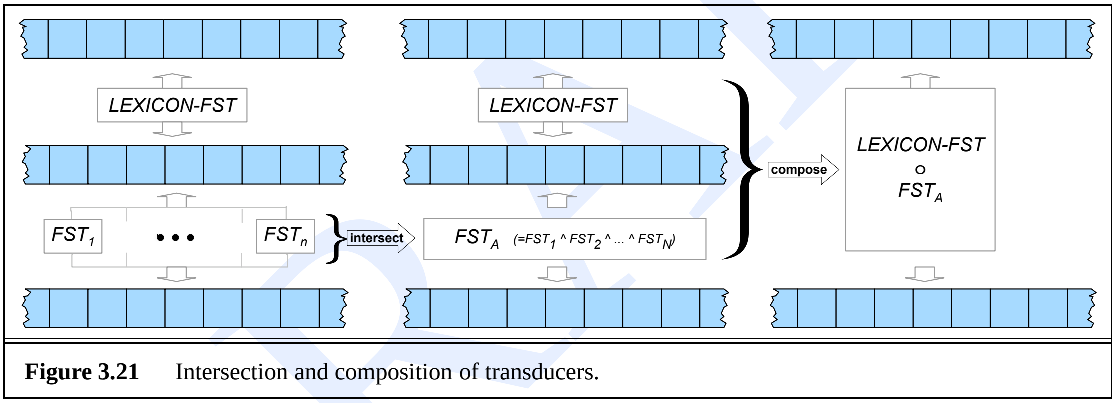


Figure 3.21 Intersection and composition of transducers.

Since there are a number of rule→FST compilers, it is almost never necessary in practice to write an FST by hand. Kaplan and Kay (1994) give the mathematics that define the mapping from rules to two-level relations, and Antworth (1990) gives details of the algorithms for rule compilation. Mohri (1997) gives algorithms for transducer minimization and determinization.

3.8 Lexicon-Free FSTs: The Porter Stemmer

KEYWORDS

While building a transducer from a lexicon plus rules is the standard algorithm for morphological parsing, there are simpler algorithms that don't require the large on-line lexicon demanded by this algorithm. These are used especially in Information Retrieval (IR) tasks like web search (Ch. 23), in which a query such as a Boolean combination of relevant **keywords** or phrases, e.g., (*marsupial OR kangaroo OR koala*) returns documents that have these words in them. Since a document with the word *marsupials* might not match the keyword *marsupial*, some IR systems first run a stemmer on the query and document words. Morphological information in IR is thus only used to determine that two words have the same stem; the suffixes are thrown away.

STEMMING

One of the most widely used such **stemming** algorithms is the simple and efficient Porter (1980) algorithm, which is based on a series of simple cascaded rewrite rules. Since cascaded rewrite rules are just the sort of thing that could be easily implemented as an FST, we think of the Porter algorithm as a lexicon-free FST stemmer (this idea will be developed further in the exercises (Exercise 3.6). The algorithm contains rules like these:

ATIONAL → ATE (e.g., relational → relate)
ING → ϵ if stem contains vowel (e.g., motoring → motor)

See Porter (1980) or Martin Porter's official homepage for the Porter stemmer for more details.

Krovetz (1993) showed that stemming tends to somewhat improve the performance of information retrieval, especially with smaller documents (the larger the document, the higher the chance the keyword will occur in the exact form used in the query). Nonetheless, not all IR engines use stemming, partly because of stemmer errors such as these noted by Krovetz:

Errors of Commission		Errors of Omission	
organization	organ	European	Europe
doing	doe	analysis	analyzes
generalization	generic	matrices	matrix
numerical	numerous	noise	noisy
policy	police	sparse	sparsity

3.9 Word and Sentence Tokenization

We have focused so far in this chapter on a problem of segmentation: how words can be segmented into morphemes. We turn now to a brief discussion of the very related problem of segmenting running text into words and sentences. This task is called **tokenization**.

TOKENIZATION

Word tokenization may seem very simple in a language like English that separates words via a special 'space' character. As we will see below, not every language does this (Chinese, Japanese, and Thai, for example, do not). But a closer examination will make it clear that whitespace is not sufficient by itself. Consider the following sentences from a Wall Street Journal and New York Times article, respectively:

```
Mr.  Sherwood said reaction to Sea Containers' proposal
has been "very positive." In New York Stock Exchange
composite trading yesterday, Sea Containers closed at
$62.625, up 62.5 cents.
```

```
‘‘I said, ‘what’re you?  Crazy?’  ’’  said Sadowsky.  ‘‘I
can’t afford to do that.’’
```

Segmenting purely on white-space would produce words like these:

```
cents.      said,      positive."      Crazy?
```

We could address these errors by treating punctuation, in addition to whitespace, as a word boundary. But punctuation often occurs word internally, in examples like *m.p.h,*, *Ph.D.*, *AT&T*, *cap’n*, *01/02/06*, and *google.com*. Similarly, assuming that we want *62.5* to be a word, we’ll need to avoid segmenting every period, since that will segment this into 62 and 5. Number expressions introduce other complications as well; while commas normally appear at word boundaries, commas are used inside numbers in English, every three digits: *555,500.50*. Languages differ on punctuation styles for numbers; many continental European languages like Spanish, French, and German, by contrast, uses a comma to mark the decimal point, and spaces (or sometimes periods) where English puts commas: *555 500,50*.

Another useful task a tokenizer can do for us is to expand clitic contractions that are marked by apostrophes, for example converting *what’re* above to the two tokens *what are*, and *we’re* to *we are*. This task is complicated by the fact that apostrophes are quite ambiguous, since they are also used as genitive markers (as in *the book’s over* or in *Containers’* above) or as quotative markers (as in *‘what’re you? Crazy?’* above). Such contractions occur in other alphabetic languages, including articles and pronouns in French (*j’ai*, *l’homme*). While these contractions tend to be clitics, not all clitics are marked this way with contraction. In general, then, segmenting and expanding clitics can be done as part of the process of morphological parsing presented earlier in the chapter.

Depending on the application, tokenization algorithms may also tokenize multiword expressions like *New York* or *rock ’n’ roll*, which requires a multiword expression dictionary of some sort. This makes tokenization intimately tied up with the task of detecting names, dates, and organizations, which is called *named entity detection* and will be discussed in Ch. 22.

SENTENCE SEGMENTATION

In addition to word segmentation, **sentence segmentation** is a crucial first step in text processing. Segmenting a text into sentences is generally based on punctuation. This is because certain kinds of punctuation (periods, question marks, exclamation points) tend to mark sentence boundaries. Question marks and exclamation points are relatively unambiguous markers of sentence boundaries. Periods, on the other hand, are more ambiguous. The period character ‘.’ is ambiguous between a sentence boundary marker and a marker of abbreviations like *Mr.* or *Inc.* The previous sentence that you just read showed an even more complex case of this ambiguity, in which the final period of *Inc.* marked both an abbreviation and the sentence boundary marker. For this reason, sentence tokenization and word tokenization tend to be addressed jointly.

In general, sentence tokenization methods work by building a binary classifier (based on a sequence of rules, or on machine learning) which decides if a period is part of the word or is a sentence boundary marker. In making this decision, it helps to know if the period is attached to a commonly used abbreviation; thus an abbreviation dictionary is useful.

State-of-the-art methods for sentence tokenization are based on machine learning and will be introduced in later chapters. But a useful first step can still be taken via a sequence of regular expressions. We introduce here the first part; a word tokenization algorithm. Fig. 3.22 gives a simple Perl word tokenization algorithm based on Grefenstette (1999). The algorithm is quite minimal, designed mainly to clarify many of the segmentation issues we discussed in previous paragraphs.

The algorithm consists of a sequence of regular expression substitution rules. The first rule separates unambiguous punctuation like question marks and parentheses. The next rule segments commas unless they are inside numbers. We then disambiguate apostrophes and pull off word-final clitics. Finally, we deal with periods, using a (toy) abbreviation dictionary and some heuristics for detecting other abbreviations.

```
#!/usr/bin/perl

$letternumber = "[A-Za-z0-9]";
$notletter = "[^A-Za-z0-9]";
$alwayssep = "[\\?!()\";/\\|‘]";
$clitic = "(’|:|-|’S|’D|’M|’LL|’RE|’VE|N’T|’s|’d|’m|’ll|’re|’ve|n’t)";

$abbr{"Co."} = 1; $abbr{"Dr."} = 1; $abbr{"Jan."} = 1; $abbr{"Feb."} = 1;

while ($line = <>){ # read the next line from standard input

    # put whitespace around unambiguous separators
    $line =~ s/$alwayssep/ $& /g;

    # put whitespace around commas that aren’t inside numbers
    $line =~ s/([^0-9]),/$1 , /g;
    $line =~ s/,([^0-9])/ , $1/g;

    # distinguish singlequotes from apostrophes by
    # segmenting off single quotes not preceded by letter
    $line =~ s/^’/$& /g;
    $line =~ s/($notletter)’/$1 ’/g;

    # segment off unambiguous word-final clitics and punctuation
    $line =~ s/$clitic$/ $&/g;
    $line =~ s/$clitic($notletter)/ $1 $2/g;

   # now deal with periods.  For each possible word
   @possiblewords=split(/\s+/,$line);
   foreach $word (@possiblewords) {
      # if it ends in a period,
      if (($word =~ /$letternumber\./)
             && !($abbr{$word})  # and isn’t on the abbreviation list
                # and isn’t a sequence of letters and periods (U.S.)
                # and doesn’t resemble an abbreviation (no vowels: Inc.)
             && !($word =~
                 /^([A-Za-z]\.([A-Za-z]\.)+|[A-Z][bcdfghj-nptvxz]+\.)$/)) {
          # then segment off the period
          $word =~ s/\.$/ \./;
      }
      # expand clitics
      $word =~s/’ve/have/;
      $word =~s/’m/am/;
      print $word," ";
   }
 print "\n";
}
```

Figure 3.22 A sample English tokenization script, adapted from Grefenstette (1999) and Palmer (2000). A real script would have a longer abbreviation dictionary.

The fact that a simple tokenizer can be build with such simple regular expression patterns suggest that tokenizers like the one in Fig. 3.22 can be easily implemented in FSTs. This is indeed the case, and (Karttunen et al., 1996) and (Beesley and Karttunen, 2003) give descriptions of such FST-based tokenizers.

3.9.1 Segmentation in Chinese

We mentioned above that some languages, including Chinese, Japanese, and Thai, do not use spaces to mark potential word-boundaries. Alternative segmentation methods are used for these languages.

In Chinese, for example, words are composed of characters known as *hanzi*. Each character generally represents a single morpheme and is pronounceable as a single syllable. Words on average are about 2.4 characters long. A simple algorithm that does remarkably well for segmenting Chinese, and is often used as a baseline comparison for more advanced methods, is a version of greedy search called **maximum matching** or sometimes **maxmatch**. The algorithm requires a dictionary (wordlist) of the language.

MAXIMUM MATCHING

The maximum matching algorithm starts by pointing at the beginning of a string. It chooses the longest word in the dictionary that matches the input at the current position. The pointer is then advanced past each character in that word. If no word matches, the pointer is instead advanced one character (creating a one-character word). The algorithm is then iteratively applied again starting from the new pointer position. To help visualize this algorithm, Palmer (2000) gives an English analogy, which approximates the Chinese situation by removing the spaces from the English sentence *the table down there* to produce *thetabledownthere*. The maximum match algorithm (given a long English dictionary) would first match the word *theta* in the input, since that is the longest sequence of letters that matches a dictionary word. Starting from the end of *theta*, the longest matching dictionary word is *bled*, followed by *own* and then *there*, producing the incorrect sequence *theta bled own there*.

The algorithm seems to work better in Chinese (with such short words) than in languages like English with long words, as our failed example shows. Even in Chinese, however, maxmatch has a number of weakness, particularly with **unknown words** (words not in the dictionary) or **unknown genres** (genres which differ a lot from the assumptions made by the dictionary builder).

There is an annual competition (technically called a **bakeoff**) for Chinese segmentation algorithms. These most successful modern algorithms for Chinese word segmentation are based on machine learning from hand-segmented training sets. We will return to these algorithms after we introduce probabilistic methods in Ch. 5.

3.10 Detecting and Correcting Spelling Errors

> ALGERNON: *But my own sweet Cecily, I have never written you any letters.*
> CECILY: *You need hardly remind me of that, Ernest. I remember only too well that I was forced to write your letters for you. I wrote always three times a week, and sometimes oftener.*
> ALGERNON: *Oh, do let me read them, Cecily?*
> CECILY: *Oh, I couldn't possibly. They would make you far too conceited. The three you wrote me after I had broken off the engagement are so beautiful, and so badly spelled, that even now I can hardly read them without crying a little.*
> Oscar Wilde, *The Importance of being Earnest*

Like Oscar Wilde's fabulous Cecily, a lot of people were thinking about spelling during the last turn of the century. Gilbert and Sullivan provide many examples. *The Gondoliers*' Giuseppe, for example, worries that his private secretary is "shaky in his spelling" while *Iolanthe*'s Phyllis can "spell every word that she uses". Thorstein Veblen's explanation (in his 1899 classic *The Theory of the Leisure Class*) was that a main purpose of the "archaic, cumbrous, and ineffective" English spelling system was to be difficult enough to provide a test of membership in the leisure class. Whatever the social role of spelling, we can certainly agree that many more of us are like Cecily than like Phyllis. Estimates for the frequency of spelling errors in human typed text vary from 0.05% of the words in carefully edited newswire text to 38% in difficult applications like telephone directory lookup (Kukich, 1992).

In this section we introduce the problem of detecting and correcting spelling errors. Since the standard algorithm for spelling error correction is probabilistic, we will continue our spell-checking discussion later in Ch. 5 after we define the probabilistic noisy channel model.

OCR

The detection and correction of spelling errors is an integral part of modern word-processors and search engines, and is also important in correcting errors in **optical character recognition** (**OCR**), the automatic recognition of machine or hand-printed characters, and **on-line handwriting recognition**, the recognition of human printed or cursive handwriting as the user is writing.

Following Kukich (1992), we can distinguish three increasingly broader problems:

1. **non-word error detection:** detecting spelling errors that result in non-words (like *graffe* for *giraffe*).
2. **isolated-word error correction:** correcting spelling errors that result in non-words, for example correcting *graffe* to *giraffe*, but looking only at the word in isolation.
3. **context-dependent error detection and correction:** using the context to help detect and correct spelling errors even if they accidentally result in an actual word of English (**real-word errors**). This can happen from typographical errors (insertion, deletion, transposition) which accidentally produce a real word (e.g., *there* for *three*), or because the writer substituted the wrong spelling of a

REAL-WORD ERRORS

homophone or near-homophone (e.g., *dessert* for *desert*, or *piece* for *peace*).

Detecting non-word errors is generally done by marking any word that is not found in a dictionary. For example, the misspelling *graffe* above would not occur in a dictionary. Some early research (Peterson, 1986) had suggested that such spelling dictionaries would need to be kept small, because large dictionaries contain very rare words that resemble misspellings of other words. For example the rare words *wont* or *veery* are also common misspelling of *won't* and *very*. In practice, Damerau and Mays (1989) found that while some misspellings were hidden by real words in a larger dictionary, the larger dictionary proved more help than harm by avoiding marking rare words as errors. This is especially true with probabilistic spell-correction algorithms that can use word frequency as a factor. Thus modern spell-checking systems tend to be based on large dictionaries.

The finite-state morphological parsers described throughout this chapter provide a technology for implementing such large dictionaries. By giving a morphological parser for a word, an FST parser is inherently a word recognizer. Indeed, an FST morphological parser can be turned into an even more efficient FSA word recognizer by using the **projection** operation to extract the lower-side language graph. Such FST dictionaries also have the advantage of representing productive morphology like the English *-s* and *-ed* inflections. This is important for dealing with new legitimate combinations of stems and inflection . For example, a new stem can be easily added to the dictionary, and then all the inflected forms are easily recognized. This makes FST dictionaries especially powerful for spell-checking in morphologically rich languages where a single stem can have tens or hundreds of possible surface forms.[5]

FST dictionaries can thus help with non-word error detection. But how about error correction? Algorithms for isolated-word error correction operate by finding words which are the likely source of the errorful form. For example, correcting the spelling error *graffe* requires searching through all possible words like *giraffe*, *graf*, *craft*, *grail*, etc, to pick the most likely source. To choose among these potential sources we need a **distance metric** between the source and the surface error. Intuitively, *giraffe* is a more likely source than *grail* for *graffe*, because *giraffe* is closer in spelling to *graffe* than *grail* is to *graffe*. The most powerful way to capture this similarity intuition requires the use of probability theory and will be discussed in Ch. 4. The algorithm underlying this solution, however, is the non-probabilistic **minimum edit distance** algorithm that we introduce in the next section.

3.11 Minimum Edit Distance

DISTANCE

Deciding which of two words is closer to some third word in spelling is a special case of the general problem of **string distance**. The distance between two strings is a measure of how alike two strings are to each other.

[5] Early spelling error detectors for English, by contrast, simply allowed any word to have any suffix – thus Unix SPELL accepts bizarre prefixed words like *misclam* and *antiundoggingly* and suffixed words based on *the* like *thehood* and *theness*.

MINIMUM EDIT DISTANCE

Many important algorithms for finding string distance rely on some version of the **minimum edit distance** algorithm, named by Wagner and Fischer (1974) but independently discovered by many people; see the History section of Ch. 6 for a discussion of the history of these algorithms. The minimum edit distance between two strings is the minimum number of editing operations (insertion, deletion, substitution) needed to transform one string into another. For example the gap between the words *intention* and *execution* is five operations, shown in Fig. 3.23 as an **alignment** between the two strings. Given two sequences, an **alignment** is a correspondance between substrings of the two sequences. Thus I aligns with the empty string, N with E, T with X, and so on. Beneath the aligned strings is another representation; a series of symbols expressing an **operation list** for converting the top string into the bottom string; d for deletion, s for substitution, i for insertion.

ALIGNMENT

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Figure 3.23 Representing the minimum edit distance between two strings as an **alignment**. The final row gives the operation list for converting the top string into the bottom string; d for deletion, s for substitution, i for insertion.

We can also assign a particular cost or weight to each of these operations. The **Levenshtein** distance between two sequences is the simplest weighting factor in which each of the three operations has a cost of 1 (Levenshtein, 1966).[6] Thus the Levenshtein distance between *intention* and *execution* is 5. Levenshtein also proposed an alternate version of his metric in which each insertion or deletion has a cost of one, and substitutions are not allowed (equivalent to allowing substitution, but giving each substitution a cost of 2, since any substitution can be represented by one insertion and one deletion). Using this version, the Levenshtein distance between *intention* and *execution* is 8.

DYNAMIC PROGRAMMING

The minimum edit distance is computed by **dynamic programming**. Dynamic programming is the name for a class of algorithms, first introduced by Bellman (1957), that apply a table-driven method to solve problems by combining solutions to subproblems. This class of algorithms includes the most commonly-used algorithms in speech and language processing; besides minimum edit distance, these include the **Viterbi** and **forward** algorithms (Ch. 6), and the **CYK** and **Earley** algorithm (Ch. 13).

The intuition of a dynamic programming problem is that a large problem can be solved by properly combining the solutions to various subproblems. For example, consider the sequence or "path" of transformed words that comprise the minimum edit distance between the strings *intention* and *execution* shown in Fig. 3.24.

Imagine some string (perhaps it is *exention*) that is in this optimal path (whatever it is). The intuition of dynamic programming is that if *exention* is in the optimal

[6] We assume that the substitution of a letter for itself, e.g. substitution *t* for *t*, has zero cost.

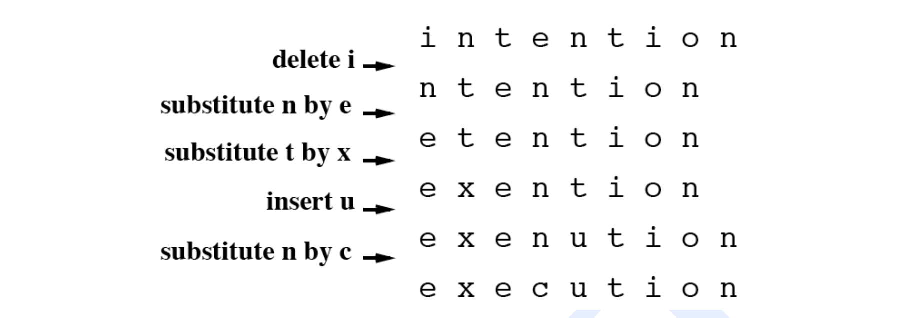


Figure 3.24 Operation list transforming *intention* to *execution* (after Kruskal 1983)

operation-list, then the optimal sequence must also include the optimal path from *intention* to *exention*. Why? If there were a shorter path from *intention* to *exention* then we could use it instead, resulting in a shorter overall path, and the optimal sequence wouldn't be optimal, thus leading to a contradiction.

Dynamic programming algorithms for sequence comparison work by creating a distance matrix with one column for each symbol in the target sequence and one row for each symbol in the source sequence (i.e., target along the bottom, source along the side). For minimum edit distance, this matrix is the *edit-distance* matrix. Each cell *edit-distance*[*i*,*j*] contains the distance between the first *i* characters of the target and the first *j* characters of the source. Each cell can be computed as a simple function of the surrounding cells; thus starting from the beginning of the matrix it is possible to fill in every entry. The value in each cell is computed by taking the minimum of the three possible paths through the matrix which arrive there:

$$distance[i,j] = \min \begin{cases} distance[i-1,j] + \text{ins-cost}(target_{i-1}) \\ distance[i-1,j-1] + \text{subst-cost}(source_{j-1}, target_{i-1}) \\ distance[i,j-1] + \text{del-cost}(source_{j-1})) \end{cases} \tag{3.5}$$

The algorithm itself is summarized in Fig. 3.25, while Fig. 3.26 shows the results of applying the algorithm to the distance between *intention* and *execution* assuming the version of Levenshtein distance in which the insertions and deletions each have a cost of 1 (ins-cost(·) = del-cost(·) = 1), and substitutions have a cost of 2 (except substitution of identical letters has zero cost).

Knowing the minimum edit distance is useful for algorithms like finding potential spelling error corrections. But the edit distance algorithm is important in another way; with a small change, it can also provide the minimum cost **alignment** between two strings. Aligning two strings is useful throughout speech and language processing. In speech recognition, minimum edit distance alignment is used to compute word error rate in speech recognition (Ch. 9). Alignment plays a role in machine translation, in which sentences in a parallel corpus (a corpus with a text in two languages) need to be matched up to each other.

In order to extend the edit distance algorithm to produce an alignment, we can start by visualizing an alignment as a path through the edit distance matrix. Fig. 3.27 shows

```
function MIN-EDIT-DISTANCE(target, source) returns min-distance

  $n \leftarrow$ LENGTH(target)
  $m \leftarrow$ LENGTH(source)
  Create a distance matrix distance[n+1,m+1]
  Initialize the zeroth row and column to be the distance from the empty string
    distance[0,0] = 0
    for each column $i$ from 1 to $n$ do
      distance[i,0] $\leftarrow$ distance[i-1,0] + ins-cost(target[i])
    for each row $j$ from 1 to $m$ do
      distance[0,j] $\leftarrow$ distance[0,j-1] + del-cost(source[j])
  for each column $i$ from 1 to $n$ do
    for each row $j$ from 1 to $m$ do
      distance[i,j] $\leftarrow$ MIN( distance[$i-1,j$] + ins-cost(target$_{i-1}$),
                                distance[$i-1,j-1$] + subst-cost(source$_{j-1}$, target$_{i-1}$),
                                distance[$i,j-1$] + del-cost(source$_{j-1}$))
  return distance[n,m]
```

Figure 3.25 The minimum edit distance algorithm, an example of the class of dynamic programming algorithms. The various costs can either be fixed (e.g. $\forall x, \text{ins-cost}(x) = 1$), or can be specific to the letter (to model the fact that some letters are more likely to be inserted than others). We assume that there is no cost for substituting a letter for itself (i.e. $\text{subst-cost}(x,x) = 0$).

n	*9*	8	9	10	11	12	11	10	9	8
o	*8*	7	8	9	10	11	10	9	8	9
i	*7*	6	7	8	9	10	9	8	9	10
t	*6*	5	6	7	8	9	8	9	10	11
n	*5*	4	5	6	7	8	9	10	11	10
e	*4*	3	4	5	6	7	8	9	10	9
t	*3*	4	5	6	7	8	7	8	9	8
n	*2*	3	4	5	6	7	8	7	8	7
i	*1*	2	3	4	5	6	7	6	7	8
#	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
	#	e	x	e	c	u	t	i	o	n

Figure 3.26 Computation of minimum edit distance between *intention* and *execution* via algorithm of Fig. 3.25, using Levenshtein distance with cost of 1 for insertions or deletions, 2 for substitutions. In italics are the initial values representing the distance from the empty string.

this path with the boldfaced cell. Each boldfaced cell represents an alignment of a pair of letters in the two strings. If two boldfaced cells occur in the same row, there will be an insertion in going from the source to the target; two boldfaced cells in the same column indicates a deletion.

Fig. 3.27 also shows the intuition of how to compute this alignment path. The computation proceeds in two steps. In the first step, we augment the minimum edit distance algorithm to store backpointers in each cell. The backpointer from a cell points to the previous cell (or cells) that were extended from in entering the current cell. We've shown a schematic of these backpointers in Fig. 3.27, after a similar diagram in Gusfield (1997). Some cells have multiple backpointers, because the minimum extension could have come from multiple previous cells. In the second step, we perform a **backtrace**. In a backtrace, we start from the last cell (at the final row and column), and follow the pointers back through the dynamic programming matrix. Each complete path between the final cell and the initial cell is a minimum distance alignment. Exercise 3.12 asks you to modify the minimum edit distance algorithm to store the pointers and compute the backtrace to output an alignment.

BACKTRACE

n	9	↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙←↓ 12	↓ 11	↓ 10	↓ 9	↙ **8**
o	8	↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↓ 10	↓ 9	↙ **8**	← 9
i	7	↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9	↙ **8**	← 9	← 10
t	6	↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙ **8**	← 9	← 10	←↓ 11
n	5	↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ **8**	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙↓ 10
e	4	↙ 3	← 4	↙← **5**	← **6**	← 7	←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9
t	3	↙←↓ 4	↙←↓ **5**	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙ 7	←↓ 8	↙←↓ 9	↓ 8
n	2	↙←↓ **3**	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↓ 7	↙←↓ 8	↙ 7
i	**1**	↙←↓ 2	↙←↓ 3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙ 6	← 7	← 8
#	**0**	1	2	3	4	5	6	7	8	9
	#	e	x	e	c	u	t	i	o	n

Figure 3.27 When entering a value in each cell, we mark which of the 3 neighboring cells we came from with up to three arrows. After the table is full we compute an **alignment** (minimum edit path) via a **backtrace**, starting at the **8** in the upper right corner and following the arrows. The sequence of boldfaced distances represents one possible minimum cost alignment between the two strings.

There are various publicly available packages to compute edit distance, including UNIX `diff`, and the NIST `sclite` program (NIST, 2005); Minimum edit distance can also be augmented in various ways. The Viterbi algorithm, for example, is an extension of minimum edit distance which uses probabilistic definitions of the operations. In this case instead of computing the "minimum edit distance" between two strings, we are interested in the "maximum probability alignment" of one string with another. The Viterbi algorithm is crucial in probabilistic tasks like speech recognition and part-of-speech tagging.

3.12 Human Morphological Processing

In this section we briefly survey psycholinguistic studies on how multi-morphemic words are represented in the minds of speakers of English. For example, consider the word *walk* and its inflected forms *walks*, and *walked*. Are all three in the human lexicon? Or merely *walk* along with *-ed* and *-s*? How about the word *happy* and its derived

FULL LISTING

MINIMUM REDUNDANCY

forms *happily* and *happiness*? We can imagine two ends of a theoretical spectrum of representations. The **full listing** hypothesis proposes that all words of a language are listed in the mental lexicon without any internal morphological structure. On this view, morphological structure is simply an epiphenomenon, and *walk*, *walks*, *walked*, *happy*, and *happily* are all separately listed in the lexicon. This hypothesis is certainly untenable for morphologically complex languages like Turkish. The **minimum redundancy** hypothesis suggests that only the constituent morphemes are represented in the lexicon, and when processing *walks*, (whether for reading, listening, or talking) we must access both morphemes (*walk* and *-s*) and combine them.

Some of the earliest evidence that the human lexicon represents at least some morphological structure comes from **speech errors**, also called **slips of the tongue**. In conversational speech, speakers often mix up the order of the words or sounds:

> if you break it it'll drop

In slips of the tongue collected by Fromkin and Ratner (1998) and Garrett (1975), inflectional and derivational affixes can appear separately from their stems. The ability of these affixes to be produced separately from their stem suggests that the mental lexicon contains some representation of morphological structure.

> it's not only us who have screw looses (for "screws loose")
> words of rule formation (for "rules of word formation")
> easy enoughly (for "easily enough")

PRIMED

More recent experimental evidence suggests that neither the full listing nor the minimum redundancy hypotheses may be completely true. Instead, it's possible that some but not all morphological relationships are mentally represented. Stanners et al. (1979), for example, found that some derived forms (*happiness*, *happily*) seem to be stored separately from their stem (*happy*), but that regularly inflected forms (*pouring*) are not distinct in the lexicon from their stems (*pour*). They did this by using a repetition priming experiment. In short, repetition priming takes advantage of the fact that a word is recognized faster if it has been seen before (if it is **primed**). They found that *lifting* primed *lift*, and *burned* primed *burn*, but for example *selective* didn't prime *select*. Marslen-Wilson et al. (1994) found that *spoken* derived words can prime their stems, but only if the meaning of the derived form is closely related to the stem. For example *government* primes *govern*, but *department* does not prime *depart*. Marslen-Wilson et al. (1994) represent a model compatible with their own findings as follows:

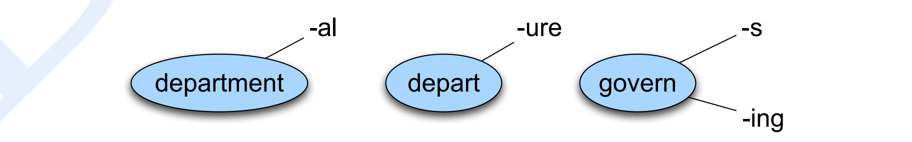


Figure 3.28 Marslen-Wilson et al. (1994) result: Derived words are linked to their stems only if semantically related.

In summary, these early results suggest that (at least) productive morphology like inflection does play an online role in the human lexicon. More recent studies have

MORPHOLOGICAL FAMILY SIZE

shown effects of non-inflectional morphological structure on word reading time as well, such as the **morphological family size**. The morphological family size of a word is the number of other multimorphemic words and compounds in which it appears; the family for *fear*, for example, includes *fearful*, *fearfully*, *fearfulness*, *fearless*, *fearlessly*, *fearlessness*, *fearsome*, and *godfearing* (according to the CELEX database), for a total size of 9. Baayen and colleagues (Baayen et al., 1997; De Jong et al., 2002; Moscoso del Prado Martín et al., 2004) have shown that words with a larger morphological family size are recognized faster. Recent work has further shown that word recognition speed is effected by the total amount of **information** (or **entropy**) contained by the morphological paradigm (Moscoso del Prado Martín et al., 2004); entropy will be introduced in the next chapter.

3.13 SUMMARY

This chapter introduced **morphology**, the arena of language processing dealing with the subparts of words, and the **finite-state transducer**, the computational device that is important for morphology but will also play a role in many other tasks in later chapters. We also introduced **stemming**, **word and sentence tokenization**, and **spelling error detection**.

Here's a summary of the main points we covered about these ideas:

- **Morphological parsing** is the process of finding the constituent **morphemes** in a word (e.g., `cat +N +PL` for *cats*).
- English mainly uses **prefixes** and **suffixes** to express **inflectional** and **derivational** morphology.
- English **inflectional** morphology is relatively simple and includes person and number agreement (*-s*) and tense markings (*-ed* and *-ing*).
- English **derivational** morphology is more complex and includes suffixes like *-ation*, *-ness*, *-able* as well as prefixes like *co-* and *re-*.
- Many constraints on the English **morphotactics** (allowable morpheme sequences) can be represented by finite automata.
- **Finite-state transducers** are an extension of finite-state automata that can generate output symbols.
- Important operations for FSTs include **composition**, **projection**, and **intersection**.
- **Finite-state morphology** and **two-level morphology** are applications of finite-state transducers to morphological representation and parsing.
- **Spelling rules** can be implemented as transducers.
- There are automatic transducer-compilers that can produce a transducer for any simple rewrite rule.
- The lexicon and spelling rules can be combined by **composing** and **intersecting** various transducers.
- The **Porter algorithm** is a simple and efficient way to do **stemming**, stripping off affixes. It is not as accurate as a transducer model that includes a lexicon,

but may be preferable for applications like **information retrieval** in which exact morphological structure is not needed.

- **Word tokenization** can be done by simple regular expressions substitutions or by transducers.
- **Spelling error detection** is normally done by finding words which are not in a dictionary; an FST dictionary can be useful for this.
- The **minimum edit distance** between two strings is the minimum number of operations it takes to edit one into the other. Minimum edit distance can be computed by **dynamic programming**, which also results in an **alignment** of the two strings.

Bibliographical and Historical Notes

Despite the close mathematical similarity of finite-state transducers to finite-state automata, the two models grew out of somewhat different traditions. Ch. 2 described how the finite automaton grew out of Turing's (1936) model of algorithmic computation, and McCulloch and Pitts finite-state-like models of the neuron. The influence of the Turing machine on the transducer was somewhat more indirect. Huffman (1954) proposed what was essentially a state-transition table to model the behavior of sequential circuits, based on the work of Shannon (1938) on an algebraic model of relay circuits. Based on Turing and Shannon's work, and unaware of Huffman's work, Moore (1956) introduced the term **finite automaton** for a machine with a finite number of states with an alphabet of input symbols and an alphabet of output symbols. Mealy (1955) extended and synthesized the work of Moore and Huffman.

The finite automata in Moore's original paper, and the extension by Mealy differed in an important way. In a Mealy machine, the input/output symbols are associated with the transitions between states. In a Moore machine, the input/output symbols are associated with the state. The two types of transducers are equivalent; any Moore machine can be converted into an equivalent Mealy machine and vice versa. Further early work on finite-state transducers, sequential transducers, and so on, was conducted by Salomaa (1973), Schützenberger (1977).

Early algorithms for morphological parsing used either the **bottom-up** or **top-down** methods that we will discuss when we turn to parsing in Ch. 13. An early bottom-up **affix-stripping** approach as Packard's (1973) parser for ancient Greek which iteratively stripped prefixes and suffixes off the input word, making note of them, and then looked up the remainder in a lexicon. It returned any root that was compatible with the stripped-off affixes. AMPLE (A Morphological Parser for Linguistic Exploration) (Weber and Mann, 1981; Weber et al., 1988; Hankamer and Black, 1991) is another early bottom-up morphological parser. Hankamer's (1986) keCi is a an early top-down *generate-and-test* or *analysis-by-synthesis* morphological parser for Turkish which is guided by a finite-state representation of Turkish morphemes. The program begins with a morpheme that might match the left edge of the word, and applies every possible phonological rule to it, checking each result against the input. If one of the outputs succeeds, the program then follows the finite-state morphotactics to the next morpheme and tries to continue matching the input.

The idea of modeling spelling rules as finite-state transducers is really based on Johnson's (1972) early idea that phonological rules (to be discussed in Ch. 7) have finite-state properties. Johnson's insight unfortunately did not attract the attention of the community, and was independently discovered by Ronald Kaplan and Martin Kay, first in an unpublished talk (Kaplan and Kay, 1981) and then finally in print (Kaplan and Kay, 1994) (see page **??** for a discussion of multiple independent discoveries). Kaplan and Kay's work was followed up and most fully worked out by Koskenniemi (1983), who described finite-state morphological rules for Finnish. Karttunen (1983) built a program called KIMMO based on Koskenniemi's models. Antworth (1990) gives many details of two-level morphology and its application to English. Besides Koskenniemi's work on Finnish and that of Antworth (1990) on English, two-level or other finite-state models of morphology have been worked out for many languages, such as Turkish (Oflazer, 1993) and Arabic (Beesley, 1996). Barton et al. (1987) bring up some computational complexity problems with two-level models, which are responded to by Koskenniemi and Church (1988). Readers with further interest in finite-state morphology should turn to Beesley and Karttunen (2003). Readers with further interest in computational models of Arabic and Semitic morphology should see Smrž (1998), Kiraz (2001), Habash et al. (2005).

A number of practical implementations of sentence segmentation were available by the 1990s. Summaries of sentence segmentation history and various algorithms can be found in Palmer (2000), Grefenstette (1999), and Mikheev (2003). Word segmentation has been studied especially in Japanese and Chinese. While the max-match algorithm we describe is very commonly used as a baseline, or when a simple but accurate algorithm is required, more recent algorithms rely on stochastic and machine learning algorithms; see for example such algorithms as Sproat et al. (1996), Xue and Shen (2003), and Tseng et al. (2005).

Gusfield (1997) is an excellent book covering everything you could want to know about string distance, minimum edit distance, and related areas.

Students interested in further details of the fundamental mathematics of automata theory should see Hopcroft and Ullman (1979) or Lewis and Papadimitriou (1988). Roche and Schabes (1997) is the definitive mathematical introduction to finite-state transducers for language applications, and together with Mohri (1997) and Mohri (2000) give many useful algorithms such as those for transducer minimization and determinization.

The CELEX dictionary is an extremely useful database for morphological analysis, containing full morphological parses of a large lexicon of English, German, and Dutch (Baayen et al., 1995).

Roark and Sproat (2007) is a general introduction to computational issues in morphology and syntax. Sproat (1993) is an older general introduction to computational morphology.

Exercises

3.1 Give examples of each of the noun and verb classes in Fig. 3.6, and find some exceptions to the rules.

3.2 Extend the transducer in Fig. 3.17 to deal with `sh` and `ch`.

3.3 Write a transducer(s) for the K insertion spelling rule in English.

3.4 Write a transducer(s) for the consonant doubling spelling rule in English.

3.5 The Soundex algorithm (Odell and Russell, 1922; Knuth, 1973) is a method commonly used in libraries and older Census records for representing people's names. It has the advantage that versions of the names that are slightly misspelled or otherwise modified (common, for example, in hand-written census records) will still have the same representation as correctly-spelled names. (e.g., Jurafsky, Jarofsky, Jarovsky, and Jarovski all map to J612).

a. Keep the first letter of the name, and drop all occurrences of non-initial a, e, h, i, o, u, w, y

b. Replace the remaining letters with the following numbers:

b, f, p, v → 1
c, g, j, k, q, s, x, z → 2
d, t → 3
l → 4
m, n → 5
r → 6

c. Replace any sequences of identical numbers , only if they derive from two or more letters that were *adjacent* in the original name, with a single number (i.e., 666 → 6).

d. Convert to the form `Letter Digit Digit Digit` by dropping digits past the third (if necessary) or padding with trailing zeros (if necessary).

The exercise: write a FST to implement the Soundex algorithm.

3.6 Implement one of the steps of the Porter Stemmer as a transducer.

3.7 Write the algorithm for parsing a finite-state transducer, using the pseudo-code introduced in Chapter 2. You should do this by modifying the algorithm ND-RECOGNIZE in Fig. **??** in Chapter 2.

3.8 Write a program that takes a word and, using an on-line dictionary, computes possible anagrams of the word, each of which is a legal word.

3.9 In Fig. 3.17, why is there a *z, s, x* arc from q_5 to q_1?

3.10 Computing minimum edit distances by hand, figure out whether *drive* is closer to *brief* or to *divers*, and what the edit distance is. You may use any version of *distance* that you like.

3.11 Now implement a minimum edit distance algorithm and use your hand-computed results to check your code.

3.12 Augment the minimum edit distance algorithm to output an alignment; you will need to store pointers and add a stage to compute the backtrace.

Antworth, E. L. (1990). *PC-KIMMO: A Two-level Processor for Morphological Analysis*. Summer Institute of Linguistics, Dallas, TX.

Baayen, R. H., Piepenbrock, R., and Gulikers, L. (1995). *The CELEX Lexical Database (Release 2) [CD-ROM]*. Linguistic Data Consortium, University of Pennsylvania [Distributor], Philadelphia, PA.

Baayen, R. H., Lieber, R., and Schreuder, R. (1997). The morphological complexity of simplex nouns. *Linguistics*, *35*(5), 861–877.

Barton, Jr., G. E., Berwick, R. C., and Ristad, E. S. (1987). *Computational Complexity and Natural Language*. MIT Press.

Bauer, L. (1983). *English word-formation*. Cambridge University Press.

Beesley, K. R. (1996). Arabic finite-state morphological analysis and generation. In *COLING-96*, Copenhagen, pp. 89–94.

Beesley, K. R. and Karttunen, L. (2003). *Finite-State Morphology*. CSLI Publications, Stanford University.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Chomsky, N. and Halle, M. (1968). *The Sound Pattern of English*. Harper and Row.

Damerau, F. J. and Mays, E. (1989). An examination of undetected typing errors. *Information Processing and Management*, *25*(6), 659–664.

De Jong, N. H., Feldman, L. B., Schreuder, R., Pastizzo, M., and Baayen, R. H. (2002). The processing and representation of Dutch and English compounds: Peripheral morphological, and central orthographic effects. *Brain and Language*, *81*, 555–567.

Fromkin, V. and Ratner, N. B. (1998). Speech production. In Gleason, J. B. and Ratner, N. B. (Eds.), *Psycholinguistics*. Harcourt Brace, Fort Worth, TX.

Garrett, M. F. (1975). The analysis of sentence production. In Bower, G. H. (Ed.), *The Psychology of Learning and Motivation*, Vol. 9. Academic.

Grefenstette, G. (1999). Tokenization. In van Halteren, H. (Ed.), *Syntactic Wordclass Tagging*. Kluwer.

Gusfield, D. (1997). *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press.

Habash, N., Rambow, O., and Kiraz, G. A. (2005). Morphological analysis and generation for arabic dialects. In *ACL Workshop on Computational Approaches to Semitic Languages*, pp. 17–24.

Hankamer, J. (1986). Finite state morphology and left to right phonology. In *Proceedings of the Fifth West Coast Conference on Formal Linguistics*, pp. 29–34.

Hankamer, J. and Black, H. A. (1991). Current approaches to computational morphology. Unpublished manuscript.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

Huffman, D. A. (1954). The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, *3*, 161–191. Continued in Volume 4.

Johnson, C. D. (1972). *Formal Aspects of Phonological Description*. Mouton, The Hague. Monographs on Linguistic Analysis No. 3.

Kaplan, R. M. and Kay, M. (1981). Phonological rules and finite-state transducers. Paper presented at the Annual meeting of the Linguistics Society of America. New York.

Kaplan, R. M. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, *20*(3), 331–378.

Karttunen, L., Chanod, J., Grefenstette, G., and Schiller, A. (1996). Regular expressions for language engineering. *Natural Language Engineering*, *2*(4), 305–238.

Karttunen, L. (1983). KIMMO: A general morphological processor. In *Texas Linguistics Forum 22*, pp. 165–186.

Kiraz, G. A. (2001). *Computational Nonlinear Morphology with Emphasis on Semitic Languages*. Cambridge University Press.

Knuth, D. E. (1973). *Sorting and Searching: The Art of Computer Programming Volume 3*. Addison-Wesley, Reading, MA.

Koskenniemi, K. (1983). Two-level morphology: A general computational model of word-form recognition and production. Tech. rep. Publication No. 11, Department of General Linguistics, University of Helsinki.

Koskenniemi, K. and Church, K. W. (1988). Complexity, two-level morphology, and Finnish. In *COLING-88*, Budapest, pp. 335–339.

Krovetz, R. (1993). Viewing morphology as an inference process. In *SIGIR-93*, pp. 191–202. ACM.

Kruskal, J. B. (1983). An overview of sequence comparison. In Sankoff, D. and Kruskal, J. B. (Eds.), *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pp. 1–44. Addison-Wesley, Reading, MA.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, *24*(4), 377–439.

Lerner, A. J. (1978). *The Street Where I Live*. Da Capo Press, New York.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, *10*(8), 707–710. Original in *Doklady Akademii Nauk SSSR* 163(4): 845–848 (1965).

Lewis, H. and Papadimitriou, C. (1988). *Elements of the Theory of Computation*. Prentice-Hall. Second edition.

Marslen-Wilson, W., Tyler, L. K., Waksler, R., and Older, L. (1994). Morphology and meaning in the English mental lexicon. *Psychological Review*, *101*(1), 3–33.

McCawley, J. D. (1978). Where you can shove infixes. In Bell, A. and Hooper, J. B. (Eds.), *Syllables and Segments*, pp. 213–221. North-Holland, Amsterdam.

Mealy, G. H. (1955). A method for synthesizing sequential circuits. *Bell System Technical Journal*, *34*(5), 1045–1079.

Mikheev, A. (2003). Text segmentation. In Mitkov, R. (Ed.), *Oxford Handbook of Computational Linguistics*. Oxford University Press, Oxford.

Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, *2*(1), 61–80.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, *23*(2), 269–312.

Mohri, M. (2000). Minimization algorithms for sequential transducers. *Theoretical Computer Science*, *234*, 177–201.

Moore, E. F. (1956). Gedanken-experiments on sequential machines. In Shannon, C. and McCarthy, J. (Eds.), *Automata Studies*, pp. 129–153. Princeton University Press, Princeton, NJ.

Moscoso del Prado Martín, F., Bertram, R., Häikiö, T., Schreuder, R., and Baayen, R. H. (2004). Morphological family size in a morphologically rich language: The case of finnish compared to dutch and hebrew. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *30*, 1271–1278.

NIST (2005). Speech recognition scoring toolkit (sctk) version 2.1. Available at http://www.nist.gov/speech/tools/.

Odell, M. K. and Russell, R. C. (1918/1922). U.S. Patents 1261167 (1918), 1435663 (1922)†. Cited in Knuth (1973).

Oflazer, K. (1993). Two-level description of Turkish morphology. In *Proceedings, Sixth Conference of the European Chapter of the ACL*.

Packard, D. W. (1973). Computer-assisted morphological analysis of ancient Greek. In Zampolli, A. and Calzolari, N. (Eds.), *Computational and Mathematical Linguistics: Proceedings of the International Conference on Computational Linguistics*, Pisa, pp. 343–355. Leo S. Olschki.

Palmer, D. D. (2000). Tokenisation and sentence segmentation. In Dale, R., Somers, H. L., and Moisl, H. (Eds.), *Handbook of Natural Language Processing*. Marcel Dekker.

Peterson, J. L. (1986). A note on undetected typing errors. *Communications of the ACM*, *29*(7), 633–637.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, *14*(3), 130–127.

Quirk, R., Greenbaum, S., Leech, G., and Svartvik, J. (1985). *A Comprehensive Grammar of the English Language*. Longman, London.

Roark, B. and Sproat, R. (2007). *Computational Approaches to Morphology and Syntax*. Oxford University Press.

Roche, E. and Schabes, Y. (1997). Introduction. In Roche, E. and Schabes, Y. (Eds.), *Finite-State Language Processing*, pp. 1–65. MIT Press.

Salomaa, A. (1973). *Formal Languages*. Academic.

Schützenberger, M. P. (1977). Sur une variante des fonctions sequentielles. *Theoretical Computer Science*, *4*, 47–57.

Seuss, D. (1960). *One Fish Two Fish Red Fish Blue Fish*. Random House, New York.

Shannon, C. E. (1938). A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers*, *57*, 713–723.

Smrž, O. (1998). *Functional Arabic Morphology*. Ph.D. thesis, Charles University in Prague.

Sproat, R. (1993). *Morphology and Computation*. MIT Press.

Sproat, R., Shih, C., Gale, W. A., and Chang, N. (1996). A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, *22*(3), 377–404.

Stanners, R. F., Neiser, J., Hernon, W. P., and Hall, R. (1979). Memory representation for morphologically related words. *Journal of Verbal Learning and Verbal Behavior*, *18*, 399–412.

Tseng, H., Chang, P., Andrew, G., Jurafsky, D., and Manning, C. D. (2005). Conditional random field word segmenter. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*.

Veblen, T. (1899). *Theory of the Leisure Class*. Macmillan Company, New York.

Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery*, *21*, 168–173.

Weber, D. J., Black, H. A., and McConnel, S. R. (1988). AMPLE: A tool for exploring morphology. Tech. rep. Occasional Publications in Academic Computing No. 12, Summer Institute of Linguistics, Dallas.

Weber, D. J. and Mann, W. C. (1981). Prospects for computer-assisted dialect adaptation. *American Journal of Computational Linguistics*, *7*, 165–177. Abridged from Summer Instititute of Linguistics *Notes on Linguistics* Special Publication 1, 1979.

Xue, N. and Shen, L. (2003). Chinese word segmentation as lmr tagging. In *Proceedings of the 2nd SIGHAN Workshop on Chinese Language Processing*, Sapporo, Japan.

4 N-GRAMS

But it must be recognized that the notion "probability of a sentence" is an entirely useless one, under any known interpretation of this term.

Noam Chomsky (1969, p. 57)

Anytime a linguist leaves the group the recognition rate goes up.

Fred Jelinek (then of the IBM speech group) (1988)[1]

Being able to predict the future is not always a good thing. Cassandra of Troy had the gift of fore-seeing, but was cursed by Apollo that her predictions would never be believed. Her warnings of the destruction of Troy were ignored and to simplify, let's just say that things just didn't go well for her later.

Predicting words seems somewhat less fraught, and in this chapter we take up this idea of word prediction. What word, for example, is likely to follow:

Please turn your homework . . .

WORD PREDICTION
N-GRAM MODELS
LANGUAGE MODELS
LMS

Hopefully most of you concluded that a very likely word is *in*, or possibly *over*, but probably not *the*. We formalize this idea of **word prediction** with probabilistic models called N**-gram models**, which predict the next word from the previous $N-1$ words. Such statistical models of word sequences are also called **language models** or **LMs**. Computing the probability of the next word will turn out to be closely related to computing the probability of a sequence of words. The following sequence, for example, has a non-zero probability of appearing in a text:

. . . all of a sudden I notice three guys standing on the sidewalk...

while this same set of words in a different order has a very low probability:

on guys all I of notice sidewalk three a sudden standing the

[1] This wording from his address is as recalled by Jelinek himself; the quote didn't appear in the proceedings (Palmer and Finin, 1990). Some remember a more snappy version: *Every time I fire a linguist the performance of the recognizer improves*.

As we will see, estimators like N-grams that assign a conditional probability to possible next words can be used to assign a joint probability to an entire sentence. Whether estimating probabilities of next words or of whole sequences, the N-gram model is one of the most important tools in speech and language processing.

N-grams are essential in any task in which we have to identify words in noisy, ambiguous input. In **speech recognition**, for example, the input speech sounds are very confusable and many words sound extremely similar. Russell and Norvig (2002) give an intuition from **handwriting recognition** for how probabilities of word sequences can help. In the movie *Take the Money and Run*, Woody Allen tries to rob a bank with a sloppily written hold-up note that the teller incorrectly reads as “I have a gub”. Any speech and language processing system could avoid making this mistake by using the knowledge that the sequence “I have a gun” is far more probable than the non-word “I have a gub” or even “I have a gull”.

N-gram models are also essential in statistical **machine translation**. Suppose we are translating a Chinese source sentence 他向记者介绍了该声明的主要内容 and as part of the process we have a set of potential rough English translations:

he briefed to reporters on the chief contents of the statement
he briefed reporters on the chief contents of the statement
he briefed to reporters on the main contents of the statement
he briefed reporters on the main contents of the statement

An N-gram grammar might tell us that, even after controlling for length, *briefed reporters* is more likely than *briefed to reporters*, and *main contents* is more likely than **chief contents**. This lets us select the bold-faced sentence above as the most fluent translation sentence, i.e. the one that has the highest probability.

In **spelling correction**, we need to find and correct spelling errors like the following (from Kukich (1992)) that accidentally result in real English words:

They are leaving in about fifteen *minuets* to go to her house.
The design *an* construction of the system will take more than a year.

Since these errors have real words, we can’t find them by just flagging words that are not in the dictionary. But note that *in about fifteen minuets* is a much less probable sequence than *in about fifteen minutes*. A spellchecker can use a probability estimator both to detect these errors and to suggest higher-probability corrections.

AUGMENTATIVE COMMUNICATION

Word prediction is also important for **augmentative communication** (Newell et al., 1998) systems that help the disabled. People who are unable to use speech or sign-language to communicate, like the physicist Steven Hawking, can communicate by using simple body movements to select words from a menu that are spoken by the system. Word prediction can be used to suggest likely words for the menu.

Besides these sample areas, N-grams are also crucial in NLP tasks like **part-of-speech tagging**, **natural language generation**, and **word similarity**, as well as in applications from **authorship identification** and **sentiment extraction** to **predictive text input** systems for cell phones.

4.1 Counting Words in Corpora

[upon being asked if there weren't enough words in the English language for him]:
"Yes, there are enough, but they aren't the right ones."
James Joyce, reported in Bates (1997)

CORPUS
CORPORA

Probabilities are based on counting things. Before we talk about probabilities, we need to decide what we are going to count. Counting of things in natural language is based on a **corpus** (plural **corpora**), an on-line collection of text or speech. Let's look at two popular corpora, Brown and Switchboard. The Brown corpus is a 1 million word collection of samples from 500 written texts from different genres (newspaper, novels, non-fiction, academic, etc.), assembled at Brown University in 1963-64 (Kučera and Francis, 1967; Francis, 1979; Francis and Kučera, 1982). How many words are in the following Brown sentence?

(4.1) He stepped out into the hall, was delighted to encounter a water brother.

Example (4.1) has 13 words if we don't count punctuation marks as words, 15 if we count punctuation. Whether we treat period ("."), comma (","), and so on as words depends on the task. Punctuation is critical for finding boundaries of things (commas, periods, colons), and for identifying some aspects of meaning (question marks, exclamation marks, quotation marks). For some tasks, like part-of-speech tagging or parsing or speech synthesis, we sometimes treat punctuation marks as if they were separate words.

UTTERANCE

The Switchboard corpus of telephone conversations between strangers was collected in the early 1990s and contains 2430 conversations averaging 6 minutes each, totaling 240 hours of speech and about 3 million words (Godfrey et al., 1992). Such corpora of spoken language don't have punctuation, but do introduce other complications with regard to defining words. Let's look at one utterance from Switchboard; an **utterance** is the spoken correlate of a sentence:

(4.2) I do uh main- mainly business data processing

DISFLUENCIES
FRAGMENT
FILLERS
FILLED PAUSES

This utterance has two kinds of **disfluencies**. The broken-off word *main-* is called a **fragment**. Words like *uh* and *um* are called **fillers** or **filled pauses**. Should we consider these to be words? Again, it depends on the application. If we are building an automatic dictation system based on automatic speech recognition, we might want to eventually strip out the disfluencies.

But we also sometimes keep disfluencies around. How disfluent a person is can be used to identify them, or to detect whether they are stressed or confused. Disfluencies also often occur with particular syntactic structures, so they may help in parsing and word prediction. Stolcke and Shriberg (1996) found for example that treating *uh* as a word improves next-word prediction (why might this be?), and so most speech recognition systems treat *uh* and *um* as words.[2]

Are capitalized tokens like *They* and uncapitalized tokens like *they* the same word? These are lumped together in speech recognition, while for part-of-speech-tagging cap-

[2] Clark and Fox Tree (2002) showed that *uh* and *um* have different meanings. What do you think they are?

italization is retained as a separate feature. For the rest of this chapter we will assume our models are not case-sensitive.

WORDFORM

How about inflected forms like *cats* versus *cat*? These two words have the same **lemma** *cat* but are different wordforms. Recall from Ch. 3 that a lemma is a set of lexical forms having the same stem, the same major part-of-speech, and the same word-sense. The **wordform** is the full inflected or derived form of the word. For morphologically complex languages like Arabic we often need to deal with lemmatization. *N*-grams for speech recognition in English, however, and all the examples in this chapter, are based on wordforms.

As we can see, *N*-gram models, and counting words in general, requires that we do the kind of tokenization or text normalization that we introduced in the previous chapter: separating out punctuation, dealing with abbreviations like *m.p.h.*, normalizing spelling, and so on.

TYPES

TOKENS

How many words are there in English? To answer this question we need to distinguish **types**, the number of distinct words in a corpus or vocabulary size V, from **tokens**, the total number N of running words. The following Brown sentence has 16 tokens and 14 types (not counting punctuation):

(4.3) They picnicked by the pool, then lay back on the grass and looked at the stars.

The Switchboard corpus has about 20,000 wordform types (from about 3 million wordform tokens) Shakespeare's complete works have 29,066 wordform types (from 884,647 wordform tokens) (Kučera, 1992) The Brown corpus has 61,805 wordform types from 37,851 lemma types (from 1 million wordform tokens). Looking at a very large corpus of 583 million wordform tokens, Brown et al. (1992a) found that it included 293,181 different wordform types. Dictionaries can help in giving lemma counts; dictionary entries, or **boldface forms** are a very rough upper bound on the number of lemmas (since some lemmas have multiple boldface forms). The American Heritage Dictionary lists 200,000 boldface forms. It seems like the larger corpora we look at, the more word types we find. In general (Gale and Church, 1990) suggest that the vocabulary size (the number of types) grows with at least the square root of the number of tokens (i.e. $V > O(\sqrt{N})$.

In the rest of this chapter we will continue to distinguish between types and tokens, using "types" to mean wordform types.

4.2 Simple (Unsmoothed) *N*-grams

Let's start with some intuitive motivations for *N*-grams. We assume that the reader has acquired some very basic background in probability theory. Our goal is to compute the probability of a word w given some history h, or $P(w|h)$. Suppose the history h is "*its water is so transparent that*" and we want to know the probability that the next word is *the*:

$$P(the|its\ water\ is\ so\ transparent\ that). \tag{4.4}$$

How can we compute this probability? One way is to estimate it from relative frequency counts. For example, we could take a very large corpus, count the number of times we see *the water is so transparent that*, and count the number of times this is followed by *the*. This would be answering the question "Out of the times we saw the history h, how many times was it followed by the word w", as follows:

$$\begin{gathered} P(the|its\ water\ is\ so\ transparent\ that) = \\ \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)} \end{gathered} \tag{4.5}$$

With a large enough corpus, such as the web, we can compute these counts, and estimate the probability from Equation (4.5). You should pause now, go to the web and compute this estimate for yourself.

While this method of estimating probabilities directly from counts works fine in many cases, it turns out that even the web isn't big enough to give us good estimates in most cases. This is because language is creative; new sentences are created all the time, and we won't always be able to count entire sentences. Even simple extensions of the example sentence may have counts of zero on the web (such as "*Walden Pond's water is so transparent that the*").

Similarly, if we wanted to know the joint probability of an entire sequence of words like *its water is so transparent*, we could do it by asking "out of all possible sequences of 5 words, how many of them are *its water is so transparent*?" We would have to get the count of *its water is so transparent*, and divide by the sum of the counts of all possible 5 word sequences. That seems rather a lot to estimate!

For this reason, we'll need to introduce cleverer ways of estimating the probability of a word w given a history h, or the probability of an entire word sequence W. Let's start with a little formalizing of notation. In order to represent the probability of a particular random variable X_i taking on the value "the", or $P(X_i = \text{"the"})$, we will use the simplification $P(the)$. We'll represent a sequence of N words either as $w_1 \ldots w_n$ or w_1^n. For the joint probability of each word in a sequence having a particular value $P(X = w_1, Y = w_2, Z = w_3, ...,)$ we'll use $P(w_1, w_2, ..., w_n)$.

Now how can we compute probabilities of entire sequences like $P(w_1, w_2, ..., w_n)$? One thing we can do is to decompose this probability using the **chain rule of probability**:

$$\begin{aligned} P(X_1 ... X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1^2)\ldots P(X_n|X_1^{n-1}) \\ &= \prod_{k=1}^{n} P(X_k|X_1^{k-1}) \end{aligned} \tag{4.6}$$

Applying the chain rule to words, we get:

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \end{aligned} \tag{4.7}$$

The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words. Equation

(4.7) suggests that we could estimate the joint probability of an entire sequence of words by multiplying together a number of conditional probabilities. But using the chain rule doesn't really seem to help us! We don't know any way to compute the exact probability of a word given a long sequence of preceding words, $P(w_n|w_1^{n-1})$. As we said above, we can't just estimate by counting the number of times every word occurs following every long string, because language is creative and any particular context might have never occurred before!

The intuition of the N-gram model is that instead of computing the probability of a word given its entire history, we will **approximate** the history by just the last few words.

BIGRAM

The **bigram** model, for example, approximates the probability of a word given all the previous words $P(w_n|w_1^{n-1})$ by using only the conditional probability of the preceding word $P(w_n|w_{n-1})$. In other words, instead of computing the probability

$$P(\text{the}|\text{Walden Pond's water is so transparent that}) \tag{4.8}$$

we approximate it with the probability

$$P(\text{the}|\text{that}) \tag{4.9}$$

When we use a bigram model to predict the conditional probability of the next word we are thus making the following approximation:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1}) \tag{4.10}$$

MARKOV

N-GRAM

This assumption that the probability of a word depends only on the previous word is called a **Markov** assumption. Markov models are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far into the past. We can generalize the bigram (which looks one word into the past) to the trigram (which looks two words into the past) and thus to the **N-gram** (which looks $N-1$ words into the past).

Thus the general equation for this N-gram approximation to the conditional probability of the next word in a sequence is:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}) \tag{4.11}$$

Given the bigram assumption for the probability of an individual word, we can compute the probability of a complete word sequence by substituting Equation (4.10) into Equation (4.7):

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}) \tag{4.12}$$

MAXIMUM LIKELIHOOD ESTIMATION

MLE

NORMALIZING

How do we estimate these bigram or N-gram probabilities? The simplest and most intuitive way to estimate probabilities is called **Maximum Likelihood Estimation**, or **MLE**. We get the MLE estimate for the parameters of an N-gram model by taking counts from a corpus, and **normalizing** them so they lie between 0 and 1.[3]

For example, to compute a particular bigram probability of a word y given a previous word x, we'll compute the count of the bigram $C(xy)$ and normalize by the sum of all the bigrams that share the same first word x:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \tag{4.13}$$

We can simplify this equation, since the sum of all bigram counts that start with a given word w_{n-1} must be equal to the unigram count for that word w_{n-1}. (The reader should take a moment to be convinced of this):

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{4.14}$$

Let's work through an example using a mini-corpus of three sentences. We'll first need to augment each sentence with a special symbol <s> at the beginning of the sentence, to give us the bigram context of the first word. We'll also need a special end-symbol `</s>`.[4]

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

Here are the calculations for some of the bigram probabilities from this corpus

$$P(\texttt{I|<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam|<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am|I}) = \tfrac{2}{3} = .67$$

$$P(\texttt{</s>|Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam|am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do|I}) = \tfrac{1}{3} = .33$$

For the general case of MLE N-gram parameter estimation:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})} \tag{4.15}$$

RELATIVE FREQUENCY

Equation 4.15 (like equation 4.14) estimates the N-gram probability by dividing the observed frequency of a particular sequence by the observed frequency of a prefix. This ratio is called a **relative frequency**. We said above that this use of relative frequencies as a way to estimate probabilities is an example of Maximum Likelihood Estimation or MLE. In Maximum Likelihood Estimation, the resulting parameter set maximizes the likelihood of the training set T given the model M (i.e., $P(T|M)$). For example, suppose the word *Chinese* occurs 400 times in a corpus of a million words like the Brown corpus. What is the probability that a random word selected from some other text of say a million words will be the word *Chinese*? The MLE estimate of its probability is $\frac{400}{1000000}$ or .0004. Now .0004 is not the best possible estimate of the probability of *Chinese* occurring in all situations; it might turn out that in some OTHER corpus or context *Chinese* is a very unlikely word. But it is the probability that makes it *most*

[3] For probabilistic models, normalizing means dividing by some total count so that the resulting probabilities fall legally between 0 and 1.

[4] As Chen and Goodman (1998) point out, we need the end-symbol to make the bigram grammar a true probability distribution. Without an end-symbol, the sentence probabilities for all sentences of a given length would sum to one, and the probability of the whole language would be infinite.

likely that Chinese will occur 400 times in a million-word corpus. We will see ways to modify the MLE estimates slightly to get better probability estimates in Sec. 4.5.

Let's move on to some examples from a slightly larger corpus than our 14-word example above. We'll use data from the now-defunct Berkeley Restaurant Project, a dialogue system from the last century that answered questions about a database of restaurants in Berkeley, California (Jurafsky et al., 1994). Here are some sample user queries, lowercased and with no punctuation (a representative corpus of 9332 sentences is on the website):

> can you tell me about any good cantonese restaurants close by
> mid priced thai food is what i'm looking for
> tell me about chez panisse
> can you give me a listing of the kinds of food that are available
> i'm looking for a good place to eat breakfast
> when is caffe venezia open during the day

Fig. 4.1 shows the bigram counts from a piece of a bigram grammar from the Berkeley Restaurant Project. Note that the majority of the values are zero. In fact, we have chosen the sample words to cohere with each other; a matrix selected from a random set of seven words would be even more sparse.

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 4.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

Fig. 4.2 shows the bigram probabilities after normalization (dividing each row by the following unigram counts):

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Here are a few other useful probabilities:

$$P(\texttt{i|<s>}) = 0.25 \qquad P(\texttt{english|want}) = 0.0011$$
$$P(\texttt{food|english}) = 0.5 \qquad P(\texttt{</s>|food}) = 0.68$$

Now we can compute the probability of sentences like *I want English food* or *I want Chinese food* by simply multiplying the appropriate bigram probabilities together, as follows:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 4.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences.

$$
\begin{aligned}
& P(\texttt{<s> i want english food </s>}) \\
& \quad = P(\texttt{i|<s>})P(\texttt{want|I})P(\texttt{english|want}) \\
& \qquad\qquad P(\texttt{food|english})P(\texttt{</s>|food}) \\
& \quad = .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\
& \quad = \; = .000031
\end{aligned}
$$

We leave it as an exercise for the reader to compute the probability of *i want chinese food*. But that exercise does suggest that we'll want to think a bit about what kinds of linguistic phenomena are captured in bigrams. Some of the bigram probabilities above encode some facts that we think of as strictly syntactic in nature, like the fact that what comes after *eat* is usually a noun or an adjective, or that what comes after *to* is usually a verb. Others might be more cultural than linguistic, like the low probability of anyone asking for advice on finding English food.

TRIGRAM

Although we will generally show bigram models in this chapter for pedagogical purposes, note that when there is sufficient training data we are more likely to use **trigram** models, which condition on the previous two words rather than the previous word. To compute trigram probabilities at the very beginning of sentence, we can use two pseudo-words for the first trigram (i.e., $P(\texttt{I|<s><s>})$).

4.3 Training and Test Sets

The N-gram model is a good example of the kind of statistical models that we will be seeing throughout speech and language processing. The probabilities of an N-gram model come from the corpus it is trained on. In general, the parameters of a statistical model are trained on some set of data, and then we apply the models to some new data in some task (such as speech recognition) and see how well they work. Of course this new data or task won't be the exact same data we trained on.

TRAINING SET
TEST SET

We can formalize this idea of training on some data, and testing on some other data by talking about these two data sets as a **training set** and a **test set** (or a **training corpus** and a **test corpus**). Thus when using a statistical model of language given some corpus of relevant data, we start by dividing the data into training and test sets.

We train the statistical parameters of the model on the training set, and then use this trained model to compute probabilities on the test set.

EVALUATE

This training-and-testing paradigm can also be used to **evaluate** different N-gram architectures. Suppose we want to compare different language models (such as those based on N-grams of different orders N, or using the different **smoothing** algorithms to be introduced in Sec. 4.5). We can do this by taking a corpus and dividing it into a training set and a test set. Then we train the two different N-gram models on the training set and see which one better models the test set. But what does it mean to "model the test set"? There is is a useful metric for how well a given statistical model matches a test corpus, called **perplexity**, introduced on page 13. Perplexity is based on computing the probability of each sentence in the test set; intuitively, whichever model assigns a higher probability to the test set (hence more accurately predicts the test set) is a better model.

Since our evaluation metric is based on test set probability, it's important not to let the test sentences into the training set. Suppose we are trying to compute the probability of a particular "test" sentence. If our test sentence is part of the training corpus, we will mistakenly assign it an artificially high probability when it occurs in the test set. We call this situation **training on the test set**. Training on the test set introduces a bias that makes the probabilities all look too high and causes huge inaccuracies in perplexity.

HELD-OUT

In addition to training and test sets, other divisions of data are often useful. Sometimes we need an extra source of data to augment the training set. Such extra data is called a **held-out** set, because we hold it out from our training set when we train our N-gram counts. The held-out corpus is then used to set some other parameters; for example we will see the use of held-out data to set interpolation weights in **interpolated** N-gram models in Sec. 4.6. Finally, sometimes we need to have multiple test sets. This happens because we might use a particular test set so often that we implicitly tune to its characteristics. Then we would definitely need a fresh test set which is truly unseen. In such cases, we call the initial test set the **development** test set or, **devset**. We will discuss development test sets again in Ch. 5.

DEVELOPMENT

How do we divide our data into training, dev, and test sets? There is a tradeoff, since we want our test set to be as large as possible and a small test set may be accidentally unrepresentative. On the other hand, we want as much training data as possible. At the minimum, we would want to pick the smallest test set that gives us enough statistical power to measure a statistically significant difference between two potential models. In practice, we often just divide our data into 80% training, 10% development, and 10% test. Given a large corpus that we want to divide into training and test, test data can either be taken from some continuous sequence of text inside the corpus, or we can remove smaller "stripes" of text from randomly selected parts of our corpus and combine them into a test set.

4.3.1 N-gram Sensitivity to the Training Corpus

The N-gram model, like many statistical models, is very dependent on the training corpus. One implication of this is that the probabilities often encode very specific facts about a given training corpus. Another implication is that N-grams do a better and better job of modeling the training corpus as we increase the value of N.

We can visualize both of these facts by borrowing the technique of Shannon (1951) and Miller and Selfridge (1950), of generating random sentences from different *N*-gram models. It's simplest to visualize how this works for the unigram case. Imagine all the words of English covering the probability space between 0 and 1, each word covering an interval equal to its frequency. We choose a random value between 0 and 1, and print out the word whose interval includes the real value we have chosen. We continue choosing random numbers and generating words until we randomly generate the sentence-final token `</s>`. The same technique can be used to generate bigrams by first generating a random bigram that starts with `<s>` (according to its bigram probability), then choosing a random bigram to follow it (again, according to its conditional probability), and so on.

To give an intuition for the increasing power of higher-order *N*-grams, Fig. 4.3 shows random sentences generated from unigram, bigram, trigram, and quadrigram models trained on Shakespeare's works.

Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

Figure 4.3 Sentences randomly generated from four *N*-gram models computed from Shakespeare's works. All characters were mapped to lower case and punctuation marks were treated as words. Output was hand-corrected for capitalization to improve readability.

The longer the context on which we train the model, the more coherent the sentences. In the unigram sentences, there is no coherent relation between words, nor any sentence-final punctuation. The bigram sentences have some very local word-to-word coherence (especially if we consider that punctuation counts as a word). The trigram and quadrigram sentences are beginning to look a lot like Shakespeare. Indeed a careful investigation of the quadrigram sentences shows that they look a little too much like Shakespeare. The words *It cannot be but so* are directly from *King John*. This is because, not to put the knock on Shakespeare, his oeuvre is not very large as corpora go ($N = 884,647, V = 29,066$), and our N-gram probability matrices are ridiculously sparse. There are $V^2 = 844,000,000$ possible bigrams alone, and the number of possible quadrigrams is $V^4 = 7 \times 10^{17}$. Thus once the generator has chosen the first quadrigram (*It cannot be but*), there are only five possible continuations (*that*, *I*, *he*, *thou*, and *so*); indeed for many quadrigrams there is only one continuation.

To get an idea of the dependence of a grammar on its training set, let's look at an N-gram grammar trained on a completely different corpus: *the Wall Street Journal* (WSJ) newspaper. Shakespeare and *the Wall Street Journal* are both English, so we might expect some overlap between our N-grams for the two genres. In order to check whether this is true, Fig. 4.4 shows sentences generated by unigram, bigram, and trigram grammars trained on 40 million words from WSJ.

unigram: Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

bigram: Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

trigram: They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Figure 4.4 Sentences randomly generated from three orders of N-gram computed from 40 million words of *the Wall Street Journal*. All characters were mapped to lower case and punctuation marks were treated as words. Output was hand corrected for capitalization to improve readability.

Compare these examples to the pseudo-Shakespeare in Fig. 4.3. While superficially they both seem to model "English-like sentences" there is obviously no overlap whatsoever in possible sentences, and little if any overlap even in small phrases. This stark difference tells us that statistical models are likely to be pretty useless as predictors if the training sets and the test sets are as different as Shakespeare and WSJ.

How should we deal with this problem when we build N-gram models? In general we need to be sure to use a training corpus that looks like our test corpus. We especially wouldn't choose training and tests from different **genres** of text like newspaper text, early English fiction, telephone conversations, and web pages. Sometimes finding appropriate training text for a specific new task can be difficult; to build N-grams for text prediction in SMS (Short Message Service), we need a training corpus of SMS data. To build N-grams on business meetings, we would need to have corpora of transcribed business meetings.

For general research where we know we want written English but don't have a domain in mind, we can use a balanced training corpus that includes cross sections from different genres, such as the 1-million-word Brown corpus of English (Francis and

Kučera, 1982) or the 100-million-word British National Corpus (Leech et al., 1994).

Recent research has also studied ways to dynamically **adapt** language models to different genres; see Sec. 4.9.4.

4.3.2 Unknown Words: Open versus closed vocabulary tasks

CLOSED VOCABULARY

Sometimes we have a language task in which we know all the words that can occur, and hence we know the vocabulary size V in advance. The **closed vocabulary** assumption is the assumption that we have such a lexicon, and that the test set can only contain words from this lexicon. The closed vocabulary task thus assumes there are no unknown words.

OOV

But of course this is a simplification; as we suggested earlier, the number of unseen words grows constantly, so we can't possibly know in advance exactly how many there are, and we'd like our model to do something reasonable with them. We call these unseen events **unknown** words, or **out of vocabulary** (**OOV**) words. The percentage of OOV words that appear in the test set is called the **OOV rate**.

OPEN VOCABULARY

An **open vocabulary** system is one where we model these potential unknown words in the test set by adding a pseudo-word called `<UNK>`. We can train the probabilities of the unknown word model `<UNK>` as follows:

1. **Choose a vocabulary** (word list) which is fixed in advance.
2. **Convert** in the training set any word that is not in this set (any OOV word) to the unknown word token `<UNK>` in a text normalization step.
3. **Estimate** the probabilities for `<UNK>` from its counts just like any other regular word in the training set.

4.4 Evaluating *N*-grams: Perplexity

EXTRINSIC EVALUATION

IN VIVO

The best way to evaluate the performance of a language model is to embed it in an application and measure the total performance of the application. Such end-to-end evaluation is called **extrinsic evaluation**, and also sometimes called **in vivo** evaluation (Sparck Jones and Galliers, 1996). Extrinisic evaluation is the only way to know if a particular improvement in a component is really going to help the task at hand. Thus for speech recognition, we can compare the performance of two language models by running the speech recognizer twice, once with each language model, and seeing which gives the more accurate transcription.

INTRINSITIC EVALUATION

Unfortunately, end-to-end evaluation is often very expensive; evaluating a large speech recognition test set, for example, takes hours or even days. Thus we would like a metric that can be used to quickly evaluate potential improvements in a language model. An **intrinsitic evaluation** metric is one which measures the quality of a model independent of any application. **Perplexity** is the most common intrinsic evaluation metric for N-gram language models. While an (intrinsic) improvement in perplexity does not guarantee an (extrinsic) improvement in speech recognition performance (or any other end-to-end metric), it often correlates with such improvements. Thus it is commonly used as a quick check on an algorithm and an improvement in perplexity can then be confirmed by an end-to-end evaluation.

The intuition of perplexity is that given two probabilistic models, the better model is the one that has a tighter fit to the test data, or predicts the details of the test data better. We can measure better prediction by looking at the probability the model assigns to the test data; the better model will assign a higher probability to the test data.

PERPLEXITY

More formally, the **perplexity** (sometimes called *PP* for short) of a language model on a test set is a function of the probability that the language model assigns to that test set. For a test set $W = w_1 w_2 \ldots w_N$, the perplexity is the probability of the test set, normalized by the number of words:

(4.16)
$$\begin{aligned} \mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N})} \end{aligned}$$

We can use the chain rule to expand the probability of W:

(4.17)
$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

Thus if we are computing the perplexity of W with a bigram language model, we get:

(4.18)
$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

Note that because of the inverse in Equation (4.17), the higher the conditional probability of the word sequence, the lower the perplexity. Thus minimizing perplexity is equivalent to maximizing the test set probability according to the language model. What we generally use for word sequence in Equation (4.17) or Equation (4.18) is the entire sequence of words in some test set. Since of course this sequence will cross many sentence boundaries, we need to include the begin- and end-sentence markers `<s>` and `</s>` in the probability computation. We also need to include the end-of-sentence marker `</s>` (but not the beginning-of-sentence marker `<s>`) in the total count of word tokens N.

There is another way to think about perplexity: as the **weighted average branching factor** of a language. The branching factor of a language is the number of possible next words that can follow any word. Consider the task of recognizing the digits in English (zero, one, two,..., nine), given that each of the 10 digits occur with equal probability $P = \frac{1}{10}$. The perplexity of this mini-language is in fact 10. To see that, imagine a string of digits of length N. By Equation (4.17), the perplexity will be:

$$\mathrm{PP}(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$
\begin{aligned}
&= (\frac{1}{10}^{N})^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10 \qquad (4.19)
\end{aligned}
$$

But now suppose that the number zero is really frequent and occurs 10 times more often than other numbers. Now we should expect the perplexity to be lower, since most of the time the next number will be zero. Thus although the branching factor is still 10, the perplexity or weighted branching factor is smaller. We leave this calculation as an exercise to the reader.

We'll see in Sec. 4.10 that perplexity is also closely related to the information-theoretic notion of entropy.

Finally, let's see an example of how perplexity can be used to compare three N-gram models. We trained unigram, bigram, and trigram grammars on 38 million words (including start-of-sentence tokens) from the Wall Street Journal, using a 19,979 word vocabulary.[5] We then computed the perplexity of each of these models on a test set of 1.5 million words via Equation (4.65). The table below shows the perplexity of a 1.5 million word WSJ test set according to each of these grammars.

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

As we see above, the more information the N-gram gives us about the word sequence, the lower the perplexity (since as Equation (4.17) showed, perplexity is related inversely to the likelihood of the test sequence according to the model).

CLOSED VOCABULARY

Note that in computing perplexities the N-gram model P must be constructed without any knowledge of the test set. Any kind of knowledge of the test set can cause the perplexity to be artificially low. For example, we defined above the **closed vocabulary** task, in which the vocabulary for the test set is specified in advance. This can greatly reduce the perplexity. As long as this knowledge is provided equally to each of the models we are comparing, the closed vocabulary perplexity can still be useful for comparing models, but care must be taken in interpreting the results. In general, the perplexity of two language models is only comparable if they use the same vocabulary.

4.5 SMOOTHING

Never do I ever want
to hear another word!
There isn't one,
I haven't heard!

[5] More specifically, Katz-style backoff grammars with Good-Turing discounting trained on 38 million words from the WSJ0 corpus (LDC, 1993), open-vocabulary, using the <UNK> token; see later sections for definitions.

Eliza Doolittle in Alan Jay Lerner's *My Fair Lady*

SPARSE DATA

There is a major problem with the maximum likelihood estimation process we have seen for training the parameters of an N-gram model. This is the problem of **sparse data** caused by the fact that our maximum likelihood estimate was based on a particular set of training data. For any N-gram that occurred a sufficient number of times, we might have a good estimate of its probability. But because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it. This missing data means that the N-gram matrix for any given training corpus is bound to have a very large number of cases of putative "zero probability N-grams" that should really have some non-zero probability. Furthermore, the MLE method also produces poor estimates when the counts are non-zero but still small.

We need a method which can help get better estimates for these zero or low-frequency counts. Zero counts turn out to cause another huge problem. The **perplexity** metric defined above requires that we compute the probability of each test sentence. But if a test sentence has an N-gram that never appeared in the training set, the Maximum Likelihood estimate of the probability for this N-gram, and hence for the whole test sentence, will be zero! This means that in order to evaluate our language models, we need to modify the MLE method to assign some non-zero probability to any N-gram, even one that was never observed in training.

SMOOTHING

For these reasons, we'll want to modify the maximum likelihood estimates for computing N-gram probabilities, focusing on the N-gram events that we incorrectly assumed had zero probability. We use the term **smoothing** for such modifications that address the poor estimates that are due to variability in small data sets. The name comes from the fact that (looking ahead a bit) we will be shaving a little bit of probability mass from the higher counts, and piling it instead on the zero counts, making the distribution a little less jagged.

In the next few sections we will introduce some smoothing algorithms and show how they modify the Berkeley Restaurant bigram probabilities in Fig. 4.2.

4.5.1 Laplace Smoothing

LAPLACE SMOOTHING

One simple way to do smoothing might be just to take our matrix of bigram counts, before we normalize them into probabilities, and add one to all the counts. This algorithm is called **Laplace smoothing**, or Laplace's Law (Lidstone, 1920; Johnson, 1932; Jeffreys, 1948). Laplace smoothing does not perform well enough to be used in modern N-gram models, but we begin with it because it introduces many of the concepts that we will see in other smoothing algorithms and also gives us a useful baseline.

Let's start with the application of Laplace smoothing to unigram probabilities. Recall that the unsmoothed maximum likelihood estimate of the unigram probability of the word w_i is its count c_i normalized by the total number of word tokens N:

$$P(w_i) = \frac{c_i}{N}$$

Laplace smoothing merely adds one to each count (hence its alternate name **add-**

ADD-ONE

one smoothing). Since there are V words in the vocabulary, and each one got incremented, we also need to adjust the the denominator to take into account the extra V observations.[6]

$$P_{\text{Laplace}}(w_i) = \frac{c_i+1}{N+V} \tag{4.20}$$

Instead of changing both the numerator and denominator it is convenient to describe how a smoothing algorithm affects the numerator, by defining an **adjusted count** c^*. This adjusted count is easier to compare directly with the MLE counts, and can be turned into a probability like an MLE count by normalizing by N. To define this count, since we are only changing the numerator, in addition to adding one we'll also need to multiply by a normalization factor $\frac{N}{N+V}$:

$$c_i^* = (c_i+1)\frac{N}{N+V} \tag{4.21}$$

We can now turn c^* into a probability p_i^* by normalizing by N.

DISCOUNTING

DISCOUNT

A related way to view smoothing is as **discounting** (lowering) some non-zero counts in order to get the probability mass that will be assigned to the zero counts. Thus instead of referring to the discounted counts c^*, we might describe a smoothing algorithm in terms of a relative **discount** d_c, the ratio of the discounted counts to the original counts:

$$d_c = \frac{c^*}{c}$$

Now that we have the intuition for the unigram case, let's smooth our Berkeley Restaurant Project bigrams. Fig. 4.5 shows the add-one smoothed counts for the bigrams in Fig. 4.1.

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Figure 4.5 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

Fig. 4.6 shows the add-one smoothed probabilities for the bigrams in Fig. 4.2. Recall that normal bigram probabilities are computed by normalizing each row of counts by the unigram count:

[6] What happens to our P values if we don't increase the denominator?

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{4.22}$$

For add-one smoothed bigram counts we need to augment the unigram count by the number of total word types in the vocabulary V:

$$P^*_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} \tag{4.23}$$

Thus each of the unigram counts given in the previous section will need to be augmented by $V = 1446$. The result is the smoothed bigram probabilities in Fig. 4.6.

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Figure 4.6 Add-one smoothed bigram probabilities for eight of the words (out of $V = 1446$) in the BeRP corpus of 9332 sentences.

It is often convenient to reconstruct the count matrix so we can see how much a smoothing algorithm has changed the original counts. These adjusted counts can be computed by Equation (4.24). Fig. 4.7 shows the reconstructed counts.

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V} \tag{4.24}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Figure 4.7 Add-one reconstituted counts for eight words (of $V = 1446$) in the BeRP corpus of 9332 sentences.

Note that add-one smoothing has made a very big change to the counts. $C(\textit{want to})$ changed from 608 to 238! We can see this in probability space as well: $P(\textit{to}|\textit{want})$

decreases from .66 in the unsmoothed case to .26 in the smoothed case. Looking at the discount d (the ratio between new and old counts) shows us how strikingly the counts for each prefix word have been reduced; the discount for the bigram *want to* is .39, while the discount for *Chinese food* is .10, a factor of 10!

The sharp change in counts and probabilities occurs because too much probability mass is moved to all the zeros. We could move a bit less mass by adding a fractional count rather than 1 (add-δ smoothing; (Lidstone, 1920; Johnson, 1932; Jeffreys, 1948)), but this method requires a method for choosing δ dynamically, results in an inappropriate discount for many counts, and turns out to give counts with poor variances. For these and other reasons (Gale and Church, 1994), we'll need better smoothing methods for N-grams like the ones we'll see in the next section.

4.5.2 Good-Turing Discounting

GOOD-TURING

There are a number of much better discounting algorithms that are only slightly more complex than add-one smoothing. In this section we introduce one of them, known as **Good-Turing** smoothing.

WITTEN-BELL DISCOUNTING
KNEYSER-NEY SMOOTHING

The intuition of a number of discounting algorithms (Good Turing, **Witten-Bell discounting**, and **Kneyser-Ney smoothing**) is to use the count of things you've seen *once* to help estimate the count of things you've *never seen*. The Good-Turing algorithm was first described by Good (1953), who credits Turing with the original idea. The basic insight of Good-Turing smoothing is to re-estimate the amount of probability mass to assign to N-grams with zero counts by looking at the number of N-grams that occurred one time. A word or N-gram (or any event) that occurs once is called a **singleton**, or a **hapax legomenon**. The Good-Turing intuition is to use the frequency of singletons as a re-estimate of the frequency of zero-count bigrams.

SINGLETON

Let's formalize the algorithm. The Good-Turing algorithm is based on computing N_c, the number of N-grams that occur c times. We refer to the number of N-grams that occur c times as the **frequency of frequency** c. So applying the idea to smoothing the joint probability of bigrams, N_0 is the number of bigrams with count 0, N_1 the number of bigrams with count 1 (singletons), and so on. We can think of each of the N_c as a bin which stores the number of different N-grams that occur in the training set with that frequency c. More formally:

$$N_c = \sum_{x:count(x)=c} 1 \tag{4.25}$$

The MLE count for N_c is c. The Good-Turing estimate replaces this with a smoothed count c^*, as a function of N_{c+1}:

$$c^* = (c+1)\frac{N_{c+1}}{N_c} \tag{4.26}$$

We can use (Equation (4.26)) to replace the MLE counts for all the bins N_1, N_2, and so on. Instead of using this equation directly to re-estimate the smoothed count c^* for N_0, we use the following equation for the probability P^*_{GT} for things that had zero count N_0, or what we might call the **missing mass**:

$$P^*_{GT}(\text{things with frequency zero in training}) = \frac{N_1}{N} \tag{4.27}$$

Here N_1 is the count of items in bin 1, i.e. that were seen once in training, and N is the total number of items we have seen in training. Equation (4.27) thus gives the probability that the $N+1$st bigram we see will be one that we never saw in training. Showing that (Equation (4.27)) follows from (Equation (4.26)) is left as Exercise 4.8 for the reader.

The Good-Turing method was first proposed for estimating the populations of animal species. Let's consider an illustrative example from this domain created by Joshua Goodman and Stanley Chen. Suppose we are fishing in a lake with 8 species (bass, carp, catfish, eel, perch, salmon, trout, whitefish) and we have seen 6 species with the following counts: 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, and 1 eel (so we haven't yet seen the catfish or bass). What is the probability that the next fish we catch will be a new species, i.e., one that had a zero frequency in our training set, i.e., in this case either a catfish or a bass?

The MLE count c of a hitherto-unseen species (bass or catfish) is 0. But Equation (4.27) tells us that the probability of a new fish being one of these unseen species is $\frac{3}{18}$, since N_1 is 3 and N is 18:

$$P^*_{GT}(\text{things with frequency zero in training}) = \frac{N_1}{N} = \frac{3}{18} \tag{4.28}$$

What is the probability that the next fish will be another trout? The MLE count for trout is 1, so the MLE estimated probability is $\frac{1}{18}$. But the Good-Turing estimate must be lower, since we just stole $\frac{3}{18}$ of our probability mass to use on unseen events! We'll need to discount the MLE probabilities for trout, perch, carp, etc. In summary, the revised counts c^* and Good-Turing smoothed probabilities p^*_{GT} for species with count 0 (like bass or catfish) or count 1 (like trout, salmon, or eel) are as follows:

	unseen (bass or catfish)	trout
c	0	1
MLE p	$p = \frac{0}{18} = 0$	$\frac{1}{18}$
c^*		$c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67$
GT p^*_{GT}	$p^*_{\text{GT}}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$	$p^*_{\text{GT}}(\text{trout}) = \frac{.67}{18} = \frac{1}{27} = .037$

Note that the revised count c^* for eel was discounted from $c = 1.0$ to $c^* = .67$, (thus leaving some probability mass $p^*_{\text{GT}}(\text{unseen}) = \frac{3}{18} = .17$ for the catfish and bass). And since we know there were 2 unknown species, the probability of the next fish being specifically a catfish is $p^*_{\text{GT}}(\text{catfish}) = \frac{1}{2} \times \frac{3}{18} = .085$.

Fig. 4.8 gives two examples of the application of Good-Turing discounting to bigram grammars, one on the BeRP corpus of 9332 sentences, and a larger example computed from 22 million words from the Associated Press (AP) newswire by Church and Gale (1991) . For both examples the first column shows the count c, i.e., the number of observed instances of a bigram. The second column shows the number of bigrams that had this count. Thus 449,721 of the AP bigrams have a count of 2. The third column shows c^*, the Good-Turing re-estimation of the count.

AP Newswire			Berkeley Restaurant		
c (MLE)	N_c	c^* (GT)	c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

Figure 4.8 Bigram "frequencies of frequencies" and Good-Turing re-estimations for the 22 million AP bigrams from Church and Gale (1991) and from the Berkeley Restaurant corpus of 9332 sentences.

4.5.3 Some advanced issues in Good-Turing estimation

Good-Turing estimation assumes that the distribution of each bigram is binomial (Church et al., 1991) and assumes we know N_0, the number of bigrams we haven't seen. We know this because given a vocabulary size of V, the total number of bigrams is V^2, hence N_0 is V^2 minus all the bigrams we have seen.

There are a number of additional complexities in the use of Good-Turing. For example, we don't just use the raw N_c values in Equation (4.26). This is because the re-estimate c^* for N_c depends on N_{c+1}, hence Equation (4.26) is undefined when $N_{c+1} = 0$. Such zeros occur quite often. In our sample problem above, for example, since $N_4 = 0$, how can we compute N_3? One solution to this is called **Simple Good-Turing** (Gale and Sampson, 1995). In Simple Good-Turing, after we compute the bins N_c, but before we compute Equation (4.26) from them, we smooth the N_c counts to replace any zeros in the sequence. The simplest thing is just to replace the value N_c with a value computed from a linear regression which is fit to map N_c to c in log space (see Gale and Sampson (1995) for details):

SIMPLE GOOD-TURING

$$\log(N_c) = a + b\,\log(c) \tag{4.29}$$

In addition, in practice, the discounted estimate c^* is not used for all counts c. Large counts (where $c > k$ for some threshold k) are assumed to be reliable. Katz (1987) suggests setting k at 5. Thus we define

$$c^* = c \;\text{ for } c > k \tag{4.30}$$

The correct equation for c^* when some k is introduced (from Katz (1987)) is:

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}, \;\text{ for } 1 \le c \le k. \tag{4.31}$$

Second, with Good-Turing discounting as with any other, it is usual to treat N-grams with low raw counts (especially counts of 1) as if the count were 0, i.e., to apply Good-Turing discounting to these as if they were unseen.

It turns out that Good-Turing discounting is not used by itself in discounting N-grams; it is only used in combination with the backoff and interpolation algorithms described in the next sections.

4.6 INTERPOLATION

The discounting we have been discussing so far can help solve the problem of zero frequency N-grams. But there is an additional source of knowledge we can draw on. If we are trying to compute $P(w_n|w_{n-1}w_{n-2})$, but we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$, we can instead estimate its probability by using the bigram probability $P(w_n|w_{n-1})$. Similarly, if we don't have counts to compute $P(w_n|w_{n-1})$, we can look to the unigram $P(w_n)$.

BACKOFF
INTERPOLATION

There are two ways to use this N-gram "hierarchy", **backoff** and **interpolation**. In backoff, if we have non-zero trigram counts, we rely solely on the trigram counts. We only "back off" to a lower order N-gram if we have zero evidence for a higher-order N-gram. By contrast, in interpolation, we always mix the probability estimates from all the N-gram estimators, i.e., we do a weighted interpolation of trigram, bigram, and unigram counts.

In simple linear interpolation, we combine different order N-grams by linearly interpolating all the models. Thus we estimate the trigram probability $P(w_n|w_{n-1}w_{n-2})$ by mixing together the unigram, bigram, and trigram probabilities, each weighted by a λ:

$$\begin{aligned}\hat{P}(w_n|w_{n-1}w_{n-2}) &= \lambda_1 P(w_n|w_{n-1}w_{n-2}) \\ &\quad +\lambda_2 P(w_n|w_{n-1}) \\ &\quad +\lambda_3 P(w_n)\end{aligned} \tag{4.32}$$

such that the λs sum to 1:

$$\sum_i \lambda_i = 1 \tag{4.33}$$

In a slightly more sophisticated version of linear interpolation, each λ weight is computed in a more sophisticated way, by conditioning on the context. This way if we have particularly accurate counts for a particular bigram, we assume that the counts of the trigrams based on this bigram will be more trustworthy, so we can make the λs for those trigrams higher and thus give that trigram more weight in the interpolation. Equation (4.34) shows the equation for interpolation with context-conditioned weights:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) \\ &\quad +\lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) \\ &\quad +\lambda_3(w_{n-2}^{n-1})P(w_n)\end{aligned} \tag{4.34}$$

HELD-OUT

How are these λ values set? Both the simple interpolation and conditional interpolation λs are learned from a **held-out** corpus. Recall from Sec. 4.3 that a held-out corpus is an additional training corpus that we use not to set the *N*-gram counts, but to set other parameters. In this case we can use such data to set the λ values. We can do this by choosing the λ values which maximize the likelihood of the held-out corpus. That is, we fix the *N*-gram probabilities and then search for the λ values that when plugged into Equation (4.32) give us the highest probability of the held-out set, There are various ways to find this optimal set of λs. One way is to use the **EM** algorithm to be defined in Ch. 6, which is an iterative learning algorithm that converges on locally optimal λs (Baum, 1972; Dempster et al., 1977; Jelinek and Mercer, 1980).

4.7 Backoff

While simple interpolation is indeed simple to understand and implement, it turns out that there are a number of better algorithms. One of these is backoff *N*-gram modeling. The version of backoff that we describe uses Good-Turing discounting as well. It was introduced by Katz (1987), hence this kind of backoff with discounting is also called **Katz backoff**. In a Katz backoff *N*-gram model, if the *N*-gram we need has zero counts, we approximate it by backing off to the (*N*-1)-gram. We continue backing off until we reach a history that has some counts:

KATZ BACKOFF

(4.35)
$$P_{\text{katz}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{\text{katz}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases}$$

Equation (4.35) shows that the Katz backoff probability for an *N*-gram just relies on the (discounted) probability P^* if we've seen this *N*-gram before (i.e. if we have non-zero counts). Otherwise, we recursively back off to the Katz probability for the shorter-history (*N*-1)-gram. We'll define the discounted probability P^*, the normalizing factor α, and other details about dealing with zero counts in Sec. 4.7.1. Based on these details, the trigram version of backoff might be represented as follows (where for pedagogical clarity, since it's easy to confuse the indices w_i, w_{i-1} and so on, we refer to the three words in a sequence as *x*, *y*, *z* in that order):

(4.36)
$$P_{\text{katz}}(z|x,y) = \begin{cases} P^*(z|x,y), & \text{if } C(x,y,z) > 0 \\ \alpha(x,y)P_{\text{katz}}(z|y), & \text{else if } C(x,y) > 0 \\ P^*(z), & \text{otherwise.} \end{cases}$$

(4.37)
$$P_{\text{katz}}(z|y) = \begin{cases} P^*(z|y), & \text{if } C(y,z) > 0 \\ \alpha(y)P^*(z), & \text{otherwise.} \end{cases}$$

Katz backoff incorporates discounting as an integral part of the algorithm. Our previous discussions of discounting showed how a method like Good-Turing could be used to assign probability mass to unseen events. For simplicity, we assumed that these unseen events were all equally probable, and so the probability mass got distributed evenly among all unseen events. Katz backoff gives us a better way to distribute the probability mass among unseen trigram events, by relying on information from unigrams and bigrams. We use discounting to tell us how much total probability mass to set aside for all the events we haven't seen and backoff to tell us how to distribute this probability.

Discounting is implemented by using discounted probabilities $P^*(\cdot)$ rather than MLE probabilities $P(\cdot)$ in Equation (4.35) and Equation (4.37).

Why do we need discounts and α values in Equation (4.35) and Equation (4.37)? Why couldn't we just have three sets of MLE probabilities without weights? Because without discounts and α weights, the result of the equation would not be a true probability! The MLE estimates of $P(w_n|w_{n-N+1}^{n-1})$ are true probabilities; if we sum the probability of all w_i over a given N-gram context, we should get 1:

$$\sum_i P(w_i|w_j w_k) = 1 \tag{4.38}$$

But if that is the case, if we use MLE probabilities but back off to a lower order model when the MLE probability is zero, we would be adding extra probability mass into the equation, and the total probability of a word would be greater than 1!

Thus any backoff language model must also be discounted. The P^* is used to discount the MLE probabilities to save some probability mass for the lower order N-grams. The α is used to ensure that the probability mass from all the lower order N-grams sums up to exactly the amount that we saved by discounting the higher-order N-grams. We define P^* as the discounted (c^*) estimate of the conditional probability of an N-gram, (and save P for MLE probabilities):

$$P^*(w_n|w_{n-N+1}^{n-1}) = \frac{c^*(w_{n-N+1}^{n})}{c(w_{n-N+1}^{n-1})} \tag{4.39}$$

Because on average the (discounted) c^* will be less than c, this probability P^* will be slightly less than the MLE estimate, which is

$$\frac{c(w_{n-N+1}^{n})}{c(w_{n-N+1}^{n-1})}$$

This will leave some probability mass for the lower order N-grams which is then distributed by the α weights; details of computing α are in Sec. 4.7.1. Fig. 4.9 shows the Katz backoff bigram probabilities for our 8 sample words, computed from the BeRP corpus using the SRILM toolkit.

4.7.1 Advanced: Details of computing Katz backoff α and P^*

In this section we give the remaining details of the computation of the discounted probability P^* and the backoff weights $\alpha(w)$.

We begin with α, which passes the left-over probability mass to the lower order N-grams. Let's represent the total amount of left-over probability mass by the function

	i	want	to	eat	chinese	food	lunch	spend
i	0.0014	0.326	0.00248	0.00355	0.000205	0.0017	0.00073	0.000489
want	0.00134	0.00152	0.656	0.000483	0.00455	0.00455	0.00384	0.000483
to	0.000512	0.00152	0.00165	0.284	0.000512	0.0017	0.00175	0.0873
eat	0.00101	0.00152	0.00166	0.00189	0.0214	0.00166	0.0563	0.000585
chinese	0.00283	0.00152	0.00248	0.00189	0.000205	0.519	0.00283	0.000585
food	0.0137	0.00152	0.0137	0.00189	0.000409	0.00366	0.00073	0.000585
lunch	0.00363	0.00152	0.00248	0.00189	0.000205	0.00131	0.00073	0.000585
spend	0.00161	0.00152	0.00161	0.00189	0.000205	0.0017	0.00073	0.000585

Figure 4.9 Good-Turing smoothed bigram probabilities for eight words (of $V = 1446$) in the BeRP corpus of 9332 sentences, computing by using SRILM, with $k = 5$ and counts of 1 replaced by 0.

β, a function of the (*N*-1)-gram context. For a given (*N*-1)-gram context, the total left-over probability mass can be computed by subtracting from 1 the total discounted probability mass for all *N*-grams starting with that context:

$$\beta(w_{n-N+1}^{n-1}) = 1 - \sum_{w_n:c(w_{n-N+1}^{n})>0} P^*(w_n|w_{n-N+1}^{n-1}) \tag{4.40}$$

This gives us the total probability mass that we are ready to distribute to all (*N*-1)-gram (e.g., bigrams if our original model was a trigram). Each individual (*N*-1)-gram (bigram) will only get a fraction of this mass, so we need to normalize β by the total probability of all the (*N*-1)-grams (bigrams) that begin some *N*-gram (trigram) which has zero count. The final equation for computing how much probability mass to distribute from an *N*-gram to an (*N*-1)-gram is represented by the function α:

$$\begin{aligned}\alpha(w_{n-N+1}^{n-1}) &= \frac{\beta(w_{n-N+1}^{n-1})}{\sum_{w_n:c(w_{n-N+1}^{n})=0} P_{\text{katz}}(w_n|w_{n-N+2}^{n-1})} \\ &= \frac{1 - \sum_{w_n:c(w_{n-N+1}^{n})>0} P^*(w_n|w_{n-N+1}^{n-1})}{1 - \sum_{w_n:c(w_{n-N+1}^{n})>0} P^*(w_n|w_{n-N+2}^{n-1})}\end{aligned} \tag{4.41}$$

Note that α is a function of the preceding word string, that is, of w_{n-N+1}^{n-1}; thus the amount by which we discount each trigram (*d*), and the mass that gets reassigned to lower order *N*-grams (α) are recomputed for every (*N*-1)-gram that occurs in any *N*-gram.

We only need to specify what to do when the counts of an (*N*-1)-gram context are 0, (i.e., when $c(w_{n-N+1}^{n-1}) = 0$) and our definition is complete:

$$P_{\text{katz}}(w_n|w_{n-N+1}^{n-1}) = P_{\text{katz}}(w_n|w_{n-N+2}^{n-1}) \qquad \text{if } c(w_{n-N+1}^{n-1}) = 0 \tag{4.42}$$

and

$$P^*(w_n|w_{n-N+1}^{n-1}) = 0 \qquad \text{if } c(w_{n-N+1}^{n-1}) = 0 \tag{4.43}$$

and

$$\beta(w_{n-N+1}^{n-1}) = 1 \qquad \text{if } c(w_{n-N+1}^{n-1}) = 0 \tag{4.44}$$

4.8 Practical Issues: Toolkits and Data Formats

Let's now examine how *N*-gram language models are represented. We represent and compute language model probabilities in log format, in order to avoid underflow and also to speed up computation. Since probabilities are (by definition) less than 1, the more probabilities we multiply together the smaller the product becomes. Multiplying enough *N*-grams together would result in numerical underflow. By using log probabilities instead of raw probabilities, the numbers are not as small. Since adding in log space is equivalent to multiplying in linear space, we combine log probabilities by adding them. Besides avoiding underflow, addition is faster to compute than multiplication. Since we do all computation and storage in log space, if we ever need to report probabilities we just take the exp of the logprob:

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4) \tag{4.45}$$

Backoff *N*-gram language models are generally stored in **ARPA format**. An *N*-gram in ARPA format is an ASCII file with a small header followed by a list of all the non-zero *N*-gram probabilities (all the unigrams, followed by bigrams, followed by trigrams, and so on). Each *N*-gram entry is stored with its discounted log probability (in $\log_{10}$ format) and its backoff weight α. Backoff weights are only necessary for *N*-grams which form a prefix of a longer *N*-gram, so no α is computed for the highest order *N*-gram (in this case the trigram) or *N*-grams ending in the end-of-sequence token `<s>`. Thus for a trigram grammar, the format of each *N*-gram is:

unigram:	$\log p^*(w_i)$	w_i	$\log \alpha(w_i)$
bigram:	$\log p^*(w_i \vert w_{i-1})$	$w_{i-1}w_i$	$\log \alpha(w_{i-1}w_i)$
trigram:	$\log p^*(w_i \vert w_{i-2}, w_{i-1})$	$w_{i-2}w_{i-1}w_i$	

Fig. 4.10 shows an ARPA formatted LM file with selected *N*-grams from the BeRP corpus. Given one of these trigrams, the probability $P(z|x,y)$ for the word sequence x,y,z can be computed as follows (repeated from (4.37)):

$$P_{\text{katz}}(z|x,y) = \begin{cases} P^*(z|x,y), & \text{if } C(x,y,z) > 0 \\ \alpha(x,y)P_{\text{katz}}(z|y), & \text{else if } C(x,y) > 0 \\ P^*(z), & \text{otherwise.} \end{cases} \tag{4.46}$$

$$P_{\text{katz}}(z|y) = \begin{cases} P^*(z|y), & \text{if } C(y,z) > 0 \\ \alpha(y)P^*(z), & \text{otherwise.} \end{cases} \tag{4.47}$$

Toolkits: There are two commonly used available toolkits for building language models, the SRILM toolkit (Stolcke, 2002) and the Cambridge-CMU toolkit (Clarkson and Rosenfeld, 1997). Both are publicly available, and have similar functionality.

```
\data\
ngram 1=1447
ngram 2=9420
ngram 3=5201

\1-grams:
-0.8679678     </s>
-99            <s>                            -1.068532
-4.743076      chow-fun                       -0.1943932
-4.266155      fries                          -0.5432462
-3.175167      thursday                       -0.7510199
-1.776296      want                           -1.04292
...

\2-grams:
-0.6077676     <s>    i                       -0.6257131
-0.4861297     i      want                    0.0425899
-2.832415      to     drink                   -0.06423882
-0.5469525     to     eat                     -0.008193135
-0.09403705    today  </s>
...

\3-grams:
-2.579416      <s>    i          prefer
-1.148009      <s>    about      fifteen
-0.4120701     to     go         to
-0.3735807     me     a          list
-0.260361      at     jupiter    </s>
-0.260361      a      malaysian  restaurant
...
\end\
```

Figure 4.10 ARPA format for N-grams, showing some sample N-grams. Each is represented by a *logprob*, the word sequence, $w_1 \dots w_n$, followed by the log backoff weight α. Note that no α is computed for the highest-order N-gram or for N-grams ending in <s>.

In training mode, each toolkit takes a raw text file, one sentence per line with words separated by white-space, and various parameters such as the order N, the type of discounting (Good Turing or Kneser-Ney, discussed in Sec. 4.9.1), and various thresholds. The output is a language model in ARPA format. In perplexity or decoding mode, the toolkits take a language model in ARPA format, and a sentence or corpus, and produce the probability and perplexity of the sentence or corpus. Both also implement many advanced features to be discussed later in this chapter and in following chapters, including skip N-grams, word lattices, confusion networks, and N-gram pruning.

4.9 Advanced Issues in Language Modeling

4.9.1 Advanced Smoothing Methods: Kneser-Ney Smoothing

KNESER-NEY

In this section we give a brief introduction to the most commonly used modern N-gram smoothing method, the interpolated **Kneser-Ney** algorithm.

Kneser-Ney has its roots in a discounting method called **absolute discounting**. Absolute discounting is a much better method of computing a revised count $c*$ than the Good-Turing discount formula we saw in Equation (4.26), based on frequencies-of-frequencies. To get the intuition, let's revisit the Good-Turing estimates of the bigram c^* extended from Fig. 4.8 and reformatted below:

c (MLE)	0	1	2	3	4	5	6	7	8	9
c^* (GT)	0.0000270	0.446	1.26	2.24	3.24	4.22	5.19	6.21	7.24	8.25

ABSOLUTE DISCOUNTING

The astute reader may have noticed that except for the re-estimated counts for 0 and 1, all the other re-estimated counts c^* could be estimated pretty well by just subtracting 0.75 from the MLE count c! **Absolute discounting** formalizes this intuition, by subtracting a fixed (absolute) discount d from each count. The intuition is that we have good estimates already for the high counts, and a small discount d won't affect them much. It will mainly modify the smaller counts, for which we don't necessarily trust the estimate anyway. The equation for absolute discounting applied to bigrams (assuming a proper coefficient α on the backoff to make everything sum to one) is:

$$P_{\text{absolute}}(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1}w_i)-\mathbf{D}}{C(w_{i-1})}, & \text{if } C(w_{i-1}w_i) > 0 \\ \alpha(w_i)P_{\text{absolute}}(w_i), & \text{otherwise.} \end{cases} \tag{4.48}$$

In practice, we might also want to keep distinct discount values d for the 0 and 1 counts.

Kneser-Ney discounting (Kneser and Ney, 1995) augments absolute discounting with a more sophisticated way to handle the backoff distribution. Consider the job of predicting the next word in this sentence, assuming we are backing off to a unigram model:

I can't see without my reading ________________ .

The word *glasses* seems much more likely to follow here than the word *Francisco*. But *Francisco* is in fact more common, so a unigram model will prefer it to *glasses*. We would like to capture the intuition that although *Francisco* is frequent, it is only frequent after the word *San*, i.e. in the phrase *San Francisco*. The word *glasses* has a much wider distribution.

Thus instead of backing off to the unigram MLE count (the number of times the word w has been seen), we want to use a completely different backoff distribution! We want a heuristic that more accurately estimates the number of times we might expect to see word w in a new unseen context. The Kneser-Ney intuition is to base our estimate on the *number of different contexts word w has appeared in*. Words that have appeared in more contexts are more likely to appear in some new context as well. We can express this new backoff probability, the "continuation probability", as follows:

$$P_{\text{CONTINUATION}}(w_i) = \frac{|\{w_{i-1} : C(w_{i-1}w_i) > 0\}|}{\sum_{w_i} |\{w_{i-1} : C(w_{i-1}w_i) > 0\}|} \tag{4.49}$$

The Kneser-Ney backoff intuition can be formalized as follows (again assuming a proper coefficient α on the backoff to make everything sum to one):

$$P_{\text{KN}}(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1}w_i)-\mathbf{D}}{C(w_{i-1})}, & \text{if } C(w_{i-1}w_i) > 0 \\ \alpha(w_i)\frac{|\{w_{i-1}:C(w_{i-1}w_i)>0\}|}{\sum_{w_i}|\{w_{i-1}:C(w_{i-1}w_i)>0\}|} & \text{otherwise.} \end{cases} \tag{4.50}$$

Finally, it turns out to be better to use an **interpolated** rather than **backoff** form of Kneser-Ney. While Sec. 4.6 showed that *linear* interpolation is not as successful

INTERPOLATED KNESER-NEY

as Katz backoff, it turns out that more powerful interpolated models, such as interpolated Kneser-Ney, work better than their backoff version. **Interpolated Kneser-Ney** discounting can be computed with an equation like the following (omitting the computation of β):

$$P_{\mathrm{KN}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - \mathbf{D}}{C(w_{i-1})} + \beta(w_i)\frac{|\{w_{i-1} : C(w_{i-1}w_i) > 0\}|}{\sum_{w_i}|\{w_{i-1} : C(w_{i-1}w_i) > 0\}|} \tag{4.51}$$

A final practical note: it turns out that any interpolation model can be represented as a backoff model, hence stored in ARPA backoff format. We simply do the interpolation when we build the model, so the ‘bigram’ probability stored in the backoff format is really ‘bigram already interpolated with unigram’.

4.9.2 Class-based N-grams

CLASS-BASED N-GRAM
CLUSTER N-GRAM

The **class-based N-gram** or **cluster N-gram** is a variant of the N-gram that uses information about word classes or clusters. Class-based N-grams can be useful for dealing with sparsity in the training data. Suppose for a flight reservation system we want to compute the probability of the bigram *to Shanghai*, but this bigram never occurs in the training set. Instead, our training data has *to London*, *to Beijing*, and *to Denver*. If we knew that these were all cities, and assuming *Shanghai* does appear in the training set in other contexts, we could predict the likelihood of a city following *from*.

IBM CLUSTERING

There are many variants of cluster N-grams. The simplest one is sometimes known as **IBM clustering**, after its originators (Brown et al., 1992b). IBM clustering is a kind of **hard clustering**, in which each word can belong to only one class. The model estimates the conditional probability of a word w_i by multiplying two factors: the probability of the word’s class c_i given the preceding classes (based on an N-gram of classes), and the probability of w_i given c_i. Here is the IBM model in bigram form:

$$P(w_i|w_{i-1}) \approx P(c_i|,c_{i-1}) \times P(w_i|c_i)$$

If we had a training corpus in which we knew the class for each word, the maximum likelihood estimate (MLE) of the probability of the word given the class and the probability of the class given the previous class could be computed as follows:

$$\begin{aligned} P(w|c) &= \frac{C(w)}{C(c)} \\ P(c_i|c_{i-1}) &= \frac{C(c_{i-1}c_i)}{\sum_c C(c_{i-1}c)} \end{aligned}$$

Cluster N-grams are generally used in two ways. In dialog systems (Ch. 24), we often hand-design domain-specific word classes. Thus for an airline information system, we might use classes like CITYNAME, AIRLINE, DAYOFWEEK, or MONTH. In other cases, we can automatically induce the classes by clustering words in a corpus (Brown et al., 1992b). Syntactic categories like part-of-speech tags don't seem to work well as classes (Niesler et al., 1998).

Whether automatically induced or hand-designed, cluster N-grams are generally mixed with regular word-based N-grams.

4.9.3 Language Model Adaptation and Using the Web

ADAPTATION

One of the most exciting recent developments in language modeling is language model **adaptation**. This is relevant when we have only a small amount of in-domain training data, but a large amount of data from some other domain. We can train on the larger out-of-domain dataset and adapt our models to the small in-domain set. (Iyer and Ostendorf, 1997, 1999a, 1999b; Bacchiani and Roark, 2003; Bacchiani et al., 2004).

An obvious large data source for this type of adaptation is the web. Indeed, use of the web does seem to be helpful in language modeling. The simplest way to apply the web to improve, say, trigram language models is to use search engines to get counts for $w_1w_2w_3$ and $w_1w_2w_3$, and then compute:

$$\hat{p}_{web} = \frac{c_{web}(w_1w_2w_3)}{c_{web}(w_1w_2)} \tag{4.52}$$

We can then mix $\hat{p}_{web}$ with a conventional N-gram (Berger and Miller, 1998; Zhu and Rosenfeld, 2001). We can also use more sophisticated combination methods that make use of topic or class dependencies, to find domain-relevant data on the web data (Bulyko et al., 2003).

In practice it is difficult or impossible to download every page from the web in order to compute N-grams. For this reason most uses of web data rely on page counts from search engines. Page counts are only an approximation to actual counts for many reasons: a page may contain an N-gram multiple times, most search engines round off their counts, punctuation is deleted, and the counts themselves may be adjusted due to link and other information. It seems that this kind of noise does not hugely affect the results of using the web as a corpus (Keller and Lapata, 2003; Nakov and Hearst, 2005), although it is possible to perform specific adjustments, such as fitting a regression to predict actual word counts from page counts (Zhu and Rosenfeld, 2001).

4.9.4 Using Longer Distance Information: A Brief Summary

There are many methods for incorporating longer-distance context into N-gram modeling. While we have limited our discussion mainly to bigram and trigrams, state-of-the-art speech recognition systems, for example, are based on longer-distance N-grams, especially 4-grams, but also 5-grams. Goodman (2006) showed that with 284 million words of training data, 5-grams do improve perplexity scores over 4-grams, but not by much. Goodman checked contexts up to 20-grams, and found that after 6-grams, longer contexts weren't useful, at least not with 284 million words of training data.

CACHE

Many models focus on more sophisticated ways to get longer-distance information. For example people tend to repeat words they have used before. Thus if a word is used once in a text, it will probably be used again. We can capture this fact by a **cache** language model (Kuhn and De Mori, 1990). For example to use a unigram cache model to predict word *i* of a test corpus, we create a unigram grammar from the preceding part of the test corpus (words 1 to $i-1$) and mix this with our conventional N-gram. We might use only a shorter window from the previous words, rather than the entire set. Cache language models are very powerful in any applications where we have perfect knowledge of the words. Cache models work less well in domains where the previous words are not known exactly. In speech applications, for example, unless there is some way for users to correct errors, cache models tend to “lock in” errors they made on earlier words.

TOPIC-BASED

The fact that words are often repeated in a text is a symptom of a more general fact about words; texts tend to be **about** things. Documents which are about particular topics tend to use similar words. This suggests that we could train separate language models for different topics. In **topic-based** language models (Chen et al., 1998; Gildea and Hofmann, 1999), we try to take advantage of the fact that different topics will have different kinds of words. For example we can train different language models for each topic t, and then mix them, weighted by how likely each topic is given the history h:

$$p(w|h) = \sum_t P(w|t)P(t|h) \tag{4.53}$$

LATENT SEMANTIC INDEXING

A very similar class of models relies on the intuition that upcoming words are semantically similar to preceding words in the text. These models use a measure of semantic word association such as the **latent semantic indexing** described in Ch. 20 (Coccaro and Jurafsky, 1998; Bellegarda, 1999, 2000), or on-line dictionaries or thesauri (Demetriou et al., 1997) to compute a probability based on a word’s similarity to preceding words, and then mix it with a conventional N-gram.

TRIGGER

SKIP N-GRAMS

VARIABLE-LENGTH N-GRAM

There are also various ways to extend the N-gram model by having the previous (conditioning) word be something other than a fixed window of previous words. For example we can choose as a predictor a word called a **trigger** which is not adjacent but which is very related (has high mutual information with) the word we are trying to predict (Rosenfeld, 1996; Niesler and Woodland, 1999; Zhou and Lua, 1998). Or we can create **skip N-grams**, where the preceding context ‘skips over’ some intermediate words, for example computing a probability such as $P(w_i|w_{i-1}, w_{i-3})$. We can also use extra previous context just in cases where a longer phrase is particularly frequent or predictive, producing a **variable-length N-gram** (Ney et al., 1994; Kneser, 1996; Niesler and Woodland, 1996).

In general, using very large and rich contexts can result in very large language models. Thus these models are often pruned by removing low-probability events. Pruning is also essential for using language models on small platforms such as cellphones (Stolcke, 1998; Church et al., 2007).

Finally, there is a wide body of research on integrating sophisticated linguistic structures into language modeling. Language models based on syntactic structure from probabilistic parsers are described in Ch. 14. Language models based on the current speech act in dialogue are described in Ch. 24.

4.10 ADVANCED: INFORMATION THEORY BACKGROUND

I got the horse right here
Frank Loesser, *Guys and Dolls*

We introduced perplexity in Sec. 4.4 as a way to evaluate N-gram models on a test set. A better N-gram model is one which assigns a higher probability to the test data, and perplexity is a normalized version of the probability of the test set. Another way to think about perplexity is based on the information-theoretic concept of cross-entropy. In order to give another intuition into perplexity as a metric, this section gives a quick review of fundamental facts from **information theory** including the concept of cross-entropy that underlies perplexity. The interested reader should consult a good information theory textbook like Cover and Thomas (1991).

ENTROPY

Perplexity is based on the information-theoretic notion of **cross-entropy**, which we will now work toward defining. **Entropy** is a measure of information, and is invaluable throughout speech and language processing. It can be used as a metric for how much information there is in a particular grammar, for how well a given grammar matches a given language, for how predictive a given N-gram grammar is about what the next word could be. Given two grammars and a corpus, we can use entropy to tell us which grammar better matches the corpus. We can also use entropy to compare how difficult two speech recognition tasks are, and also to measure how well a given probabilistic grammar matches human grammars.

Computing entropy requires that we establish a random variable X that ranges over whatever we are predicting (words, letters, parts of speech, the set of which we'll call χ), and that has a particular probability function, call it $p(x)$. The entropy of this random variable X is then

$$H(X) = -\sum_{x\in\chi} p(x)\log_2 p(x) \tag{4.54}$$

The log can in principle be computed in any base. If we use log base 2, the resulting value of entropy will be measured in **bits**.

The most intuitive way to define entropy for computer scientists is to think of the entropy as a lower bound on the number of bits it would take to encode a certain decision or piece of information in the optimal coding scheme.

Cover and Thomas (1991) suggest the following example. Imagine that we want to place a bet on a horse race but it is too far to go all the way to Yonkers Racetrack, and we'd like to send a short message to the bookie to tell him which horse to bet on. Suppose there are eight horses in this particular race.

One way to encode this message is just to use the binary representation of the horse's number as the code; thus horse 1 would be `001`, horse 2 `010`, horse 3 `011`, and so on, with horse 8 coded as `000`. If we spend the whole day betting, and each horse is coded with 3 bits, on average we would be sending 3 bits per race.

Can we do better? Suppose that the spread is the actual distribution of the bets placed, and that we represent it as the prior probability of each horse as follows:

Horse 1	$\frac{1}{2}$	Horse 5	$\frac{1}{64}$
Horse 2	$\frac{1}{4}$	Horse 6	$\frac{1}{64}$
Horse 3	$\frac{1}{8}$	Horse 7	$\frac{1}{64}$
Horse 4	$\frac{1}{16}$	Horse 8	$\frac{1}{64}$

The entropy of the random variable X that ranges over horses gives us a lower bound on the number of bits, and is:

$$\begin{aligned} H(X) &= -\sum_{i=1}^{i=8} p(i)\log p(i) \\ &= -\tfrac{1}{2}\log\tfrac{1}{2}-\tfrac{1}{4}\log\tfrac{1}{4}-\tfrac{1}{8}\log\tfrac{1}{8}-\tfrac{1}{16}\log\tfrac{1}{16}-4(\tfrac{1}{64}\log\tfrac{1}{64}) \\ &= 2 \text{ bits} \end{aligned} \tag{4.55}$$

A code that averages 2 bits per race can be built by using short encodings for more probable horses, and longer encodings for less probable horses. For example, we could encode the most likely horse with the code `0`, and the remaining horses as `10`, then `110`, `1110`, `111100`, `111101`, `111110`, and `111111`.

What if the horses are equally likely? We saw above that if we used an equal-length binary code for the horse numbers, each horse took 3 bits to code, so the average was 3. Is the entropy the same? In this case each horse would have a probability of $\frac{1}{8}$. The entropy of the choice of horses is then:

$$H(X) = -\sum_{i=1}^{i=8}\frac{1}{8}\log\frac{1}{8} = -\log\frac{1}{8} = 3 \text{ bits} \tag{4.56}$$

Until now we have been computing the entropy of a single variable. But most of what we will use entropy for involves *sequences*. For a grammar, for example, we will be computing the entropy of some sequence of words $W = \{w_0, w_1, w_2, \ldots, w_n\}$. One way to do this is to have a variable that ranges over sequences of words. For example we can compute the entropy of a random variable that ranges over all finite sequences of words of length n in some language L as follows:

$$H(w_1, w_2, \ldots, w_n) = -\sum_{W_1^n \in L} p(W_1^n)\log p(W_1^n) \tag{4.57}$$

ENTROPY RATE

We could define the **entropy rate** (we could also think of this as the **per-word entropy**) as the entropy of this sequence divided by the number of words:

$$\frac{1}{n}H(W_1^n) = -\frac{1}{n}\sum_{W_1^n \in L} p(W_1^n)\log p(W_1^n) \tag{4.58}$$

But to measure the true entropy of a language, we need to consider sequences of infinite length. If we think of a language as a stochastic process L that produces a sequence of words, its entropy rate $H(L)$ is defined as:

$$\begin{aligned} H(L) &= -\lim_{n\to\infty}\frac{1}{n}H(w_1,w_2,\ldots,w_n) \\ &= -\lim_{n\to\infty}\frac{1}{n}\sum_{W\in L}p(w_1,\ldots,w_n)\log p(w_1,\ldots,w_n) \end{aligned} \tag{4.59}$$

The Shannon-McMillan-Breiman theorem (Algoet and Cover, 1988; Cover and Thomas, 1991) states that if the language is regular in certain ways (to be exact, if it is both stationary and ergodic),

$$H(L) = \lim_{n\to\infty} -\frac{1}{n}\log p(w_1w_2\ldots w_n) \tag{4.60}$$

That is, we can take a single sequence that is long enough instead of summing over all possible sequences. The intuition of the Shannon-McMillan-Breiman theorem is that a long enough sequence of words will contain in it many other shorter sequences, and that each of these shorter sequences will reoccur in the longer sequence according to their probabilities.

STATIONARY

A stochastic process is said to be **stationary** if the probabilities it assigns to a sequence are invariant with respect to shifts in the time index. In other words, the probability distribution for words at time t is the same as the probability distribution at time $t+1$. Markov models, and hence N-grams, are stationary. For example, in a bigram, P_i is dependent only on P_{i-1}. So if we shift our time index by x, P_{i+x} is still dependent on P_{i+x-1}. But natural language is not stationary, since as we will see in Ch. 12, the probability of upcoming words can be dependent on events that were arbitrarily distant and time dependent. Thus our statistical models only give an approximation to the correct distributions and entropies of natural language.

To summarize, by making some incorrect but convenient simplifying assumptions, we can compute the entropy of some stochastic process by taking a very long sample of the output, and computing its average log probability. In the next section we talk about the why and how: *why* we would want to do this (i.e., for what kinds of problems would the entropy tell us something useful), and *how* to compute the probability of a very long sequence.

4.10.1 Cross-Entropy for Comparing Models

CROSS-ENTROPY

In this section we introduce **cross-entropy**, and discuss its usefulness in comparing different probabilistic models. The cross-entropy is useful when we don't know the actual probability distribution p that generated some data. It allows us to use some m, which is a model of p (i.e., an approximation to p). The cross-entropy of m on p is defined by:

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\sum_{W\in L}p(w_1,\ldots,w_n)\log m(w_1,\ldots,w_n) \tag{4.61}$$

That is, we draw sequences according to the probability distribution p, but sum the log of their probabilities according to m.

Again, following the Shannon-McMillan-Breiman theorem, for a stationary ergodic process:

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\log m(w_1 w_2 \ldots w_n) \tag{4.62}$$

This means that, as for entropy, we can estimate the cross-entropy of a model m on some distribution p by taking a single sequence that is long enough instead of summing over all possible sequences.

What makes the cross entropy useful is that the cross entropy $H(p,m)$ is an upper bound on the entropy $H(p)$. For any model m:

$$H(p) \le H(p,m) \tag{4.63}$$

This means that we can use some simplified model m to help estimate the true entropy of a sequence of symbols drawn according to probability p. The more accurate m is, the closer the cross entropy $H(p,m)$ will be to the true entropy $H(p)$. Thus the difference between $H(p,m)$ and $H(p)$ is a measure of how accurate a model is. Between two models m_1 and m_2, the more accurate model will be the one with the lower cross-entropy. (The cross-entropy can never be lower than the true entropy, so a model cannot err by underestimating the true entropy).

We are finally ready to see the relation between perplexity and cross-entropy as we saw it in Equation (4.62). Cross-entropy is defined in the limit, as the length of the observed word sequence goes to infinity. We will need an approximation to cross-entropy, relying on a (sufficiently long) sequence of fixed length. This approximation to the cross-entropy of a model $M = P(w_i|w_{i-N+1}...w_{i-1})$ on a sequence of words W is:

$$H(W) = -\frac{1}{N}\log P(w_1 w_2 \ldots w_N) \tag{4.64}$$

PERPLEXITY

The **perplexity** of a model P on a sequence of words W is now formally defined as the exp of this cross-entropy:

$$\begin{aligned}
\text{Perplexity}(W) &= 2^{H(W)} \\
&= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \\
&= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}
\end{aligned} \tag{4.65}$$

4.11 ADVANCED: THE ENTROPY OF ENGLISH AND ENTROPY RATE CONSTANCY

As we suggested in the previous section, the cross-entropy of some model *m* can be used as an upper bound on the true entropy of some process. We can use this method to get an estimate of the true entropy of English. Why should we care about the entropy of English?

One reason is that the true entropy of English would give us a solid lower bound for all of our future experiments on probabilistic grammars. Another is that we can use the entropy values for English to help understand what parts of a language provide the most information (for example, is the predictability of English mainly based on word order, on semantics, on morphology, on constituency, or on pragmatic cues?) This can help us immensely in knowing where to focus our language-modeling efforts.

There are two common methods for computing the entropy of English. The first was employed by Shannon (1951), as part of his groundbreaking work in defining the field of information theory. His idea was to use human subjects, and to construct a psychological experiment that requires them to guess strings of letters. By looking at how many guesses it takes them to guess letters correctly we can estimate the probability of the letters, and hence the entropy of the sequence.

The actual experiment is designed as follows: we present a subject with some English text and ask the subject to guess the next letter. The subjects will use their knowledge of the language to guess the most probable letter first, the next most probable next, and so on. We record the number of guesses it takes for the subject to guess correctly. Shannon's insight was that the entropy of the number-of-guesses sequence is the same as the entropy of English. (The intuition is that given the number-of-guesses sequence, we could reconstruct the original text by choosing the "*n*th most probable" letter whenever the subject took *n* guesses). This methodology requires the use of letter guesses rather than word guesses (since the subject sometimes has to do an exhaustive search of all the possible letters!), so Shannon computed the **per-letter entropy** of English rather than the per-word entropy. He reported an entropy of 1.3 bits (for 27 characters (26 letters plus space)). Shannon's estimate is likely to be too low, since it is based on a single text (*Jefferson the Virginian* by Dumas Malone). Shannon notes that his subjects had worse guesses (hence higher entropies) on other texts (newspaper writing, scientific work, and poetry). More recent variations on the Shannon experiments include the use of a gambling paradigm where the subjects get to bet on the next letter (Cover and King, 1978; Cover and Thomas, 1991).

The second method for computing the entropy of English helps avoid the single-text problem that confounds Shannon's results. This method is to take a very good stochastic model, train it on a very large corpus, and use it to assign a log-probability to a very long sequence of English, using the Shannon-McMillan-Breiman theorem:

$$H(\text{English}) \leq \lim_{n\to\infty} -\frac{1}{n}\log m(w_1w_2\ldots w_n) \tag{4.66}$$

For example, Brown et al. (1992a) trained a trigram language model on 583 million words of English (293,181 different types) and used it to compute the probability of the entire Brown corpus (1,014,312 tokens). The training data include newspapers, encyclopedias, novels, office correspondence, proceedings of the Canadian parliament, and other miscellaneous sources.

They then computed the character entropy of the Brown corpus by using their word-trigram grammar to assign probabilities to the Brown corpus, considered as a sequence of individual letters. They obtained an entropy of 1.75 bits per character (where the set of characters included all the 95 printable ASCII characters).

The average length of English written words (including space) has been reported at 5.5 letters (Nádas, 1984). If this is correct, it means that the Shannon estimate of 1.3 bits per letter corresponds to a per-word perplexity of 142 for general English. The numbers we report earlier for the WSJ experiments are significantly lower than this, since the training and test set came from the same subsample of English. That is, those experiments underestimate the complexity of English (since the Wall Street Journal looks very little like Shakespeare, for example)

A number of scholars have independently made the intriguing suggestion that entropy rate plays a role in human communication in general (Lindblom, 1990; Van Son et al., 1998; Aylett, 1999; Genzel and Charniak, 2002; Van Son and Pols, 2003). The idea is that people speak so as to keep the rate of information being transmitted per second roughly constant, i.e., transmitting a constant number of bits per second, or maintaining a constant entropy rate. Since the most efficient way of transmitting information through a channel is at a constant rate, language may even have evolved for such communicative efficiency (Plotkin and Nowak, 2000). There is a wide variety of evidence for the constant entropy rate hypothesis. One class of evidence, for speech, shows that speakers shorten predictable words (i.e., they take less time to say predictable words) and lengthen unpredictable words (Aylett, 1999; Jurafsky et al., 2001; Aylett and Turk, 2004). In another line of research, Genzel and Charniak (2002, 2003) show that entropy rate constancy makes predictions about the entropy of individual sentences from a text. In particular, they show that it predicts that local measures of sentence entropy which ignore previous discourse context (for example the *N*-gram probability of sentence), should increase with the sentence number, and they document this increase in corpora. Keller (2004) provides evidence that entropy rate plays a role for the addressee as well, showing a correlation between the entropy of a sentence and the processing effort it causes in comprehension, as measured by reading times in eye-tracking data.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The underlying mathematics of the *N*-gram was first proposed by Markov (1913), who used what are now called **Markov chains** (bigrams and trigrams) to predict whether an upcoming letter in Pushkin's *Eugene Onegin* would be a vowel or a consonant. Markov classified 20,000 letters as V or C and computed the bigram and trigram probability that a given letter would be a vowel given the previous one or two letters. Shannon (1948) applied *N*-grams to compute approximations to English word sequences. Based on Shannon's work, Markov models were commonly used in engineering, linguistic, and psychological work on modeling word sequences by the 1950s.

In a series of extremely influential papers starting with Chomsky (1956) and including Chomsky (1957) and Miller and Chomsky (1963), Noam Chomsky argued that "finite-state Markov processes", while a possibly useful engineering heuristic, were incapable of being a complete cognitive model of human grammatical knowledge. These arguments led many linguists and computational linguists to ignore work in statistical modeling for decades.

The resurgence of N-gram models came from Jelinek, Mercer, Bahl, and colleagues at the IBM Thomas J. Watson Research Center, who were influenced by Shannon, and Baker at CMU, who was influenced by the work of Baum and colleagues. Independently these two labs successfully used N-grams in their speech recognition systems (Baker, 1990; Jelinek, 1976; Baker, 1975; Bahl et al., 1983; Jelinek, 1990). A trigram model was used in the IBM TANGORA speech recognition system in the 1970s, but the idea was not written up until later.

Add-one smoothing derives from Laplace's 1812 law of succession, and was first applied as an engineering solution to the zero-frequency problem by Jeffreys (1948) based on an earlier Add-K suggestion by Johnson (1932). Problems with the Add-one algorithm are summarized in Gale and Church (1994). The Good-Turing algorithm was first applied to the smoothing of N-gram grammars at IBM by Katz, as cited in Nádas (1984). Church and Gale (1991) give a good description of the Good-Turing method, as well as the proof. Sampson (1996) also has a useful discussion of Good-Turing. Jelinek (1990) summarizes this and many other early language model innovations used in the IBM language models.

A wide variety of different language modeling and smoothing techniques were tested through the 1980's and 1990's, including Witten-Bell discounting (Witten and Bell, 1991), varieties of class-based models (Jelinek, 1990; Kneser and Ney, 1993; Heeman, 1999; Samuelsson and Reichl, 1999), and others (Gupta et al., 1992). In the late 1990's, Chen and Goodman produced a very influential series of papers with a comparison of different language models (Chen and Goodman, 1996, 1998, 1999; Goodman, 2006). They performed a number of carefully controlled experiments comparing different discounting algorithms, cache models, class-based (cluster) models, and other language model parameters. They showed the advantages of Interpolated Kneser-Ney, which has since become one of the most popular current methods for language modeling. These papers influenced our discussion in this chapter, and are recommended reading if you have further interest in language modeling.

As we suggested earlier in the chapter, recent research in language modeling has focused on adaptation, on the use of sophisticated linguistic structures based on syntactic and dialogue structure, and on very very large N-grams. For example in 2006, Google publicly released a very large set of N-grams that is a useful research resource, consisting of all the five-word sequences that appear at least 40 times from 1,024,908,267,229 words of running text; there are 1,176,470,663 five-word sequences using over 13 million unique words types (Franz and Brants, 2006). Large language models generally need to be pruned to be practical, using techniques such as Stolcke (1998) and Church et al. (2007).

4.12 Summary

This chapter introduced the N-gram, one of the oldest and most broadly useful practical tools in language processing.

- An N-gram probability is the conditional probability of a word given the previous $N-1$ words. N-gram probabilities can be computed by simply counting in a corpus and normalizing (the **Maximum Likelihood Estimate**) or they can be computed by more sophisticated algorithms. The advantage of N-grams is that they take advantage of lots of rich lexical knowledge. A disadvantage for some purposes is that they are very dependent on the corpus they were trained on.
- **Smoothing** algorithms provide a better way of estimating the probability of N-grams than Maximum Likelihood Estimation. Commonly used N-gram smoothing algorithms rely on lower-order N-gram counts via **backoff** or **interpolation**.
- Both backoff and interpolation require discounting such as **Kneser-Ney**, **Witten-Bell** or **Good-Turing** discounting.
- N-gram **language models** are evaluated by separating the corpus into a **training set** and a **test set**, training the model on the training set, and evaluating on the test set. The **perplexity** 2^H of of the language model on a test set is used to compare language models.

Exercises

4.1 Write out the equation for trigram probability estimation (modifying Eq. 4.14).

4.2 Write a program to compute unsmoothed unigrams and bigrams.

4.3 Run your N-gram program on two different small corpora of your choice (you might use email text or newsgroups). Now compare the statistics of the two corpora. What are the differences in the most common unigrams between the two? How about interesting differences in bigrams?

4.4 Add an option to your program to generate random sentences.

4.5 Add an option to your program to do Good-Turing discounting.

4.6 Add an option to your program to implement Katz backoff.

4.7 Add an option to your program to compute the perplexity of a test set.

4.8 (Adapted from Michael Collins). Prove Equation (4.27) given Equation (4.26) and any necessary assumptions. That is, show that given a probability distribution defined by the GT formula in Equation (4.26) for the N items seen in training, that the probability of the next, (i.e. $N+1$st) item being unseen in training can be estimated by Equation (4.27). You may make any necessary assumptions for the proof, including assuming that all N_c are non-zero.

BAG OF WORDS

BAG GENERATION

4.9 (Advanced) Suppose someone took all the words in a sentence and reordered them randomly. Write a program which take as input such a **bag of words** and produces as output a guess at the original order. You will need to an N-gram grammar produced by your N-gram program (on some corpus), and you will need to use the Viterbi algorithm introduced in the next chapter. This task is sometimes called **bag generation**.

AUTHORSHIP ATTRIBUTION

4.10 The field of **authorship attribution** is concerned with discovering the author of a particular text. Authorship attribution is important in many fields, including history, literature, and forensic linguistics. For example Mosteller and Wallace (1964) applied authorship identification techniques to discover who wrote *The Federalist* papers. The Federalist papers were written in 1787-1788 by Alexander Hamilton, John Jay and James Madison to persuade New York to ratify the United States Constitution. They were published anonymously, and as a result, although some of the 85 essays were clearly attributable to one author or another, the authorship of 12 were in dispute between Hamilton and Madison. Foster (1989) applied authorship identification techniques to suggest that W.S.'s *Funeral Elegy* for William Peter might have been written by William Shakespeare (he turned out to be wrong on this one), and that the anonymous author of *Primary Colors*, the roman à clef about the Clinton campaign for the American presidency, was journalist Joe Klein (Foster, 1996).

A standard technique for authorship attribution, first used by Mosteller and Wallace, is a Bayesian approach. For example, they trained a probabilistic model of the writing of Hamilton and another model on the writings of Madison, then computed the maximum-likelihood author for each of the disputed essays. There are many complex factors that go into these models, including vocabulary use, word length, syllable structure, rhyme, grammar; see Holmes (1994) for a summary. This approach can also be used for identifying which genre a text comes from.

One factor in many models is the use of rare words. As a simple approximation to this one factor, apply the Bayesian method to the attribution of any particular text. You will need three things: a text to test and two potential authors or genres, with a large on-line text sample of each. One of them should be the correct author. Train a unigram language model on each of the candidate authors. You are only going to use the **singleton** unigrams in each language model. You will compute $P(T|A_1)$, the probability of the text given author or genre A_1, by (1) taking the language model from A_1, (2) by multiplying together the probabilities of all the unigrams that only occur once in the "unknown" text and (3) taking the geometric mean of these (i.e., the nth root, where n is the number of probabilities you multiplied). Do the same for A_2. Choose whichever is higher. Did it produce the correct candidate?

Algoet, P. H. and Cover, T. M. (1988). A sandwich proof of the Shannon-McMillan-Breiman theorem. *The Annals of Probability*, *16*(2), 899–909.

Aylett, M. P. (1999). Stochastic suprasegmentals - relationships between redundancy, prosodic structure and syllable duration. In *Proceedings of the International Congress of Phonetic Sciences (ICPhS-99)*, San Francisco, California.

Aylett, M. P. and Turk, A. (2004). The smooth signal redundancy hypothesis: A functional explanation for relationships between redundancy, prosodic prominence, and duration in spontaneous speech. *Language and Speech*, *47*(1), 31–56.

Bacchiani, M. and Roark, B. (2003). Unsupervised language model adaptation. In *IEEE ICASSP-03*, pp. 224–227.

Bacchiani, M., Roark, B., and Saraclar, M. (2004). Language model adaptation with MAP estimation and the perceptron algorithm. In *HLT-NAACL-04*, pp. 21–24.

Bahl, L. R., Jelinek, F., and Mercer, R. L. (1983). A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *5*(2), 179–190.

Baker, J. K. (1975). The DRAGON system – An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *ASSP-23*(1), 24–29.

Baker, J. K. (1975/1990). Stochastic modeling for automatic speech understanding. In Waibel, A. and Lee, K.-F. (Eds.), *Readings in Speech Recognition*, pp. 297–307. Morgan Kaufmann, Los Altos. Originally appeared in *Speech Recognition*, Academic Press, 1975.

Bates, R. (1997). The corrections officer: Can John Kidd save Ulysses. *Lingua Franca*, *7*(8). October.

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In Shisha, O. (Ed.), *Inequalities III: Proceedings of the Third Symposium on Inequalities*, University of California, Los Angeles, pp. 1–8. Academic Press.

Bellegarda, J. R. (2000). Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE*, *89*(8), 1279–1296.

Bellegarda, J. R. (1999). Speech recognition experiments using multi-span statistical language models. In *IEEE ICASSP-99*, pp. 717–720.

Berger, A. and Miller, R. (1998). Just-in-time language modeling. In *IEEE ICASSP-98*, Vol. II, pp. 705–708.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., Lai, J. C., and Mercer, R. L. (1992a). An estimate of an upper bound for the entropy of English. *Computational Linguistics*, *18*(1), 31–40.

Brown, P. F., Della Pietra, V. J., de Souza, P. V., Lai, J. C., and Mercer, R. L. (1992b). Class-based *n*-gram models of natural language. *Computational Linguistics*, *18*(4), 467–479.

Bulyko, I., Ostendorf, M., and Stolcke, A. (2003). Getting more mileage from web text sources for conversational speech language modeling using class-dependent mixtures. In *HLT-NAACL-03*, Edmonton, Canada, Vol. 2, pp. 7–9.

Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *ACL-96*, Santa Cruz, CA, pp. 310–318.

Chen, S. F. and Goodman, J. (1998). An empirical study of smoothing techniques for language modeling. Tech. rep. TR-10-98, Computer Science Group, Harvard University.

Chen, S. F. and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, *13*(359–394).

Chen, S. F., Seymore, K., and Rosenfeld, R. (1998). Topic adaptation for language modeling using unnormalized exponential models. In *IEEE ICASSP-98*, pp. 681–684. IEEE.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, *2*(3), 113–124.

Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague.

Chomsky, N. (1969). Quine's empirical assumptions. In Davidson, D. and Hintikka, J. (Eds.), *Words and objections. Essays on the work of W. V. Quine*, pp. 53–68. D. Reidel, Dordrecht.

Church, K., Hart, T., and Gao, J. (2007). Compressing trigram language models with Golomb coding. In *EMNLP/CoNLL 2007*, pp. 199–207.

Church, K. W. and Gale, W. A. (1991). A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, *5*, 19–54.

Church, K. W., Gale, W. A., and Kruskal, J. B. (1991). Appendix A: the Good-Turing theorem. In *Computer Speech and Language* (Church and Gale, 1991), pp. 19–54.

Clark, H. H. and Fox Tree, J. E. (2002). Using uh and um in spontaneous speaking. *Cognition*, *84*, 73–111.

Clarkson, P. R. and Rosenfeld, R. (1997). Statistical language modeling using the CMU-Cambridge toolkit. In *EUROSPEECH-97*, Vol. 1, pp. 2707–2710.

Coccaro, N. and Jurafsky, D. (1998). Towards better integration of semantic predictors in statistical language modeling. In *ICSLP-98*, Sydney, Vol. 6, pp. 2403–2406.

Cover, T. M. and King, R. C. (1978). A convergent gambling estimate of the entropy of English. *IEEE Transactions on Information Theory*, *24*(4), 413–421.

Cover, T. M. and Thomas, J. A. (1991). *Elements of information theory*. Wiley.

Demetriou, G., Atwell, E., and Souter, C. (1997). Large-scale lexical semantics for speech recognition support. In *EUROSPEECH-97*, pp. 2755–2758.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the *EM* algorithm. *Journal of the Royal Statistical Society*, *39*(1), 1–21.

Foster, D. W. (1989). *Elegy by W.S.: A Study in Attribution*. Associated University Presses, Cranbury, NJ.

Foster, D. W. (1996). Primary culprit. *New York*, *29*(8), 50–57. February 26.

Francis, W. N. (1979). A tagged corpus – problems and prospects. In Greenbaum, S., Leech, G., and Svartvik, J. (Eds.), *Studies in English linguistics for Randolph Quirk*, pp. 192–209. Longman.

Francis, W. N. and Kučera, H. (1982). *Frequency Analysis of English Usage*. Houghton Mifflin, Boston.

Franz, A. and Brants, T. (2006). All our n-gram are belong to you. `http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html`.

Gale, W. A. and Church, K. W. (1990). Estimation procedures for language context: poor estimates are worse than none. In *COMPSTAT: Proceedings in Computational Statistics*, pp. 69–74.

Gale, W. A. and Church, K. W. (1994). What is wrong with adding one?. In Oostdijk, N. and de Haan, P. (Eds.), *Corpus-based Research into Language*, pp. 189–198. Rodopi, Amsterdam.

Gale, W. A. and Sampson, G. (1995). Good-turing frequency estimation without tears. *Journal of Quantitative Linguistics*, *2*, 217–237.

Genzel, D. and Charniak, E. (2002). Entropy rate constancy in text. In *ACL-02*.

Genzel, D. and Charniak, E. (2003). Variation of entropy and parse trees of sentences as a function of the sentence number. In *EMNLP 2003*.

Gildea, D. and Hofmann, T. (1999). Topic-based language models using EM. In *EUROSPEECH-99*, Budapest, pp. 2167–2170. http://www.cis.upenn.edu/ dgildea/gildea_hofmann_99.ps.

Godfrey, J., Holliman, E., and McDaniel, J. (1992). SWITCHBOARD: Telephone speech corpus for research and development. In *IEEE ICASSP-92*, San Francisco, pp. 517–520. IEEE.

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, *40*, 16–264.

Goodman, J. (2006). A bit of progress in language modeling: Extended version. Tech. rep. MSR-TR-2001-72, Machine Learning and Applied Statistics Group, Microsoft Research, Redmond, WA.

Gupta, V., Lennig, M., and Mermelstein, P. (1992). A language model for very large-vocabulary speech recognition. *Computer Speech and Language*, *6*, 331–344.

Heeman, P. A. (1999). POS tags and decision trees for language modeling. In *EMNLP/VLC-99*, College Park, MD, pp. 129–137.

Holmes, D. I. (1994). Authorship attribution. *Computers and the Humanities*, *28*, 87–106.

Iyer, R. M. and Ostendorf, M. (1999a). Modeling long distance dependencies in language: Topic mixtures versus dynamic cache model. *IEEE Transactions on Speech and Audio Processing*, *7*.

Iyer, R. M. and Ostendorf, M. (1999b). Relevance weighting for combining multi-domain data for n-gram language modeling. *Computer Speech and Language*, *13*(3), 267–282.

Iyer, R. M. and Ostendorf, M. (1997). Transforming out-of-domain estimates to improve in-domain language models. In *EUROSPEECH-97*, pp. 1975–1978.

Jeffreys, H. (1948). *Theory of Probability*. Clarendon Press, Oxford. 2nd edn Section 3.23.

Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, *64*(4), 532–557.

Jelinek, F. (1988). Address to the first workshop on the evaluation of natural language processing systems. December 7, 1988.

Jelinek, F. (1990). Self-organized language modeling for speech recognition. In Waibel, A. and Lee, K.-F. (Eds.), *Readings in Speech Recognition*, pp. 450–506. Morgan Kaufmann, Los Altos. Originally distributed as IBM technical report in 1985.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In Gelsema, E. S. and Kanal, L. N. (Eds.), *Proceedings, Workshop on Pattern Recognition in Practice*, pp. 381–397. North Holland, Amsterdam.

Johnson, W. E. (1932). Probability: deductive and inductive problems (appendix to). *Mind*, *41*(164), 421–423.

Jurafsky, D., Bell, A., Gregory, M. L., and Raymond, W. D. (2001). Probabilistic relations between words: Evidence from reduction in lexical production. In Bybee, J. L. and Hopper, P. (Eds.), *Frequency and the Emergence of Linguistic Structure*, pp. 229–254. Benjamins, Amsterdam.

Jurafsky, D., Wooters, C., Tajchman, G., Segal, J., Stolcke, A., Fosler, E., and Morgan, N. (1994). The Berkeley restaurant project. In *ICSLP-94*, Yokohama, Japan, pp. 2139–2142.

Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *35*(3), 400–401.

Keller, F. (2004). The entropy rate principle as a predictor of processing effort: An evaluation against eye-tracking data. In *EMNLP 2004*, Barcelona, pp. 317–324.

Keller, F. and Lapata, M. (2003). Using the web to obtain frequencies for unseen bigrams. *Computational Linguistics*, *29*, 459–484.

Kneser, R. (1996). Statistical language modeling using a variable context length. In *ICSLP-96*, Philadelphia, PA, Vol. 1, pp. 494–497.

Kneser, R. and Ney, H. (1993). Improved clustering techniques for class-based statistical language modelling. In *EUROSPEECH-93*, pp. 973–976.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *IEEE ICASSP-95*, Vol. 1, pp. 181–184.

Kuhn, R. and De Mori, R. (1990). A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *12*(6), 570–583.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, *24*(4), 377–439.

Kučera, H. (1992). The mathematics of language. In *The American Heritage Dictionary of the English Language*, pp. xxxi–xxxiii. Houghton Mifflin, Boston.

Kučera, H. and Francis, W. N. (1967). *Computational analysis of present-day American English*. Brown University Press, Providence, RI.

LDC (1993). *LDC Catalog: CSR-I (WSJ0) Complete*. University of Pennsylvania. `www.ldc.upenn.edu/Catalog/LDC93S6A.html`.

Leech, G., Garside, R., and Bryant, M. (1994). CLAWS4: The tagging of the British National Corpus. In *COLING-94*, Kyoto, pp. 622–628.

Lidstone, G. J. (1920). Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities. *Transactions of the Faculty of Actuaries*, *8*, 182–192.

Lindblom, B. E. F. (1990). Explaining phonetic variation: A sketch of the H&H theory. In Hardcastle, W. J. and Marchal, A. (Eds.), *Speech Production and Speech Modelling*, pp. 403–439. Kluwer.

Markov, A. A. (1913). Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain'). *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, *7*, 153–162. English translation by Morris Halle, 1956.

Miller, G. A. and Chomsky, N. (1963). Finitary models of language users. In Luce, R. D., Bush, R. R., and Galanter, E. (Eds.), *Handbook of Mathematical Psychology*, Vol. II, pp. 419–491. John Wiley.

Miller, G. A. and Selfridge, J. A. (1950). Verbal context and the recall of meaningful material. *American Journal of Psychology*, *63*, 176–185.

Mosteller, F. and Wallace, D. L. (1964). *Inference and Disputed Authorship: The Federalist*. Springer-Verlag. A second edition appeared in 1984 as *Applied Bayesian and Classical Inference*.

Nádas, A. (1984). Estimation of probabilities in the language model of the IBM speech recognition system. *IEEE Transactions on Acoustics, Speech, Signal Processing*, *32*(4), 859–861.

Nakov, P. I. and Hearst, M. A. (2005). A study of using search engine page hits as a proxy for n-gram frequencies. In *Proceedings of RANLP-05 (Recent Advances in Natural Language Processing)*, Borovets, Bulgaria.

Newell, A., Langer, S., and Hickey, M. (1998). The rôle of natural language processing in alternative and augmentative communication. *Natural Language Engineering*, *4*(1), 1–16.

Ney, H., Essen, U., and Kneser, R. (1994). On structuring probabilistic dependencies in stochastic language modelling. *Computer Speech and Language*, *8*, 1–38.

Niesler, T. R., Whittaker, E. W. D., and Woodland, P. C. (1998). Comparison of part-of-speech and automatically derived category-based language models for speech recognition. In *IEEE ICASSP-98*, Vol. 1, pp. 177–180.

Niesler, T. R. and Woodland, P. C. (1996). A variable-length category-based n-gram language model. In *IEEE ICASSP-96*, Atlanta, GA, Vol. I, pp. 164–167. IEEE.

Niesler, T. R. and Woodland, P. C. (1999). Modelling word-pair relations in a category-based language model. In *IEEE ICASSP-99*, pp. 795–798. IEEE.

Palmer, M. and Finin, T. (1990). Workshop on the evaluation of natural language processing systems. *Computational Linguistics*, *16*(3), 175–181.

Plotkin, J. B. and Nowak, M. A. (2000). Language evolution and information theory. *Journal of Theoretical Biology*, *205*(1), 147–159.

Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, *10*, 187–228.

Russell, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall. Second edition.

Sampson, G. (1996). *Evolutionary Language Understanding*. Cassell, London.

Samuelsson, C. and Reichl, W. (1999). A class-based language model for large-vocabulary speech recognition extracted from part-of-speech statistics. In *IEEE ICASSP-99*, pp. 537–540. IEEE.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, *27*(3), 379–423. Continued in the following volume.

Shannon, C. E. (1951). Prediction and entropy of printed English. *Bell System Technical Journal*, *30*, 50–64.

Sparck Jones, K. and Galliers, J. R. (Eds.). (1996). *Evaluating Natural Language Processing Systems*. Springer.

Stolcke, A. (1998). Entropy-based pruning of backoff language models. In *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, Lansdowne, VA, pp. 270–274.

Stolcke, A. (2002). Srilm - an extensible language modeling toolkit. In *ICSLP-02*, Denver, CO.

Stolcke, A. and Shriberg, E. (1996). Statistical language modeling for speech disfluencies. In *IEEE ICASSP-96*, Atlanta, GA, Vol. 1, pp. 405–408. IEEE.

Van Son, R. J. J. H., Koopmans-van Beinum, F. J., and Pols, L. C. W. (1998). Efficiency as an organizing principle of natural speech. In *ICSLP-98*, Sydney.

Van Son, R. J. J. H. and Pols, L. C. W. (2003). How efficient is speech?. *Proceedings of the Institute of Phonetic Sciences*, *25*, 171–184.

Witten, I. H. and Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, *37*(4), 1085–1094.

Zhou, G. and Lua, K. (1998). Word association and MI-trigger-based language modelling. In *COLING/ACL-98*, Montreal, Canada, pp. 1465–1471.

Zhu, X. and Rosenfeld, R. (2001). Improving trigram language modeling with the world wide web. In *IEEE ICASSP-01*, Salt Lake City, UT, Vol. I, pp. 533–536.

5 WORD CLASSES AND PART-OF-SPEECH TAGGING

Conjunction Junction, what's your function?

Bob Dorough, *Schoolhouse Rock, 1973*

> *A gnostic was seated before a grammarian. The grammarian said, 'A word must be one of three things: either it is a noun, a verb, or a particle.' The gnostic tore his robe and cried, "Alas! Twenty years of my life and striving and seeking have gone to the winds, for I laboured greatly in the hope that there was another word outside of this. Now you have destroyed my hope.' Though the gnostic had already attained the word which was his purpose, he spoke thus in order to arouse the grammarian.*
>
> Rumi (1207–1273), *The Discourses of Rumi*, Translated by A. J. Arberry

PARTS-OF-SPEECH

Dionysius Thrax of Alexandria (*c.* 100 B.C.), or perhaps someone else (exact authorship being understandably difficult to be sure of with texts of this vintage), wrote a grammatical sketch of Greek (a "*technē*") which summarized the linguistic knowledge of his day. This work is the direct source of an astonishing proportion of our modern linguistic vocabulary, including among many other words, *syntax*, *diphthong*, *clitic*, and *analogy*. Also included are a description of eight **parts-of-speech**: noun, verb, pronoun, preposition, adverb, conjunction, participle, and article. Although earlier scholars (including Aristotle as well as the Stoics) had their own lists of parts-of-speech, it was Thrax's set of eight which became the basis for practically all subsequent part-of-speech descriptions of Greek, Latin, and most European languages for the next 2000 years.

Schoolhouse Rock was a popular series of 3-minute musical animated clips first aired on television in 1973. The series was designed to inspire kids to learn multiplication tables, grammar, and basic science and history. The Grammar Rock sequence, for example, included songs about parts-of-speech, thus bringing these categories into the realm of popular culture. As it happens, Grammar Rock was remarkably traditional in its grammatical notation, including exactly eight songs about parts-of-speech.

Although the list was slightly modified from Thrax's original, substituting adjective and interjection for the original participle and article, the astonishing durability of the parts-of-speech through two millenia is an indicator of both the importance and the transparency of their role in human language.

TAGSETS

More recent lists of parts-of-speech (or **tagsets**) have many more word classes; 45 for the Penn Treebank (Marcus et al., 1993), 87 for the Brown corpus (Francis, 1979; Francis and Kučera, 1982), and 146 for the C7 tagset (Garside et al., 1997).

POS

The significance of parts-of-speech (also known as **POS**, **word classes**, **morphological classes**, or **lexical tags**) for language processing is the large amount of information they give about a word and its neighbors. This is clearly true for major categories, (**verb** versus **noun**), but is also true for the many finer distinctions. For example these tagsets distinguish between possessive pronouns (*my*, *your*, *his*, *her*, *its*) and personal pronouns (*I*, *you*, *he*, *me*). Knowing whether a word is a possessive pronoun or a personal pronoun can tell us what words are likely to occur in its vicinity (possessive pronouns are likely to be followed by a noun, personal pronouns by a verb). This can be useful in a language model for speech recognition.

A word's part-of-speech can tell us something about how the word is pronounced. As Ch. 8 will discuss, the word *content*, for example, can be a noun or an adjective. They are pronounced differently (the noun is pronounced *CONtent* and the adjective *conTENT*). Thus knowing the part-of-speech can produce more natural pronunciations in a speech synthesis system and more accuracy in a speech recognition system. (Other pairs like this include *OBject* (noun) and *obJECT* (verb), *DIScount* (noun) and *disCOUNT* (verb); see Cutler (1986)).

Parts-of-speech can also be used in stemming for informational retrieval (IR), since knowing a word's part-of-speech can help tell us which morphological affixes it can take, as we saw in Ch. 3. They can also enhance an IR application by selecting out nouns or other important words from a document. Automatic assignment of part-of-speech plays a role in parsing, in word-sense disambiguation algorithms, and in shallow parsing of texts to quickly find names, times, dates, or other named entities for the information extraction applications discussed in Ch. 22. Finally, corpora that have been marked for parts-of-speech are very useful for linguistic research. For example, they can be used to help find instances or frequencies of particular constructions.

This chapter focuses on computational methods for assigning parts-of-speech to words (**part-of-speech tagging**). Many algorithms have been applied to this problem, including hand-written rules (**rule-based tagging**), probabilistic methods (**HMM tagging** and **maximum entropy tagging**), as well as other methods such as **transformation-based tagging** and **memory-based tagging**. We will introduce three of these algorithms in this chapter: rule-based tagging, HMM tagging, and transformation-based tagging. But before turning to the algorithms themselves, let's begin with a summary of English word classes, and of various tagsets for formally coding these classes.

5.1 (Mostly) English Word Classes

Until now we have been using part-of-speech terms like **noun** and **verb** rather freely. In this section we give a more complete definition of these and other classes. Traditionally the definition of parts-of-speech has been based on syntactic and morphological function; words that function similarly with respect to what can occur nearby (their “syntactic distributional properties”), or with respect to the affixes they take (their morphological properties) are grouped into classes. While word classes do have tendencies toward semantic coherence (nouns do in fact often describe “people, places or things”, and adjectives often describe properties), this is not necessarily the case, and in general we don’t use semantic coherence as a definitional criterion for parts-of-speech.

CLOSED CLASS
OPEN CLASS

Parts-of-speech can be divided into two broad supercategories: **closed class** types and **open class** types. Closed classes are those that have relatively fixed membership. For example, prepositions are a closed class because there is a fixed set of them in English; new prepositions are rarely coined. By contrast nouns and verbs are open classes because new nouns and verbs are continually coined or borrowed from other languages (e.g., the new verb *to fax* or the borrowed noun *futon*). It is likely that any given speaker or corpus will have different open class words, but all speakers of a language, and corpora that are large enough, will likely share the set of closed class words. Closed class words are also generally **function words** like *of*, *it*, *and*, or *you*, which tend to be very short, occur frequently, and often have structuring uses in grammar.

FUNCTION WORDS

There are four major open classes that occur in the languages of the world; **nouns**, **verbs**, **adjectives**, and **adverbs**. It turns out that English has all four of these, although not every language does.

NOUN

Noun is the name given to the syntactic class in which the words for most people, places, or things occur. But since syntactic classes like **noun** are defined syntactically and morphologically rather than semantically, some words for people, places, and things may not be nouns, and conversely some nouns may not be words for people, places, or things. Thus nouns include concrete terms like *ship* and *chair*, abstractions like *bandwidth* and *relationship*, and verb-like terms like *pacing* as in *His pacing to and fro became quite annoying*. What defines a noun in English, then, are things like its ability to occur with determiners (*a goat*, *its bandwidth*, *Plato’s Republic*), to take possessives (*IBM’s annual revenue*), and for most but not all nouns, to occur in the plural form (*goats*, *abaci*).

PROPER NOUNS
COMMON NOUNS

Nouns are traditionally grouped into **proper nouns** and **common nouns**. Proper nouns, like *Regina*, *Colorado*, and *IBM*, are names of specific persons or entities. In English, they generally aren’t preceded by articles (e.g., *the book is upstairs*, but *Regina is upstairs*). In written English, proper nouns are usually capitalized.

COUNT NOUNS
MASS NOUNS

In many languages, including English, common nouns are divided into **count nouns** and **mass nouns**. Count nouns are those that allow grammatical enumeration; that is, they can occur in both the singular and plural (*goat/goats*, *relationship/relationships*) and they can be counted (*one goat*, *two goats*). Mass nouns are used when something is conceptualized as a homogeneous group. So words like *snow*, *salt*, and *communism* are not counted (i.e., **two snows* or **two communisms*). Mass nouns can also appear without articles where singular count nouns cannot (*Snow is white* but not **Goat is* *white*).

VERB

The **verb** class includes most of the words referring to actions and processes, including main verbs like *draw*, *provide*, *differ*, and *go*. As we saw in Ch. 3, English verbs have a number of morphological forms (non-3rd-person-sg (*eat*), 3rd-person-sg (*eats*), progressive (*eating*), past participle (*eaten*)). A subclass of English verbs called **auxiliaries** will be discussed when we turn to closed class forms.

AUXILIARIES

While many researchers believe that all human languages have the categories of noun and verb, others have argued that some languages, such as Riau Indonesian and Tongan, don't even make this distinction (Broschart, 1997; Evans, 2000; Gil, 2000).

The third open class English form is adjectives; semantically this class includes many terms that describe properties or qualities. Most languages have adjectives for the concepts of color (*white*, *black*), age (*old*, *young*), and value (*good*, *bad*), but there are languages without adjectives. In Korean, for example, the words corresponding to English adjectives act as a subclass of verbs, so what is in English an adjective 'beautiful' acts in Korean like a verb meaning 'to be beautiful' (Evans, 2000).

ADVERBS

The final open class form, **adverbs**, is rather a hodge-podge, both semantically and formally. For example Schachter (1985) points out that in a sentence like the following, all the italicized words are adverbs:

Unfortunately, John walked *home extremely slowly yesterday*

LOCATIVE
DEGREE
MANNER
TEMPORAL

What coherence the class has semantically may be solely that each of these words can be viewed as modifying something (often verbs, hence the name "adverb", but also other adverbs and entire verb phrases). **Directional adverbs** or **locative adverbs** (*home*, *here*, *downhill*) specify the direction or location of some action; **degree adverbs** (*extremely*, *very*, *somewhat*) specify the extent of some action, process, or property; **manner adverbs** (*slowly*, *slinkily*, *delicately*) describe the manner of some action or process; and **temporal adverb** describe the time that some action or event took place (*yesterday*, *Monday*). Because of the heterogeneous nature of this class, some adverbs (for example temporal adverbs like *Monday*) are tagged in some tagging schemes as nouns.

The closed classes differ more from language to language than do the open classes. Here's a quick overview of some of the more important closed classes in English, with a few examples of each:

- **prepositions:** on, under, over, near, by, at, from, to, with
- **determiners:** a, an, the
- **pronouns:** she, who, I, others
- **conjunctions:** and, but, or, as, if, when
- **auxiliary verbs:** can, may, should, are
- **particles:** up, down, on, off, in, out, at, by,
- **numerals:** one, two, three, first, second, third

PREPOSITIONS

Prepositions occur before noun phrases; semantically they are relational, often indicating spatial or temporal relations, whether literal (*on it*, *before then*, *by the house*) or metaphorical (*on time*, *with gusto*, *beside herself*). But they often indicate other relations as well (*Hamlet was written by Shakespeare*, and [from Shakespeare] "*And I did laugh sans intermission an hour by his dial*"). Fig. 5.1 shows the prepositions of

English according to the CELEX on-line dictionary (Baayen et al., 1995), sorted by their frequency in the COBUILD 16 million word corpus of English. Fig. 5.1 should not be considered a definitive list, since different dictionaries and tagsets label word classes differently. Furthermore, this list combines prepositions and particles.

of	540,085	through	14,964	worth	1,563	pace	12
in	331,235	after	13,670	toward	1,390	nigh	9
for	142,421	between	13,275	plus	750	re	4
to	125,691	under	9,525	till	686	mid	3
with	124,965	per	6,515	amongst	525	o’er	2
on	109,129	among	5,090	via	351	but	0
at	100,169	within	5,030	amid	222	ere	0
by	77,794	towards	4,700	underneath	164	less	0
from	74,843	above	3,056	versus	113	midst	0
about	38,428	near	2,026	amidst	67	o’	0
than	20,210	off	1,695	sans	20	thru	0
over	18,071	past	1,575	circa	14	vice	0

Figure 5.1 Prepositions (and particles) of English from the CELEX on-line dictionary. Frequency counts are from the COBUILD 16 million word corpus.

PARTICLE

PHRASAL VERB

A **particle** is a word that resembles a preposition or an adverb, and is used in combination with a verb. When a verb and a particle behave as a single syntactic and/or semantic unit, we call the combination a **phrasal verb**. Phrasal verbs can behave as a semantic unit; thus they often have a meaning that is not predictable from the separate meanings of the verb and the particle. Thus *turn down* means something like ‘reject’, *rule out* means ‘eliminate’, *find out* is ‘discover’, and *go on* is ‘continue’; these are not meanings that could have been predicted from the meanings of the verb and the particle independently. Here are some examples of phrasal verbs from Thoreau:

> So I *went on* for some days cutting and hewing timber. . .
> Moral reform is the effort to *throw off* sleep. . .

Particles don’t always occur with idiomatic phrasal verb semantics; here are more examples of particles from the Brown corpus:

> . . . she had turned the paper *over*.
> He arose slowly and brushed himself *off*.
> He packed *up* his clothes.

We show in Fig. 5.2 a list of single-word particles from Quirk et al. (1985). Since it is extremely hard to automatically distinguish particles from prepositions, some tagsets (like the one used for CELEX) do not distinguish them, and even in corpora that do (like the Penn Treebank) the distinction is very difficult to make reliably in an automatic process, so we do not give counts.

DETERMINERS

ARTICLES

A closed class that occurs with nouns, often marking the beginning of a noun phrase, is the **determiners**. One small subtype of determiners is the **articles**: English has three articles: *a*, *an*, and *the*. Other determiners include *this* (as in *this chapter*) and *that* (as in *that page*). *A* and *an* mark a noun phrase as indefinite, while *the* can mark it

aboard	aside	besides	forward(s)	opposite	through
about	astray	between	home	out	throughout
above	away	beyond	in	outside	together
across	back	by	inside	over	under
ahead	before	close	instead	overhead	underneath
alongside	behind	down	near	past	up
apart	below	east, etc.	off	round	within
around	beneath	eastward(s),etc.	on	since	without

Figure 5.2 English single-word particles from Quirk et al. (1985).

as definite; definiteness is a discourse and semantic property that will be discussed in Ch. 21. Articles are quite frequent in English; indeed *the* is the most frequently occurring word in most corpora of written English. Here are COBUILD statistics, again out of 16 million words:

the: 1,071,676 a: 413,887 an: 59,359

CONJUNCTIONS

Conjunctions are used to join two phrases, clauses, or sentences. Coordinating conjunctions like *and*, *or*, and *but*, join two elements of equal status. Subordinating conjunctions are used when one of the elements is of some sort of embedded status. For example *that* in “*I thought that you might like some milk*” is a subordinating conjunction that links the main clause *I thought* with the subordinate clause *you might like some milk*. This clause is called subordinate because this entire clause is the “content” of the main verb *thought*. Subordinating conjunctions like *that* which link a verb to its argument in this way are also called **complementizers**. Ch. 12 and Ch. 16 will discuss complementation in more detail. Table 5.3 lists English conjunctions.

COMPLEMENTIZERS

and	514,946	yet	5,040	considering	174	forasmuch as	0
that	134,773	since	4,843	lest	131	however	0
but	96,889	where	3,952	albeit	104	immediately	0
or	76,563	nor	3,078	providing	96	in as far as	0
as	54,608	once	2,826	whereupon	85	in so far as	0
if	53,917	unless	2,205	seeing	63	inasmuch as	0
when	37,975	why	1,333	directly	26	insomuch as	0
because	23,626	now	1,290	ere	12	insomuch that	0
so	12,933	neither	1,120	notwithstanding	3	like	0
before	10,720	whenever	913	according as	0	neither nor	0
though	10,329	whereas	867	as if	0	now that	0
than	9,511	except	864	as long as	0	only	0
while	8,144	till	686	as though	0	provided that	0
after	7,042	provided	594	both and	0	providing that	0
whether	5,978	whilst	351	but that	0	seeing as	0
for	5,935	suppose	281	but then	0	seeing as how	0
although	5,424	cos	188	but then again	0	seeing that	0
until	5,072	supposing	185	either or	0	without	0

Figure 5.3 Coordinating and subordinating conjunctions of English from CELEX. Frequency counts are from COBUILD (16 million words).

PRONOUNS
PERSONAL
POSSESSIVE

Pronouns are forms that often act as a kind of shorthand for referring to some noun phrase or entity or event. **Personal pronouns** refer to persons or entities (*you*, *she*, *I*, *it*, *me*, etc.). **Possessive pronouns** are forms of personal pronouns that indicate either actual possession or more often just an abstract relation between the person and some object (*my, your, his, her, its, one's, our, their*). WH **Wh-pronouns** (*what, who, whom, whoever*) are used in certain question forms, or may also act as complementizers (*Frieda, who I met five years ago ...*). Table 5.4 shows English pronouns, again from CELEX.

it	199,920	how	13,137	yourself	2,437	no one	106
I	198,139	another	12,551	why	2,220	wherein	58
he	158,366	where	11,857	little	2,089	double	39
you	128,688	same	11,841	none	1,992	thine	30
his	99,820	something	11,754	nobody	1,684	summat	22
they	88,416	each	11,320	further	1,666	suchlike	18
this	84,927	both	10,930	everybody	1,474	fewest	15
that	82,603	last	10,816	ourselves	1,428	thyself	14
she	73,966	every	9,788	mine	1,426	whomever	11
her	69,004	himself	9,113	somebody	1,322	whosoever	10
we	64,846	nothing	9,026	former	1,177	whomsoever	8
all	61,767	when	8,336	past	984	wherefore	6
which	61,399	one	7,423	plenty	940	whereat	5
their	51,922	much	7,237	either	848	whatsoever	4
what	50,116	anything	6,937	yours	826	whereon	2
my	46,791	next	6,047	neither	618	whoso	2
him	45,024	themselves	5,990	fewer	536	aught	1
me	43,071	most	5,115	hers	482	howsoever	1
who	42,881	itself	5,032	ours	458	thrice	1
them	42,099	myself	4,819	whoever	391	wheresoever	1
no	33,458	everything	4,662	least	386	you-all	1
some	32,863	several	4,306	twice	382	additional	0
other	29,391	less	4,278	theirs	303	anybody	0
your	28,923	herself	4,016	wherever	289	each other	0
its	27,783	whose	4,005	oneself	239	once	0
our	23,029	someone	3,755	thou	229	one another	0
these	22,697	certain	3,345	'un	227	overmuch	0
any	22,666	anyone	3,318	ye	192	such and such	0
more	21,873	whom	3,229	thy	191	whate'er	0
many	17,343	enough	3,197	whereby	176	whenever	0
such	16,880	half	3,065	thee	166	whereof	0
those	15,819	few	2,933	yourselves	148	whereto	0
own	15,741	everyone	2,812	latter	142	whereunto	0
us	15,724	whatever	2,571	whichever	121	whichsoever	0

Figure 5.4 Pronouns of English from the CELEX on-line dictionary. Frequency counts are from the COBUILD 16 million word corpus.

AUXILIARY A closed class subtype of English verbs are the **auxiliary** verbs. Crosslinguistically, auxiliaries are words (usually verbs) that mark certain semantic features of a main verb, including whether an action takes place in the present, past or future (tense), whether it is completed (aspect), whether it is negated (polarity), and whether an action is necessary, possible, suggested, desired, etc. (mood).

COPULA English auxiliaries include the **copula** verb *be*, the two verbs *do* and *have*, along MODAL with their inflected forms, as well as a class of **modal verbs**. *Be* is called a copula because it connects subjects with certain kinds of predicate nominals and adjectives (*He is a duck*). The verb *have* is used for example to mark the perfect tenses (*I have gone, I had gone*), while *be* is used as part of the passive (*We were robbed*), or progressive (*We are leaving*) constructions. The modals are used to mark the mood associated with the event or action depicted by the main verb. So *can* indicates ability or possibility, *may* indicates permission or possibility, *must* indicates necessity, and so on. Fig. 5.5 gives counts for the frequencies of the modals in English. In addition to the perfect *have* mentioned above, there is a modal verb *have* (e.g., *I have to go*), which is very common in spoken English. Neither it nor the modal verb *dare*, which is very rare, have frequency counts because the CELEX dictionary does not distinguish the main verb sense (*I have three oranges*, *He dared me to eat them*), from the modal sense (*There has to be some mistake*, *Dare I confront him?*), from the non-modal auxiliary verb sense (*I have never seen that*).

can	70,930	might	5,580	shouldn't	858
will	69,206	couldn't	4,265	mustn't	332
may	25,802	shall	4,118	'll	175
would	18,448	wouldn't	3,548	needn't	148
should	17,760	won't	3,100	mightn't	68
must	16,520	'd	2,299	oughtn't	44
need	9,955	ought	1,845	mayn't	3
can't	6,375	will	862	dare, have	???

Figure 5.5 English modal verbs from the CELEX on-line dictionary. Frequency counts are from the COBUILD 16 million word corpus.

INTERJECTIONS
NEGATIVES
POLITENESS MARKERS

English also has many words of more or less unique function, including **interjections** (*oh*, *ah*, *hey*, *man*, *alas*, *uh*, *um*), **negatives** (*no*, *not*), **politeness markers** (*please*, *thank you*), **greetings** (*hello*, *goodbye*), and the existential **there** (*there are two on the table*) among others. Whether these classes are assigned particular names or lumped together (as interjections or even adverbs) depends on the purpose of the labeling.

5.2 Tagsets for English

The previous section gave broad descriptions of the kinds of syntactic classes that English words fall into. This section fleshes out that sketch by describing the actual tagsets used in part-of-speech tagging, in preparation for the various tagging algorithms to be described in the following sections.

There are a small number of popular tagsets for English, many of which evolved from the 87-tag tagset used for the Brown corpus (Francis, 1979; Francis and Kučera, 1982). The Brown corpus is a 1 million word collection of samples from 500 written texts from different genres (newspaper, novels, non-fiction, academic, etc.) which was assembled at Brown University in 1963–1964 (Kučera and Francis, 1967; Francis, 1979; Francis and Kučera, 1982). This corpus was tagged with parts-of-speech by first applying the TAGGIT program and then hand-correcting the tags.

Besides this original Brown tagset, two of the most commonly used tagsets are the small 45-tag Penn Treebank tagset (Marcus et al., 1993), and the medium-sized 61 tag C5 tagset used by the Lancaster UCREL project's CLAWS (the Constituent Likelihood Automatic Word-tagging System) tagger to tag the British National Corpus (BNC) (Garside et al., 1997). We give all three of these tagsets here, focusing on the

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	*+,%, &*
CD	Cardinal number	*one, two, three*	TO	“to”	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential ‘there’	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	#
PDT	Predeterminer	*all, both*	“	Left quote	‘ or “
POS	Possessive ending	*’s*	”	Right quote	’ or ”
PRP	Personal pronoun	*I, you, he*	(	Left parenthesis	[, (, {, <
PRP$	Possessive pronoun	*your, one’s*	)	Right parenthesis	],), }, >
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	. ! ?
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	: ; ... – -
RP	Particle	*up, off*			

Figure 5.6 Penn Treebank part-of-speech tags (including punctuation).

smallest, the Penn Treebank set, and discuss difficult tagging decisions in that tag set and some useful distinctions made in the larger tagsets.

The Penn Treebank tagset, shown in Fig. 5.6, has been applied to the Brown corpus, the Wall Street Journal corpus, and the Switchboard corpus among others; indeed, perhaps partly because of its small size, it is one of the most widely used tagsets. Here are some examples of tagged sentences from the Penn Treebank version of the Brown corpus (we will represent a tagged word by placing the tag after each word, delimited by a slash):

(5.1) The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.

(5.2) **There/EX** are/VBP 70/CD children/NNS **there/RB**

(5.3) Although/IN preliminary/JJ findings/NNS were/VBD **reported/VBN** more/RBR than/IN a/DT year/NN ago/IN ,/, the/DT latest/JJS results/NNS appear/VBP in/IN today/NN **’s/POS** New/NNP England/NNP Journal/NNP of/IN Medicine/NNP ,/,

Example (5.1) shows phenomena that we discussed in the previous section; the determiners *the* and *a*, the adjectives *grand* and *other*, the common nouns *jury*, *number*, and *topics*, the past tense verb *commented*. Example (5.2) shows the use of the EX tag to mark the existential *there* construction in English, and, for comparison, another use of *there* which is tagged as an adverb (RB). Example (5.3) shows the segmentation of the possessive morpheme *’s*, and shows an example of a passive construction, 'were reported', in which the verb *reported* is marked as a past participle (VBN), rather than a simple past (VBD). Note also that the proper noun *New England* is tagged NNP. Finally, note that since *New England Journal of Medicine* is a proper noun, the Treebank tagging chooses to mark each noun in it separately as NNP, including *journal* and *medicine*, which might otherwise be labeled as common nouns (NN).

Some tagging distinctions are quite hard for both humans and machines to make. For example prepositions (IN), particles (RP), and adverbs (RB) can have a large overlap. Words like *around* can be all three:

(5.4) Mrs./NNP Shaefer/NNP never/RB got/VBD **around/RP** to/TO joining/VBG

(5.5) All/DT we/PRP gotta/VBN do/VB is/VBZ go/VB **around/IN** the/DT corner/NN

(5.6) Chateau/NNP Petrus/NNP costs/VBZ **around/RB** 250/CD

Making these decisions requires sophisticated knowledge of syntax; tagging manuals (Santorini, 1990) give various heuristics that can help human coders make these decisions, and that can also provide useful features for automatic taggers. For example two heuristics from Santorini (1990) are that prepositions generally are associated with a following noun phrase (although they also may be followed by prepositional phrases), and that the word *around* is tagged as an adverb when it means "approximately". Furthermore, particles often can either precede or follow a noun phrase object, as in the following examples:

(5.7) She told off/RP her friends

(5.8) She told her friends off/RP.

Prepositions, on the other hand, cannot follow their noun phrase (* is used here to mark an ungrammatical sentence, a concept which we will return to in Ch. 12):

(5.9) She stepped off/IN the train

(5.10) *She stepped the train off/IN.

Another difficulty is labeling the words that can modify nouns. Sometimes the modifiers preceding nouns are common nouns like *cotton* below, other times the Treebank tagging manual specifies that modifiers be tagged as adjectives (for example if the modifier is a hyphenated common noun like *income-tax*) and other times as proper nouns (for modifiers which are hyphenated proper nouns like *Gramm-Rudman*):

(5.11) cotton/NN sweater/NN

(5.12) income-tax/JJ return/NN

(5.13) the/DT Gramm-Rudman/NP Act/NP

Some words that can be adjectives, common nouns, or proper nouns, are tagged in the Treebank as common nouns when acting as modifiers:

(5.14) Chinese/NN cooking/NN

(5.15) Pacific/NN waters/NNS

A third known difficulty in tagging is distinguishing past participles (VBN) from adjectives (JJ). A word like *married* is a past participle when it is being used in an eventive, verbal way, as in (5.16) below, and is an adjective when it is being used to express a property, as in (5.17):

(5.16) They were married/VBN by the Justice of the Peace yesterday at 5:00.

(5.17) At the time, she was already married/JJ.

Tagging manuals like Santorini (1990) give various helpful criteria for deciding how ‘verb-like’ or ‘eventive’ a particular word is in a specific context.

The Penn Treebank tagset was culled from the original 87-tag tagset for the Brown corpus. This reduced set leaves out information that can be recovered from the identity of the lexical item. For example the original Brown and C5 tagsets include a separate tag for each of the different forms of the verbs *do* (e.g. C5 tag “VDD” for *did* and “VDG” for *doing*), *be*, and *have*. These were omitted from the Treebank set.

Certain syntactic distinctions were not marked in the Penn Treebank tagset because Treebank sentences were parsed, not merely tagged, and so some syntactic information is represented in the phrase structure. For example, the single tag IN is used for both prepositions and subordinating conjunctions since the tree-structure of the sentence disambiguates them (subordinating conjunctions always precede clauses, prepositions precede noun phrases or prepositional phrases). Most tagging situations, however, do not involve parsed corpora; for this reason the Penn Treebank set is not specific enough for many uses. The original Brown and C5 tagsets, for example, distinguish prepositions (IN) from subordinating conjunctions (CS), as in the following examples:

(5.18) **after/CS** spending/VBG a/AT few/AP days/NNS at/IN the/AT Brown/NP Palace/NN Hotel/NN

(5.19) **after/IN** a/AT wedding/NN trip/NN to/IN Corpus/NP Christi/NP ./.

The original Brown and C5 tagsets also have two tags for the word *to*; in Brown the infinitive use is tagged TO, while the prepositional use as IN:

(5.20) **to/TO** give/VB priority/NN **to/IN** teacher/NN pay/NN raises/NNS

Brown also has the tag NR for adverbial nouns like *home*, *west*, *Monday*, and *tomorrow*. Because the Treebank lacks this tag, it has a much less consistent policy for adverbial nouns; *Monday*, *Tuesday*, and other days of the week are marked NNP, *tomorrow*, *west*, and *home* are marked sometimes as NN, sometimes as RB. This makes the Treebank tagset less useful for high-level NLP tasks like the detection of time phrases.

Nonetheless, the Treebank tagset has been the most widely used in evaluating tagging algorithms, and so many of the algorithms we describe below have been evaluated mainly on this tagset. Of course whether a tagset is useful for a particular application depends on how much information the application needs.

5.3 PART-OF-SPEECH TAGGING

TAGGING

Part-of-speech tagging (or just **tagging** for short) is the process of assigning a part-of-speech or other syntactic class marker to each word in a corpus. Because tags are generally also applied to punctuation, tagging requires that the punctuation marks (period, comma, etc) be separated off of the words. Thus **tokenization** of the sort described in Ch. 3 is usually performed before, or as part of, the tagging process, separating commas, quotation marks, etc., from words, and disambiguating end-of-sentence

Tag	Description	Example
(	opening parenthesis	(, [
)	closing parenthesis	),]
*	negator	not n’t
,	comma	,
–	dash	–
.	sentence terminator	. ; ? !
:	colon	:
ABL	pre-qualifier	quite, rather, such
ABN	pre-quantifier	half, all,
ABX	pre-quantifier, double conjunction	both
AP	post-determiner	many, next, several, last
AT	article	a the an no a every
BE/BED/BEDZ/BEG/BEM/BEN/BER/BEZ		be/were/was/being/am/been/are/is
CC	coordinating conjunction	and or but either neither
CD	cardinal numeral	two, 2, 1962, million
CS	subordinating conjunction	that as after whether before
DO/DOD/DOZ		do, did, does
DT	singular determiner,	this, that
DTI	singular or plural determiner	some, any
DTS	plural determiner	these those them
DTX	determiner, double conjunction	either, neither
EX	existential there	there
HV/HVD/HVG/HVN/HVZ		have, had, having, had, has
IN	preposition	of in for by to on at
JJ	adjective	
JJR	comparative adjective	better, greater, higher, larger, lower
JJS	semantically superlative adj.	main, top, principal, chief, key, foremost
JJT	morphologically superlative adj.	best, greatest, highest, largest, latest, worst
MD	modal auxiliary	would, will, can, could, may, must, should
NN	(common) singular or mass noun	time, world, work, school, family, door
NN$	possessive singular common noun	father’s, year’s, city’s, earth’s
NNS	plural common noun	years, people, things, children, problems
NNS$	possessive plural noun	children’s, artist’s parent’s years’
NP	singular proper noun	Kennedy, England, Rachel, Congress
NP$	possessive singular proper noun	Plato’s Faulkner’s Viola’s
NPS	plural proper noun	Americans Democrats Belgians Chinese Sox
NPS$	possessive plural proper noun	Yankees’, Gershwins’ Earthmen’s
NR	adverbial noun	home, west, tomorrow, Friday, North,
NR$	possessive adverbial noun	today’s, yesterday’s, Sunday’s, South’s
NRS	plural adverbial noun	Sundays Fridays
OD	ordinal numeral	second, 2nd, twenty-first, mid-twentieth
PN	nominal pronoun	one, something, nothing, anyone, none,
PN$	possessive nominal pronoun	one’s someone’s anyone’s
PP$	possessive personal pronoun	his their her its my our your
PP$$	second possessive personal pronoun	mine, his, ours, yours, theirs
PPL	singular reflexive personal pronoun	myself, herself
PPLS	plural reflexive personal pronoun	ourselves, themselves
PPO	objective personal pronoun	me, us, him
PPS	3rd. sg. nominative pronoun	he, she, it
PPSS	other nominative pronoun	I, we, they
QL	qualifier	very, too, most, quite, almost, extremely
QLP	post-qualifier	enough, indeed
RB	adverb	
RBR	comparative adverb	later, more, better, longer, further
RBT	superlative adverb	best, most, highest, nearest
RN	nominal adverb	here, then

Figure 5.7 First part of original 87-tag Brown corpus tagset (Francis and Kučera, 1982).

Tag	Description	Example
RP	adverb or particle	across, off, up
TO	infinitive marker	to
UH	interjection, exclamation	well, oh, say, please, okay, uh, goodbye
VB	verb, base form	make, understand, try, determine, drop
VBD	verb, past tense	said, went, looked, brought, reached kept
VBG	verb, present participle, gerund	getting, writing, increasing
VBN	verb, past participle	made, given, found, called, required
VBZ	verb, 3rd singular present	says, follows, requires, transcends
WDT	wh- determiner	what, which
WP$	possessive wh- pronoun	whose
WPO	objective wh- pronoun	whom, which, that
WPS	nominative wh- pronoun	who, which, that
WQL	how	
WRB	wh- adverb	how, when

Figure 5.8 Rest of 87-tag Brown corpus tagset (Francis and Kučera, 1982).

punctuation (period, question mark, etc) from part-of-word punctuation (such as in abbreviations like *e.g.* and *etc.*)

The input to a tagging algorithm is a string of words and a specified tagset of the kind described in the previous section. The output is a single best tag for each word. For example, here are some sample sentences from the ATIS corpus of dialogues about air-travel reservations that we will discuss in Ch. 12. For each we have shown a potential tagged output using the Penn Treebank tagset defined in Fig. 5.6 on page 9:

(5.21) Book/VB that/DT flight/NN ./.

(5.22) Does/VBZ that/DT flight/NN serve/VB dinner/NN ?/.

The previous section discussed some tagging decisions that are difficult to make for humans. Even in these simple examples, automatically assigning a tag to each word is not trivial. For example, *book* is **ambiguous**. That is, it has more than one possible usage and part-of-speech. It can be a verb (as in *book that flight* or *to book the suspect*) or a noun (as in *hand me that book*, or *a book of matches*). Similarly *that* can be a determiner (as in *Does that flight serve dinner*), or a complementizer (as in *I thought that your flight was earlier*). The problem of POS-tagging is to **resolve** these ambiguities, choosing the proper tag for the context. Part-of-speech tagging is thus one of the many **disambiguation** tasks we will see in this book.

AMBIGUOUS

RESOLVE

DISAMBIGUATION

How hard is the tagging problem? The previous section described some difficult tagging decisions; how common is tag ambiguity? It turns out that most words in English are unambiguous; i.e., they have only a single tag. But many of the most common words of English are ambiguous (for example *can* can be an auxiliary (‘to be able’), a noun (‘a metal container’), or a verb (‘to put something in such a metal container’)). In fact, DeRose (1988) reports that while only 11.5% of English word types in the Brown corpus are ambiguous, over 40% of Brown tokens are ambiguous. Fig. 5.10 shows the number of word types with different levels of part-of-speech ambiguity from the Brown corpus. We show these computations from two versions of the tagged Brown corpus, the original tagging done at Brown by Francis and Kučera (1982), and the Treebank-3 tagging done at the University of Pennsylvania. Note that despite having more coarse-grained tags, the 45-tag corpus unexpectedly has more ambiguity than the 87-tag corpus.

Tag	Description	Example
AJ0	adjective (unmarked)	*good, old*
AJC	comparative adjective	*better, older*
AJS	superlative adjective	*best, oldest*
AT0	article	*the, a, an*
AV0	adverb (unmarked)	*often, well, longer, furthest*
AVP	adverb particle	*up, off, out*
AVQ	wh-adverb	*when, how, why*
CJC	coordinating conjunction	*and, or*
CJS	subordinating conjunction	*although, when*
CJT	the conjunction *that*	
CRD	cardinal numeral (except *one*)	*3, twenty-five, 734*
DPS	possessive determiner	*your, their*
DT0	general determiner	*these, some*
DTQ	wh-determiner	*whose, which*
EX0	existential *there*	
ITJ	interjection or other isolate	*oh, yes, mhm*
NN0	noun (neutral for number)	*aircraft, data*
NN1	singular noun	*pencil, goose*
NN2	plural noun	*pencils, geese*
NP0	proper noun	*London, Michael, Mars*
ORD	ordinal	*sixth, 77th, last*
PNI	indefinite pronoun	*none, everything*
PNP	personal pronoun	*you, them, ours*
PNQ	wh-pronoun	*who, whoever*
PNX	reflexive pronoun	*itself, ourselves*
POS	possessive *’s* or *’*	
PRF	the preposition *of*	
PRP	preposition (except *of*)	*for, above, to*
PUL	punctuation – left bracket	(or [
PUN	punctuation – general mark	. ! , : ; - ? ...
PUQ	punctuation – quotation mark	‘ ’ ”
PUR	punctuation – right bracket	) or]
TO0	infinitive marker *to*	
UNC	unclassified items (not English)	
VBB	base forms of *be* (except infinitive)	*am, are*
VBD	past form of *be*	*was, were*
VBG	-ing form of *be*	*being*
VBI	infinitive of *be*	
VBN	past participle of *be*	*been*
VBZ	-s form of *be*	*is, ’s*
VDB/D/G/I/N/Z	form of *do*	*do, does, did, doing, to do, etc.*
VHB/D/G/I/N/Z	form of *have*	*have, had, having, to have, etc.*
VM0	modal auxiliary verb	*can, could, will, ’ll*
VVB	base form of lexical verb (except infin.)	*take, live*
VVD	past tense form of lexical verb	*took, lived*
VVG	-ing form of lexical verb	*taking, living*
VVI	infinitive of lexical verb	*take, live*
VVN	past participle form of lex. verb	*taken, lived*
VVZ	-s form of lexical verb	*takes, lives*
XX0	the negative *not* or *n’t*	
ZZ0	alphabetical symbol	*A, B, c, d*

Figure 5.9 UCREL’s C5 tagset for the British National Corpus (Garside et al., 1997).

Luckily, it turns out that many of the 40% ambiguous tokens are easy to disambiguate. This is because the various tags associated with a word are not equally likely. For example, *a* can be a determiner, or the letter *a* (perhaps as part of an acronym or an

	Original 87-tag corpus		Treebank 45-tag corpus	
Unambiguous (1 tag)	**44,019**		**38,857**	
Ambiguous (2–7 tags)	**5,490**		**8844**	
Details: 2 tags	4,967		6,731	
3 tags	411		1621	
4 tags	91		357	
5 tags	17		90	
6 tags	2	(*well*, *beat*)	32	
7 tags	2	(*still*, *down*)	6	(*well*, *set*, *round*, *open*, *fit*, *down*)
8 tags			4	(*'s*, *half*, *back*, *a*)
9 tags			3	(*that*, *more*, *in*)

Figure 5.10 The amount of tag ambiguity for word types in the Brown corpus, from the ICAME release of the original (87-tag) tagging and the Treebank-3 (45-tag) tagging. Numbers are not strictly comparable because only the Treebank segments *'s*. An earlier estimate of some of these numbers is reported in DeRose (1988).

initial). But the determiner sense of *a* is much more likely.

RULE-BASED

STOCHASTIC TAGGERS

Most tagging algorithms fall into one of two classes: **rule-based** taggers and **stochastic taggers** taggers. Rule-based taggers generally involve a large database of hand-written disambiguation rules which specify, for example, that an ambiguous word is a noun rather than a verb if it follows a determiner. The next section will describe a sample rule-based tagger, **EngCG**, based on the Constraint Grammar architecture of Karlsson et al. (1995b).

HMM TAGGER

Stochastic taggers generally resolve tagging ambiguities by using a training corpus to compute the probability of a given word having a given tag in a given context. Sec. 5.5 describes the Hidden Markov Model or **HMM tagger**.

BRILL TAGGER

Finally, Sec. 5.6 will describe an approach to tagging called the **transformation-based tagger** or the **Brill tagger**, after Brill (1995). The Brill tagger shares features of both tagging architectures. Like the rule-based tagger, it is based on rules which determine when an ambiguous word should have a given tag. Like the stochastic taggers, it has a machine-learning component: the rules are automatically induced from a previously tagged training corpus.

5.4 Rule-Based Part-of-Speech Tagging

The earliest algorithms for automatically assigning part-of-speech were based on a two-stage architecture (Harris, 1962; Klein and Simmons, 1963; Greene and Rubin, 1971). The first stage used a dictionary to assign each word a list of potential parts-of-speech. The second stage used large lists of hand-written disambiguation rules to winnow down this list to a single part-of-speech for each word.

Modern rule-based approaches to part-of-speech tagging have a similar architecture, although the dictionaries and the rule sets are vastly larger than in the 1960's.

ENGCG

One of the most comprehensive rule-based approaches is the Constraint Grammar approach (Karlsson et al., 1995a). In this section we describe a tagger based on this approach, the **EngCG** tagger (Voutilainen, 1995, 1999).

The EngCG ENGTWOL lexicon is based on the two-level morphology described in Ch. 3, and has about 56,000 entries for English word stems (Heikkilä, 1995), counting a word with multiple parts-of-speech (e.g., nominal and verbal senses of *hit*) as separate entries, and not counting inflected and many derived forms. Each entry is annotated with a set of morphological and syntactic features. Fig. 5.11 shows some selected words, together with a slightly simplified listing of their features; these features are used in rule writing.

Word	**POS**	**Additional POS features**
smaller	ADJ	COMPARATIVE
entire	ADJ	ABSOLUTE ATTRIBUTIVE
fast	ADV	SUPERLATIVE
that	DET	CENTRAL DEMONSTRATIVE SG
all	DET	PREDETERMINER SG/PL QUANTIFIER
dog's	N	GENITIVE SG
furniture	N	NOMINATIVE SG NOINDEFDETERMINER
one-third	NUM	SG
she	PRON	PERSONAL FEMININE NOMINATIVE SG3
show	V	PRESENT -SG3 VFIN
show	N	NOMINATIVE SG
shown	PCP2	SVOO SVO SV
occurred	PCP2	SV
occurred	V	PAST VFIN SV

Figure 5.11 Sample lexical entries from the ENGTWOL lexicon described in Voutilainen (1995) and Heikkilä (1995).

Most of the features in Fig. 5.11 are relatively self-explanatory; SG for singular, -SG3 for other than third-person-singular. ABSOLUTE means non-comparative and non-superlative for an adjective, NOMINATIVE just means non-genitive, and PCP2 means past participle. PRE, CENTRAL, and POST are ordering slots for determiners (predeterminers (*all*) come before determiners (*the*): *all the president's men*). NOINDEFDETERMINER means that words like *furniture* do not appear with the indefinite determiner *a*. SV, SVO, and SVOO specify the **subcategorization** or **complementation** pattern for the verb. Subcategorization will be discussed in Ch. 12 and Ch. 16, but briefly SV means the verb appears solely with a subject (*nothing occurred*); SVO with a subject and an object (*I showed the film*); SVOO with a subject and two complements: *She showed her the ball*.

SUBCATEGORIZATION

COMPLEMENTATION

In the first stage of the tagger, each word is run through the two-level lexicon transducer and the entries for all possible parts-of-speech are returned. For example the phrase *Pavlov had shown that salivation ...* would return the following list (one line per possible tag, with the correct tag shown in boldface):

Pavlov	**PAVLOV N NOM SG PROPER**
had	**HAVE V PAST VFIN SVO**
	HAVE PCP2 SVO
shown	**SHOW PCP2 SVOO SVO SV**
that	ADV
	PRON DEM SG
	DET CENTRAL DEM SG
	CS
salivation	**N NOM SG**
. . .	

EngCG then applies a large set of constraints (as many as 3,744 constraints in the EngCG-2 system) to the input sentence to rule out incorrect parts-of-speech. The boldfaced entries in the table above show the desired result, in which the simple past tense tag (rather than the past participle tag) is applied to *had*, and the complementizer (CS) tag is applied to *that*. The constraints are used in a negative way, to eliminate tags that are inconsistent with the context. For example one constraint eliminates all readings of *that* except the ADV (adverbial intensifier) sense (this is the sense in the sentence *it isn't* that *odd*). Here's a simplified version of the constraint:

```
ADVERBIAL-THAT RULE
Given input: "that"
if
  (+1 A/ADV/QUANT); /* if next word is adj, adverb, or quantifier */
  (+2 SENT-LIM);       /* and following which is a sentence boundary, */
  (NOT -1 SVOC/A); /* and the previous word is not a verb like */
                  /* 'consider' which allows adjs as object complements */
then eliminate non-ADV tags
else eliminate ADV tag
```

The first two clauses of this rule check to see that the *that* directly precedes a sentence-final adjective, adverb, or quantifier. In all other cases the adverb reading is eliminated. The last clause eliminates cases preceded by verbs like *consider* or *believe* which can take a noun and an adjective; this is to avoid tagging the following instance of *that* as an adverb:

I consider that odd.

Another rule is used to express the constraint that the complementizer sense of *that* is most likely to be used if the previous word is a verb which expects a complement (like *believe*, *think*, or *show*), and if *that* is followed by the beginning of a noun phrase, and a finite verb.

This description oversimplifies the EngCG architecture; the system also includes probabilistic constraints, and also makes use of other syntactic information we haven't discussed. The interested reader should consult Karlsson et al. (1995b) and Voutilainen (1999).

5.5 HMM Part-of-Speech Tagging

The use of probabilities in tags is quite old; probabilities in tagging were first used by Stolz et al. (1965), a complete probabilistic tagger with Viterbi decoding was sketched by Bahl and Mercer (1976), and various stochastic taggers were built in the 1980s (Marshall, 1983; Garside, 1987; Church, 1988; DeRose, 1988). This section describes a particular stochastic tagging algorithm generally known as the Hidden Markov Model or HMM tagger. Hidden Markov Models themselves will be more fully introduced and defined in Ch. 6. In this section, we prefigure Ch. 6 a bit by giving an initial introduction to the Hidden Markov Model as applied to part-of-speech tagging.

BAYESIAN INFERENCE

Use of a Hidden Markov Model to do part-of-speech-tagging, as we will define it, is a special case of **Bayesian inference**, a paradigm that has been known since the work of Bayes (1763). Bayesian inference or Bayesian classification was applied successfully to language problems as early as the late 1950s, including the OCR work of Bledsoe in 1959, and the seminal work of Mosteller and Wallace (1964) on applying Bayesian inference to determine the authorship of the Federalist papers.

In a classification task, we are given some observation(s) and our job is to determine which of a set of classes it belongs to. Part-of-speech tagging is generally treated as a sequence classification task. So here the observation is a sequence of words (let's say a sentence), and it is our job to assign them a sequence of part-of-speech tags.

For example, say we are given a sentence like

(5.23) Secretariat is expected to **race** tomorrow.

What is the best sequence of tags which corresponds to this sequence of words? The Bayesian interpretation of this task starts by considering all possible sequences of classes—in this case, all possible sequences of tags. Out of this universe of tag sequences, we want to choose the tag sequence which is most probable given the observation sequence of n words w_1^n. In other words, we want, out of all sequences of n tags t_1^n the single tag sequence such that $P(t_1^n|w_1^n)$ is highest. We use the hat notation ˆ to mean "our estimate of the correct tag sequence".

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \tag{5.24}$$

The function $\operatorname{argmax}_x f(x)$ means "the x such that $f(x)$ is maximized". Equation (5.24) thus means, out of all tag sequences of length n, we want the particular tag sequence t_1^n which maximizes the right-hand side. While (5.24) is guaranteed to give us the optimal tag sequence, it is not clear how to make the equation operational; that is, for a given tag sequence t_1^n and word sequence w_1^n, we don't know how to directly compute $P(t_1^n|w_1^n)$.

The intuition of Bayesian classification is to use Bayes' rule to transform (5.24) into a set of other probabilities which turn out to be easier to compute. Bayes' rule is presented in (5.25); it gives us a way to break down any conditional probability $P(x|y)$ into three other probabilities:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{5.25}$$

We can then substitute (5.25) into (5.24) to get (5.26):

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)} \tag{5.26}$$

We can conveniently simplify 5.26 by dropping the denominator $P(w_1^n)$. Why is that? Since we are choosing a tag sequence out of all tag sequences, we will be computing $\frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$ for each tag sequence. But $P(w_1^n)$ doesn't change for each tag sequence; we are always asking about the most likely tag sequence for the same observation w_1^n, which must have the same probability $P(w_1^n)$. Thus we can choose the tag sequence which maximizes this simpler formula:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n|t_1^n)P(t_1^n) \tag{5.27}$$

PRIOR PROBABILITY

LIKELIHOOD

To summarize, the most probable tag sequence $\hat{t}_1^n$ given some word string w_1^n can be computed by taking the product of two probabilities for each tag sequence, and choosing the tag sequence for which this product is greatest. The two terms are the **prior probability** of the tag sequence $P(t_1^n)$), and the **likelihood** of the word string $P(w_1^n|t_1^n)$:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \overbrace{P(w_1^n|t_1^n)}^{\text{likelihood}} \overbrace{P(t_1^n)}^{\text{prior}} \tag{5.28}$$

Unfortunately, (5.28) is still too hard to compute directly. HMM taggers therefore make two simplifying assumptions. The first assumption is that the probability of a word appearing is dependent only on its own part-of-speech tag; that it is independent of other words around it, and of the other tags around it:

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i) \tag{5.29}$$

The second assumption is that the probability of a tag appearing is dependent only on the previous tag, the **bigram** assumption we saw in Ch. 4:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1}) \tag{5.30}$$

Plugging the simplifying assumptions (5.29) and (5.30) into (5.28) results in the following equation by which a bigram tagger estimates the most probable tag sequence:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}) \tag{5.31}$$

Equation (5.31) contains two kinds of probabilities, tag transition probabilities and word likelihoods. Let's take a moment to see what these probabilities represent. The tag transition probabilities, $P(t_i|t_{i-1})$, represent the probability of a tag given the previous tag. For example, determiners are very likely to precede adjectives and nouns, as in sequences like *that/DT flight/NN* and *the/DT yellow/JJ hat/NN*. Thus we would expect the probabilities $P(\mathrm{NN}|\mathrm{DT})$ and $P(\mathrm{JJ}|\mathrm{DT})$ to be high. But in English, adjectives don't tend to precede determiners, so the probability $P(\mathrm{DT}|\mathrm{JJ})$ ought to be low.

We can compute the maximum likelihood estimate of a tag transition probability $P(\mathrm{NN}|\mathrm{DT})$ by taking a corpus in which parts-of-speech are labeled and counting, out of the times we see DT, how many of those times we see NN after the DT. That is, we compute the following ratio of counts:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})} \tag{5.32}$$

Let's choose a specific corpus to examine. For the examples in this chapter we'll use the Brown corpus, the 1 million word corpus of American English described earlier. The Brown corpus has been tagged twice, once in the 1960's with the 87-tag tagset, and again in the 1990's with the 45-tag Treebank tagset. This makes it useful for comparing tagsets, and is also widely available.

In the 45-tag Treebank Brown corpus, the tag DT occurs 116,454 times. Of these, DT is followed by NN 56,509 times (if we ignore the few cases of ambiguous tags). Thus the MLE estimate of the transition probability is calculated as follows:

$$P(NN|DT) = \frac{C(DT,NN)}{C(DT)} = \frac{56,509}{116,454} = .49 \tag{5.33}$$

The probability of getting a common noun after a determiner, .49, is indeed quite high, as we suspected.

The word likelihood probabilities, $P(w_i|t_i)$, represent the probability, given that we see a given tag, that it will be associated with a given word. For example if we were to see the tag VBZ (third person singular present verb) and guess the verb that is likely to have that tag, we might likely guess the verb *is*, since the verb *to be* is so common in English.

We can compute the MLE estimate of a word likelihood probability like $P(\mathrm{is}|\mathrm{VBZ})$ again by counting, out of the times we see VBZ in a corpus, how many of those times the VBZ is labeling the word *is*. That is, we compute the following ratio of counts:

$$P(w_i|t_i) = \frac{C(t_i,w_i)}{C(t_i)} \tag{5.34}$$

In Treebank Brown corpus, the tag VBZ occurs 21,627 times, and VBZ is the tag for *is* 10,073 times. Thus:

$$P(is|VBZ) = \frac{C(VBZ,is)}{C(VBZ)} = \frac{10,073}{21,627} = .47 \tag{5.35}$$

For those readers who are new to Bayesian modeling note that this likelihood term is not asking "which is the most likely tag for the word *is*". That is, the term is not $P(\mathrm{VBZ}|\mathrm{is})$. Instead we are computing $P(\mathrm{is}|\mathrm{VBZ})$. The probability, slightly counterintuitively, answers the question "If we were expecting a third person singular verb, how likely is it that this verb would be *is*?".

We have now defined HMM tagging as a task of choosing a tag-sequence with the maximum probability, derived the equations by which we will compute this probability, and shown how to compute the component probabilities. In fact we have simplified the presentation of the probabilities in many ways; in later sections we will return to these equations and introduce the deleted interpolation algorithm for smoothing these counts, the trigram model of tag history, and a model for unknown words.

But before turning to these augmentations, we need to introduce the decoding algorithm by which these probabilities are combined to choose the most likely tag sequence.

5.5.1 Computing the most-likely tag sequence: A motivating example

The previous section showed that the HMM tagging algorithm chooses as the most likely tag sequence the one that maximizes the product of two terms; the probability of the sequence of tags, and the probability of each tag generating a word. In this section we ground these equations in a specific example, showing for a particular sentence how the correct tag sequence achieves a higher probability than one of the many possible wrong sequences.

We will focus on resolving the part-of-speech ambiguity of the word *race*, which can be a noun or verb in English, as we show in two examples modified from the Brown and Switchboard corpus. For this example, we will use the 87-tag Brown corpus tagset, because it has a specific tag for *to*, TO, used only when *to* is an infinitive; prepositional uses of *to* are tagged as IN. This will come in handy in our example.[1]

In (5.36) *race* is a verb (VB) while in (5.37) *race* is a common noun (NN):

(5.36) Secretariat/NNP is/BEZ expected/VBN to/TO **race**/VB tomorrow/NR

(5.37) People/NNS continue/VB to/TO inquire/VB the/AT reason/NN for/IN the/AT **race**/NN for/IN outer/JJ space/NN

Let's look at how *race* can be correctly tagged as a VB instead of an NN in (5.36). HMM part-of-speech taggers resolve this ambiguity globally rather than locally, picking the best tag sequence for the whole sentence. There are many hypothetically possible tag sequences for (5.36), since there are other ambiguities in the sentence (for example *expected* can be an adjective (JJ), a past tense/preterite (VBD) or a past participle (VBN)). But let's just consider two of the potential sequences, shown in Fig. 5.12. Note that these sequences differ only in one place; whether the tag chosen for *race* is VB or NN.

Almost all the probabilities in these two sequences are identical; in Fig. 5.12 we have highlighted in boldface the three probabilities that differ. Let's consider two of these, corresponding to $P(t_i|t_{i-1})$ and $P(w_i|t_i)$. The probability $P(t_i|t_{i-1})$ in Figure 5.12a is $P(\mathrm{VB}|\mathrm{TO})$, while in Figure 5.12b the transition probability is $P(\mathrm{NN}|\mathrm{TO})$.

The tag transition probabilities $P(\mathrm{NN}|\mathrm{TO})$ and $P(\mathrm{VB}|\mathrm{TO})$ give us the answer to the question "How likely are we to expect a verb (noun) given the previous tag?" As we

[1] The 45-tag Treebank-3 tagset does make this distinction in the Switchboard corpus but not, alas, in the Brown corpus. Recall that in the 45-tag tagset time adverbs like *tomorrow* are tagged as NN; in the 87-tag tagset they appear as NR.

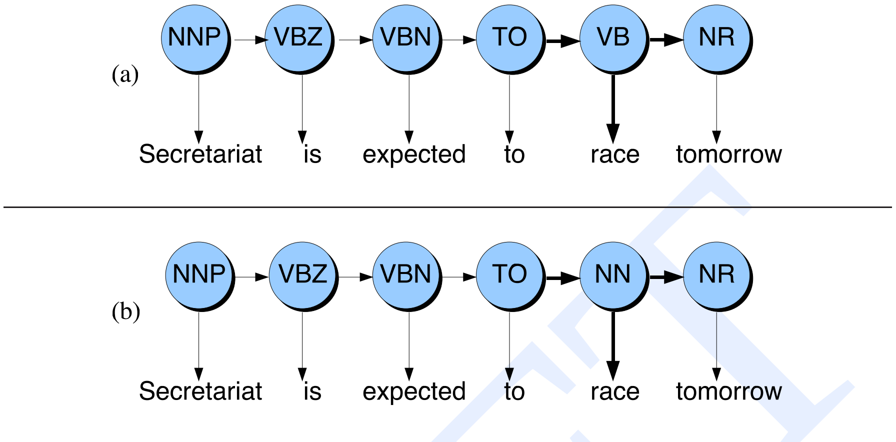


Figure 5.12 Two of the possible sequences of tags corresponding to the Secretariat sentence, one of them corresponding to the correct sequence, in which *race* is a VB. Each arc in these graphs would be associated with a probability. Note that the two graphs differ only in 3 arcs, hence in 3 probabilities.

saw in the previous section, the maximum likelihood estimate for these probabilities can be derived from corpus counts.

Since the (87-tag Brown tagset) tag TO is used only for the infinitive marker *to*, we expect that only a very small number of nouns can follow this marker (as an exercise, try to think of a sentence where a noun can follow the infinitive marker use of *to*). Sure enough, a look at the (87-tag) Brown corpus gives us the following probabilities, showing that verbs are about 500 times as likely as nouns to occur after TO:

$$\begin{aligned} P(\text{NN}|\text{TO}) &= .00047 \\ P(\text{VB}|\text{TO}) &= .83 \end{aligned}$$

Let's now turn to $P(w_i|t_i)$, the lexical likelihood of the word *race* given a part-of-speech tag. For the two possible tags VB and NN, these correspond to the probabilities $P(\text{race}|\text{VB})$ and $P(\text{race}|\text{NN})$. Here are the lexical likelihoods from Brown:

$$\begin{aligned} P(\text{race}|\text{NN}) &= .00057 \\ P(\text{race}|\text{VB}) &= .00012 \end{aligned}$$

Finally, we need to represent the tag sequence probability for the following tag (in this case the tag NR for *tomorrow*):

$$\begin{aligned} P(\text{NR}|\text{VB}) &= .0027 \\ P(\text{NR}|\text{NN}) &= .0012 \end{aligned}$$

If we multiply the lexical likelihoods with the tag sequence probabilities, we see that the probability of the sequence with the VB tag is higher and the HMM tagger correctly tags *race* as a VB in Fig. 5.12 despite the fact that it is the less likely sense of *race*:

$$\begin{aligned} P(\text{VB}|\text{TO})P(\text{NR}|\text{VB})P(\text{race}|\text{VB}) &= .00000027 \\ P(\text{NN}|\text{TO})P(\text{NR}|\text{NN})P(\text{race}|\text{NN}) &= .00000000032 \end{aligned}$$

5.5.2 Formalizing Hidden Markov Model taggers

Now that we have seen the equations and some examples of choosing the most probable tag sequence, we show a brief formalization of this problem as a Hidden Markov Model (see Ch. 6 for the more complete formalization).

WEIGHTED

MARKOV CHAIN

The HMM is an extension of the finite automata of Ch. 3. Recall that a finite automaton is defined by a set of states, and a set of transitions between states that are taken based on the input observations. A **weighted finite-state automaton** is a simple augmentation of the finite automaton in which each arc is associated with a probability, indicating how likely that path is to be taken. The probability on all the arcs leaving a node must sum to 1. A **Markov chain** is a special case of a weighted automaton in which the input sequence uniquely determines which states the automaton will go through. Because they can't represent inherently ambiguous problems, a Markov chain is only useful for assigning probabilities to unambiguous sequences.

HIDDEN MARKOV MODEL

While the Markov chain is appropriate for situations where we can see the actual conditioning events, it is not appropriate in part-of-speech tagging. This is because in part-of-speech tagging, while we observe the words in the input, we do *not* observe the part-of-speech tags. Thus we can't condition any probabilities on, say, a previous part-of-speech tag, because we cannot be completely certain exactly which tag applied to the previous word. A **Hidden Markov Model** (**HMM**) allows us to talk about both *observed* events (like words that we see in the input) and *hidden* events (like part-of-speech tags) that we think of as causal factors in our probabilistic model.

HMM

An **HMM** is specified by the following components:

$Q = q_1 q_2 \dots q_N$ a set of N **states**

$A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$ a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$

$O = o_1 o_2 \dots o_T$ a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$.

$B = b_i(o_t)$ A sequence of **observation likelihoods:**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i.

q_0, q_F a special **start state** and **end (final) state** which are not associated with observations, together with transition probabilities $a_{01} a_{02} .. a_{0n}$ out of the start state and $a_{1F} a_{2F} ... a_{nF}$ into the end state.

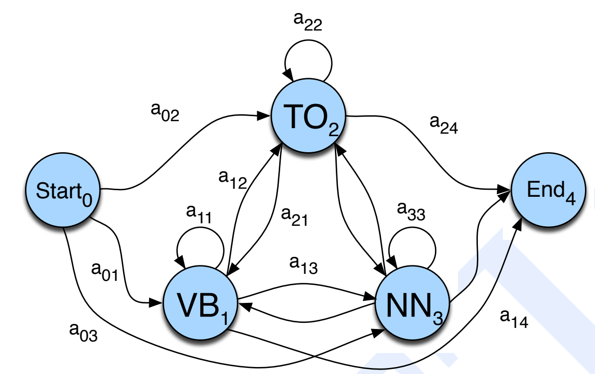


Figure 5.13 The Markov chain corresponding to the hidden states of the HMM. The *A* transition probabilities are used to compute the prior probability.

An HMM thus has two kinds of probabilities; the *A* transition probabilities, and the *B* observation likelihoods, corresponding respectively to the **prior** and **likelihood** probabilities that we saw in equation (5.31). Fig. 5.13 illustrates the prior probabilities in an HMM part-of-speech tagger, showing 3 sample states and some of the *A* transition probabilities between them. Fig. 5.14 shows another view of an HMM part-of-speech tagger, focusing on the word likelihoods *B*. Each hidden state is associated with a vector of likelihoods for each observation word.

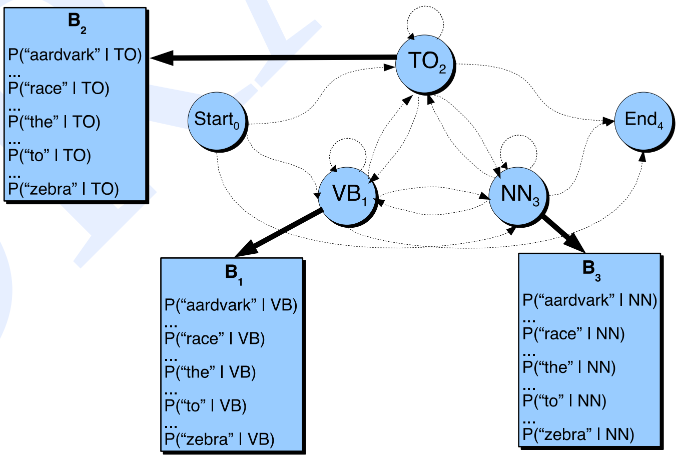


Figure 5.14 The *B* observation likelihoods for the HMM in the previous figure. Each state (except the non-emitting Start and End states) is associated with a vector of probabilities, one likelihood for each possible observation word.

5.5.3 The Viterbi Algorithm for HMM Tagging

DECODING

VITERBI

For any model, such as an HMM, that contains hidden variables, the task of determining which sequence of variables is the underlying source of some sequence of observations is called the **decoding** task. The **Viterbi** algorithm is perhaps the most common decoding algorithm used for HMMs, whether for part-of-speech tagging or for speech recognition. The term **Viterbi** is common in speech and language processing, but this is really a standard application of the classic **dynamic programming** algorithm, and looks a lot like the **minimum edit distance** algorithm of Ch. 3. The Viterbi algorithm was first applied to speech and language processing in the context of speech recognition by Vintsyuk (1968), but has what Kruskal (1983) calls a 'remarkable history of multiple independent discovery and publication'; see the History section at the end of Ch. 6 for more details.

The slightly simplified version of the Viterbi algorithm that we will present takes as input a single HMM and a set of observed words $O = (o_1 o_2 o_3 \ldots o_T)$ and returns the most probable state/tag sequence $Q = (q_1 q_2 q_3 \ldots q_T)$, together with its probability.

Let the HMM be defined by the two tables in Fig. 5.15 and Fig. 5.16. Fig. 5.15 expresses the a_{ij} probabilities, the *transition* probabilities between hidden states (i.e. part-of-speech tags). Fig. 5.16 expresses the $b_i(o_t)$ probabilities, the *observation* likelihoods of words given tags.

	VB	**TO**	**NN**	**PPSS**
<s>	.019	.0043	.041	.067
VB	.0038	.035	.047	.0070
TO	.83	0	.00047	0
NN	.0040	.016	.087	.0045
PPSS	.23	.00079	.0012	.00014

Figure 5.15 Tag transition probabilities (the a array, $p(t_i|t_{i-1})$) computed from the 87-tag Brown corpus without smoothing. The rows are labeled with the conditioning event; thus $P(PPSS|VB)$ is .0070. The symbol <s> is the start-of-sentence symbol.

	I	**want**	**to**	**race**
VB	0	.0093	0	.00012
TO	0	0	.99	0
NN	0	.000054	0	.00057
PPSS	.37	0	0	0

Figure 5.16 Observation likelihoods (the b array) computed from the 87-tag Brown corpus without smoothing.

Fig. 5.17 shows pseudocode for the Viterbi algorithm. The Viterbi algorithm sets up a probability matrix, with one column for each observation t and one row for each state in the state graph. Each column thus has a cell for each state q_i in the single combined automaton for the four words.

```
function VITERBI(observations of len T,state-graph of len N) returns best-path

  create a path probability matrix viterbi[N+2,T]
  for each state s from 1 to N do                          ;initialization step
      $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
      $backpointer[s,1] \leftarrow 0$
  for each time step t from 2 to T do                      ;recursion step
    for each state s from 1 to N do
      $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
      $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
  $viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$          ; termination step
  $backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$          ; termination step
  return the backtrace path by following backpointers to states back in time from
$backpointer[q_F,T]$
```

Figure 5.17 Viterbi algorithm for finding optimal sequence of tags. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state-path through the HMM which assigns maximum likelihood to the observation sequence. Note that states 0 and q_F are non-emitting.

The algorithm first creates N or four state columns. The first column corresponds to the observation of the first word *i*, the second to the second word *want*, the third to the third word *to*, and the fourth to the fourth word *race*. We begin in the first column by setting the viterbi value in each cell to the product of the transition probability (into it from the state state) and the observation probability (of the first word); the reader should find this in Fig. 5.18.

Then we move on, column by column; for every state in column 1, we compute the probability of moving into each state in column 2, and so on. For each state q_j at time t, the value $viterbi[s,t]$ is computed by taking the maximum over the extensions of all the paths that lead to the current cell, following the following equation:

$$v_t(j) \quad = \quad \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t) \qquad (5.38)$$

The three factors that are multiplied in Eq. 5.38 for extending the previous paths to compute the Viterbi probability at time t are:

$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

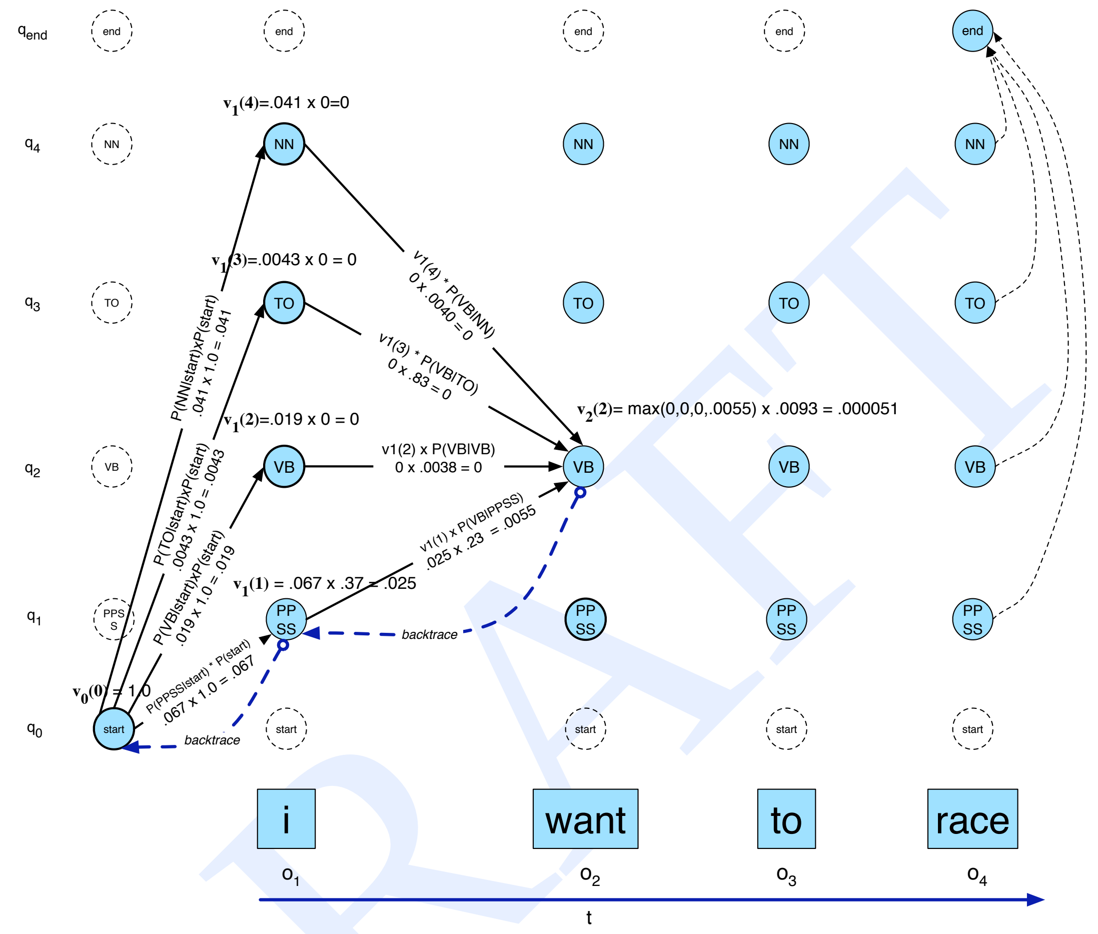


Figure 5.18 The entries in the individual state columns for the Viterbi algorithm. Each cell keeps the probability of the best path so far and a pointer to the previous cell along that path. We have only filled out columns 0 and 1 and one cell of column 2; the rest is left as an exercise for the reader. After the cells are filled in, backtracing from the *end* state, we should be able to reconstruct the correct state sequence PPSS VB TO VB.

In Fig. 5.18, each cell of the trellis in the column for the word *I* is computed by multiplying the previous probability at the start state (1.0), the transition probability from the start state to the tag for that cell, and the observation likelihood of the word *I* given the tag for that cell. As it turns out, three of the cells are zero (since the word *I* can be neither NN, TO nor VB). Next, each cell in the *want* column gets updated with the maximum probability path from the previous column. We have shown only the value for the VB cell. That cell gets the max of four values; as it happens in this case, three of them are zero (since there were zero values in the previous column). The remaining value is multiplied by the relevant transition probability, and the (trivial) max is taken. In this case the final value, .000051, comes from the PPSS state at the previous column.

The reader should fill in the rest of the trellis in Fig. 5.18, and backtrace to reconstruct the correct state sequence PPSS VB TO VB.

5.5.4 Extending the HMM algorithm to trigrams

We mentioned earlier that HMM taggers in actual use have a number of sophistications not present in the simplified tagger as we have described it so far. One important missing feature has to do with the tag context. In the tagger described above, we assume that the probability of a tag appearing is dependent only on the previous tag:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1}) \tag{5.39}$$

Most modern HMM taggers actually use a little more of the history, letting the probability of a tag depend on the two previous tags:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1},t_{i-2}) \tag{5.40}$$

In addition to increasing the window before a tagging decision, state-of-the-art HMM taggers like Brants (2000) let the tagger know the location of the end of the sentence by adding dependence on an end-of-sequence marker for t_{n+1}. This gives the following equation for part of speech tagging:

$$\hat{t}_1^n = \operatorname*{argmax}_{t_1^n} P(t_1^n|w_1^n) \approx \operatorname*{argmax}_{t_1^n} \left[\prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1},t_{i-2})\right] P(t_{n+1}|t_n) \tag{5.41}$$

In tagging any sentence with (5.41), three of the tags used in the context will fall off the edge of the sentence, and hence will not match regular words. These tags, t_{-1}, t_0, and t_{n+1}, can all be set to be a single special 'sentence boundary' tag which is added to the tagset. This requires that sentences passed to the tagger have sentence boundaries demarcated, as discussed in Ch. 3.

There is one large problem with (5.41); data sparsity. Any particular sequence of tags t_{i-2},t_{i-1},t_i that occurs in the test set may simply never have occurred in the training set. That means we cannot compute the tag trigram probability just by the maximum likelihood estimate from counts, following Equation (5.42):

$$P(t_i|t_{i-1},t_{i-2}) = \frac{C(t_{i-2},t_{i-1},t_i)}{C(t_{i-2},t_{i-1})} : \tag{5.42}$$

Why not? Because many of these counts will be zero in any training set, and we will incorrectly predict that a given tag sequence will never occur! What we need is a way to estimate $P(t_i|t_{i-1},t_{i-2})$ even if the sequence t_{i-2},t_{i-1},t_i never occurs in the training data.

The standard approach to solve this problem is to estimate the probability by combining more robust, but weaker estimators. For example, if we've never seen the tag sequence PRP VB TO, so we can't compute $P(\text{TO}|\text{PRP,VB})$ from this frequency, we still could rely on the bigram probability $P(\text{TO}|\text{VB})$, or even the unigram probability $P(\text{TO})$. The maximum likelihood estimation of each of these probabilities can be computed from a corpus via the following counts:

$$\text{Trigrams} \quad \hat{P}(t_i|t_{i-1},t_{i-2}) = \frac{C(t_{i-2},t_{i-1},t_i)}{C(t_{i-2},t_{i-1})} \tag{5.43}$$

$$\text{Bigrams} \quad \hat{P}(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})} \tag{5.44}$$

$$\text{Unigrams} \quad \hat{P}(t_i) = \frac{C(t_i)}{N} \tag{5.45}$$

How should these three estimators be combined in order to estimate the trigram probability $P(t_i|t_{i-1},t_{i-2})$? The simplest method of combination is linear interpolation. In linear interpolation, we estimate the probability $P(t_i|t_{i-1}t_{i-2})$ by a weighted sum of the unigram, bigram, and trigram probabilities:

$$P(t_i|t_{i-1}t_{i-2}) = \lambda_1\hat{P}(t_i|t_{i-1}t_{i-2}) + \lambda_2\hat{P}(t_i|t_{i-1}) + \lambda_3\hat{P}(t_i) \tag{5.46}$$

DELETED INTERPOLATION

We require $\lambda_1 + \lambda_2 + \lambda_3 = 1$, insuring that the resulting P is a probability distribution. How should these λs be set? One good way is **deleted interpolation**, developed by Jelinek and Mercer (1980). In deleted interpolation, we successively delete each trigram from the training corpus, and choose the λs so as to maximize the likelihood of the rest of the corpus. The idea of the deletion is to set the λs in such a way as to generalize to unseen data and not overfit the training corpus. Fig. 5.19 gives the Brants (2000) version of the deleted interpolation algorithm for tag trigrams.

Brants (2000) achieves an accuracy of 96.7% on the Penn Treebank with a trigram HMM tagger. Weischedel et al. (1993) and DeRose (1988) have also reported accuracies of above 96% for HMM tagging. (Thede and Harper, 1999) offer a number of augmentations of the trigram HMM model, including the idea of conditioning word likelihoods on neighboring words and tags.

The HMM taggers we have seen so far are trained on hand-tagged data. Kupiec (1992), Cutting et al. (1992), and others show that it is also possible to train an HMM tagger on unlabeled data, using the EM algorithm that we will introduce in Ch. 6. These taggers still start with a dictionary which lists which tags can be assigned to which words; the EM algorithm then learns the word likelihood function for each tag, and the tag transition probabilities. An experiment by Merialdo (1994), however, indicates that with even a small amount of training data, a tagger trained on hand-tagged data worked better than one trained via EM. Thus the EM-trained "pure HMM" tagger is probably best suited to cases where no training data is available, for example when tagging languages for which there is no previously hand-tagged data.

```
function DELETED-INTERPOLATION(corpus) returns $\lambda_1, \lambda_2, \lambda_3$

  $\lambda_1 \leftarrow 0$
  $\lambda_2 \leftarrow 0$
  $\lambda_3 \leftarrow 0$
  foreach trigram $t_1, t_2, t_3$ with $f(t_1, t_2, t_3) > 0$
    depending on the maximum of the following three values
      case $\frac{C(t_1,t_2,t_3)-1}{C(t_1,t_2)-1}$: increment $\lambda_3$ by $C(t_1, t_2, t_3)$
      case $\frac{C(t_2,t_3)-1}{C(t_2)-1}$: increment $\lambda_2$ by $C(t_1, t_2, t_3)$
      case $\frac{C(t_3)-1}{N-1}$: increment $\lambda_1$ by $C(t_1, t_2, t_3)$
    end
  end
  normalize $\lambda_1, \lambda_2, \lambda_3$
  return $\lambda_1, \lambda_2, \lambda_3$
```

Figure 5.19 The deleted interpolation algorithm for setting the weights for combining unigram, bigram, and trigram tag probabilities. If the denominator is 0 for any case, we define the result of that case to be 0. N is the total number of tokens in the corpus. After Brants (2000).

5.6 TRANSFORMATION-BASED TAGGING

TRANSFORMATION-BASED LEARNING

Transformation-Based Tagging, sometimes called Brill tagging, is an instance of the **Transformation-Based Learning** (TBL) approach to machine learning (Brill, 1995), and draws inspiration from both the rule-based and stochastic taggers. Like the rule-based taggers, TBL is based on rules that specify what tags should be assigned to what words. But like the stochastic taggers, TBL is a machine learning technique, in which rules are automatically induced from the data. Like some but not all of the HMM taggers, TBL is a supervised learning technique; it assumes a pre-tagged training corpus.

Samuel et al. (1998) offer a useful analogy for understanding the TBL paradigm, which they credit to Terry Harvey. Imagine an artist painting a picture of a white house with green trim against a blue sky. Suppose most of the picture was sky, and hence most of the picture was blue. The artist might begin by using a very broad brush and painting the entire canvas blue. Next she might switch to a somewhat smaller white brush, and paint the entire house white. She would just color in the whole house, not worrying about the brown roof, or the blue windows or the green gables. Next she takes a smaller brown brush and colors over the roof. Now she takes up the blue paint on a small brush and paints in the blue windows on the house. Finally she takes a very fine green brush and does the trim on the gables.

The painter starts with a broad brush that covers a lot of the canvas but colors a lot of areas that will have to be repainted. The next layer colors less of the canvas, but also makes less "mistakes". Each new layer uses a finer brush that corrects less of the picture, but makes fewer mistakes. TBL uses somewhat the same method as this painter. The TBL algorithm has a set of tagging rules. A corpus is first tagged using the broadest rule, that is, the one that applies to the most cases. Then a slightly more specific rule is chosen, which changes some of the original tags. Next an even narrower rule, which changes a smaller number of tags (some of which might be previously changed tags).

5.6.1 How TBL Rules Are Applied

Let's look at one of the rules used by Brill's (1995) tagger. Before the rules apply, the tagger labels every word with its most-likely tag. We get these most-likely tags from a tagged corpus. For example, in the Brown corpus, *race* is most likely to be a noun:

$$\begin{aligned} P(\text{NN}|\text{race}) &= .98 \\ P(\text{VB}|\text{race}) &= .02 \end{aligned}$$

This means that the two examples of *race* that we saw above will both be coded as NN. In the first case, this is a mistake, as NN is the incorrect tag:

(5.47) is/VBZ expected/VBN to/TO race/**NN** tomorrow/NN

In the second case this *race* is correctly tagged as an NN:

(5.48) the/DT race/**NN** for/IN outer/JJ space/NN

After selecting the most-likely tag, Brill's tagger applies its transformation rules. As it happens, Brill's tagger learned a rule that applies exactly to this mistagging of *race*:

Change NN *to* VB *when the previous tag is* TO

This rule would change *race/NN* to *race/VB* in exactly the following situation, since it is preceded by *to/TO*:

(5.49) expected/VBN to/TO race/NN → expected/VBN to/TO race/VB

5.6.2 How TBL Rules Are Learned

Brill's TBL algorithm has three major stages. It first labels every word with its most-likely tag. It then examines every possible transformation, and selects the one that results in the most improved tagging. Finally, it then re-tags the data according to this rule. The last two stages are repeated until some stopping criterion is reached, such as insufficient improvement over the previous pass. Note that stage two requires that TBL knows the correct tag of each word; that is, TBL is a supervised learning algorithm.

The output of the TBL process is an ordered list of transformations; these then constitute a "tagging procedure" that can be applied to a new corpus. In principle the set of possible transformations is infinite, since we could imagine transformations such as "transform NN to VB if the previous word was "IBM" and the word "the" occurs between 17 and 158 words before that". But TBL needs to consider every possible transformation, in order to pick the best one on each pass through the algorithm. Thus the algorithm needs a way to limit the set of transformations. This is done by designing a small set of **templates** (abstracted transformations). Every allowable transformation TEMPLATES is an instantiation of one of the templates. Brill's set of templates is listed in Fig. 5.20. Fig. 5.21 gives the details of this algorithm for learning transformations.

The preceding (following) word is tagged **z**.
The word two before (after) is tagged **z**.
One of the two preceding (following) words is tagged **z**.
One of the three preceding (following) words is tagged **z**.
The preceding word is tagged **z** and the following word is tagged **w**.
The preceding (following) word is tagged **z** and the word
two before (after) is tagged **w**.

Figure 5.20 Brill's (1995) templates. Each begins with *"Change tag* **a** *to tag* **b** *when:* ... ". The variables **a**, **b**, **z**, and **w** range over parts-of-speech.

At the heart of Fig. 5.21 are the two functions GET_BEST_TRANSFORMATION and GET_BEST_INSTANCE. GET_BEST_TRANSFORMATION is called with a list of potential templates; for each template, it calls GET_BEST_INSTANCE. GET_BEST_INSTANCE iteratively tests every possible instantiation of each template by filling in specific values for the tag variables **a**, **b**, **z**, and **w**.

In practice, there are a number of ways to make the algorithm more efficient. For example, templates and instantiated transformations can be suggested in a data-driven manner; a transformation-instance might only be suggested if it would improve the tagging of some specific word. The search can also be made more efficient by pre-indexing the words in the training corpus by potential transformation. Roche and Schabes (1997) show how the tagger can also be speeded up by converting each rule into a finite-state transducer and composing all the transducers.

Fig. 5.22 shows a few of the rules learned by Brill's original tagger.

5.7 Evaluation and Error Analysis

The probabilities in a statistical model like an HMM POS-tagger come from the corpus it is trained on. We saw in Sec. **??** that in order to train statistical models like taggers or *N*-grams, we need to set aside a **training set**. The design of the training set or **training corpus** needs to be carefully considered. If the training corpus is too specific to the task or domain, the probabilities may be too narrow and not generalize well to tagging sentences in very different domains. But if the training corpus is too general, the probabilities may not do a sufficient job of reflecting the task or domain.

For evaluating *N*-grams models, we said in Sec. **??** that we need to divide our corpus into a distinct training set, test set, and a second test set called a development test set. We train our tagger on the training set. Then we use the **development test set** (also called a **devtest** set) to perhaps tune some parameters, and in general decide what the best model is. Then once we come up with what we think is the best model, we run it on the (hitherto unseen) test set to see its performance. We might use 80% of our data for training, and save 10% each for devtest and test. Why do we need a development test set distinct from the final test set? Because if we used the final test

DEVELOPMENT TEST SET
DEVTEST

```
function TBL(corpus) returns transforms-queue
 INTIALIZE-WITH-MOST-LIKELY-TAGS(corpus)
 until end condition is met do
   templates ← GENERATE-POTENTIAL-RELEVANT-TEMPLATES
   best-transform ← GET-BEST-TRANSFORM(corpus, templates)
   APPLY-TRANSFORM(best-transform, corpus)
   ENQUEUE(best-transform-rule, transforms-queue)
 end
 return(transforms-queue)
```

```
 function GET-BEST-TRANSFORM(corpus, templates) returns transform
   for each template in templates
     (instance, score) ← GET-BEST-INSTANCE(corpus, template)
     if (score > best-transform.score) then best-transform ← (instance, score)
   return(best-transform)
```

```
 function GET-BEST-INSTANCE(corpus, template) returns transform
   for from-tag ← from $tag_1$ to $tag_n$ do
     for to-tag ← from $tag_1$ to $tag_n$ do
       for pos ← from 1 to corpus-size do
         if (correct-tag(pos) == to-tag && current-tag(pos) == from-tag)
              num-good-transforms(current-tag(pos−1))++
         elseif (correct-tag(pos)==from-tag && current-tag(pos)==from-tag)
              num-bad-transforms(current-tag(pos−1))++
       end
       best-Z ← ARGMAX$_t$(num-good-transforms(t) - num-bad-transforms(t))
        if(num-good-transforms(best-Z) - num-bad-transforms(best-Z)
                     > best-instance.score) then
          best.rule ← "Change tag from from-tag to to-tag if prev tag is best-Z"
          best.score ← num-good-transforms(best-Z) - num-bad-transforms(best-Z)
 return(best)
```

```
 procedure APPLY-TRANSFORM(transform, corpus)
 for pos ← from 1 to corpus-size do
   if (current-tag(pos)==best-rule-from)
          && (current-tag(pos−1)==best-rule-prev))
      current-tag(pos) ← best-rule-to
```

Figure 5.21 The TBL algorithm for learning to tag. GET_BEST_INSTANCE would have to change for transformation templates other than *"Change tag from X to Y if previous tag is Z"*. After Brill (1995).

set to compute performance for all our experiments during our development phase, we would be tuning the various changes and parameters to this set. Our final error rate on the test set would then be optimistic: it would underestimate the true error rate.

	Change tags			
#	From	To	Condition	Example
1	NN	VB	Previous tag is TO	to/TO race/NN → VB
2	VBP	VB	One of the previous 3 tags is MD	might/MD vanish/VBP → VB
3	NN	VB	One of the previous 2 tags is MD	might/MD not reply/NN → VB
4	VB	NN	One of the previous 2 tags is DT	
5	VBD	VBN	One of the previous 3 tags is VBZ	

Figure 5.22 The first 20 nonlexicalized transformations from Brill (1995).

CROSSVALIDATION

10-FOLD CROSSVALIDATION

The problem with having a fixed training set, devset, and test set is that in order to save lots of data for training, the test set might not be large enough to be representative. Thus a better approach would be to somehow use **all** our data both for training and test. How is this possible? The idea is to use **crossvalidation**. In crossvalidation, we randomly choose a training and test set division of our data, train our tagger, and then compute the error rate on the test set. Then we repeat with a different randomly selected training set and test set. We do this sampling process 10 times, and then average these 10 runs to get an average error rate. This is called **10-fold crossvalidation**.

The only problem with cross-validation is that because all the data is used for testing, we need the whole corpus to be blind; we can't examine any of the data to suggest possible features, and in general see what's going on. But looking at the corpus is often important for designing the system. For this reason it is common to create a fixed training set and test set, and then to do 10-fold crossvalidation inside the training set, but compute error rate the normal way in the test set.

Once we have a test set, taggers are evaluated by comparing their labeling of the test set with a human-labeled **Gold Standard** test set, based on **accuracy**: the percentage of all tags in the test set where the tagger and the Gold standard agree. Most current tagging algorithms have an accuracy of around 96–97% for simple tagsets like the Penn Treebank set. These accuracies are for words and punctuation; the accuracy for words only would be lower.

BASELINE

CEILING

How good is 97%? Since tagsets and tasks differ, the performance of tags can be compared against a lower-bound **baseline** and an upper-bound **ceiling**. One way to set a ceiling is to see how well humans do on the task. Marcus et al. (1993), for example, found that human annotators agreed on about 96–97% of the tags in the Penn Treebank version of the Brown corpus. This suggests that the Gold Standard may have a 3-4% margin of error, and that it is meaningless to get 100% accuracy, (modeling the last 3% would just be modeling noise). Indeed Ratnaparkhi (1996) showed that the tagging ambiguities that caused problems for his tagger were exactly the ones that humans had labeled inconsistently in the training set. Two experiments by Voutilainen (1995, p. 174), however, found that when humans were allowed to discuss tags, they reached consensus on 100% of the tags.

> **Human Ceiling:** When using a human Gold Standard to evaluate a classification algorithm, check the agreement rate of humans on the standard.

The standard **baseline**, suggested by Gale et al. (1992) (in the slightly different context of word-sense disambiguation), is to choose the **unigram most-likely tag** for each ambiguous word. The most-likely tag for each word can be computed from a hand-tagged corpus (which may be the same as the training corpus for the tagger being evaluated).

> **Most Frequent Class Baseline:** Always compare a classifier against a baseline at least as good as the most frequent class baseline (assigning each token to the class it occurred in most often in the training set).

Tagging algorithms since Harris (1962) incorporate this tag frequency intuition. Charniak et al. (1993) showed that this baseline algorithm achieves an accuracy of 90–91% on the 87-tag Brown tagset; Toutanova et al. (2003) showed that a more complex version, augmented with an unknown word model, achieved 93.69% on the 45-tag Treebank tagset.

PAIRED TESTS
WILCOXON SIGNED-RANK TEST
PAIRED T-TESTS
MAPSSWE
MCNEMAR TEST

When comparing models it is important to use statistical tests (introduced in any statistics class or textbook for the social sciences) to determine if the difference between two models is significant. Cohen (1995) is a useful reference which focuses on statistical research methods for artificial intelligence. Dietterich (1998) focuses on statistical tests for comparing classifiers. When statistically comparing sequence models like part-of-speech taggers, it is important to use **paired tests**. Commonly used paired tests for evaluating part-of-speech taggers include the **Wilcoxon signed-rank test**, **paired t-tests**, versions of matched t-tests such as the Matched-Pair Sentence Segment Word Error (**MAPSSWE**) test originally applied to speech recognition word error rate, and the **McNemar test** .

5.7.1 Error Analysis

In order to improve any model we need to understand where it went wrong. Analyzing the error in a classifier like a part-of-speech tagger is done via a **confusion matrix**, or **contingency table**. A confusion matrix for an N-way classification task is an N-by-N matrix where the cell (x,y) contains the number of times an item with correct classification x was classified by the model as y. For example, the following table shows a portion of the confusion matrix from the HMM tagging experiments of Franz (1996). The row labels indicate correct tags, column labels indicate the tagger's hypothesized tags, and each cell indicates percentage of the overall tagging error. Thus 4.4% of the total errors were caused by mistagging a VBD as a VBN. Common errors are boldfaced.

	IN	**JJ**	**NN**	**NNP**	**RB**	**VBD**	**VBN**
IN	-	.2			.7		
JJ	.2	-	**3.3**	2.1	1.7	.2	**2.7**
NN		**8.7**	-				.2
NNP	.2	**3.3**	**4.1**	-	.2		
RB	**2.2**	2.0	.5		-		
VBD		.3	.5			-	**4.4**
VBN		**2.8**				**2.6**	-

The confusion matrix above, and related error analyses in Franz (1996), Kupiec (1992), and Ratnaparkhi (1996), suggest that some major problems facing current taggers are:

1. **NN versus NNP versus JJ:** These are hard to distinguish prenominally. Distinguishing proper nouns is especially important for information extraction and machine translation.
2. **RP versus RB versus IN:** All of these can appear in sequences of satellites immediately following the verb.
3. **VBD versus VBN versus JJ:** Distinguishing these is important for partial parsing (participles are used to find passives), and for correctly labeling the edges of noun-phrases.

Error analysis like this is a crucial part of any computational linguistic application. Error analysis can help find bugs, find problems in the training data, and, most important, help in developing new kinds of knowledge or algorithms to use in solving problems.

5.8 Advanced Issues in Part-of-Speech Tagging

5.8.1 Practical Issues: Tag Indeterminacy and Tokenization

Tag indeterminacy arises when a word is ambiguous between multiple tags and it is impossible or very difficult to disambiguate. In this case, some taggers allow the use of multiple tags. This is the case in both the Penn Treebank and in the British National Corpus. Common tag indeterminacies include adjective versus preterite versus past participle (JJ/VBD/VBN), and adjective versus noun as prenominal modifier (JJ/NN). Given a corpus with these indeterminate tags, there are 3 ways to deal with tag indeterminacy when training and scoring part-of-speech taggers:

1. Somehow replace the indeterminate tags with only one tag.
2. In testing, count a tagger as having correctly tagged an indeterminate token if it gives either of the correct tags. In training, somehow choose only one of the tags for the word.
3. Treat the indeterminate tag as a single complex tag.

The second approach is perhaps the most sensible, although most previous published results seem to have used the third approach. This third approach applied to the Penn Treebank Brown corpus, for example, results in a much larger tagset of 85 tags instead of 45, but the additional 40 complex tags cover a total of only 121 word instances out of the million word corpus.

Most tagging algorithms assume a process of tokenization has been applied to the tags. Ch. 3 discussed the issue of tokenization of periods for distinguishing sentence-final periods from word-internal period in words like *etc.*. An additional role for tokenization is in word splitting. The Penn Treebank and the British National Corpus split contractions and the *’s*-genitive from their stems:

would/MD n’t/RB
children/NNS ’s/POS

Indeed, the special Treebank tag POS is used only for the morpheme *'s* which must be segmented off during tokenization.

Another tokenization issue concerns multi-part words. The Treebank tagset assumes that tokenization of words like *New York* is done at whitespace. The phrase *a New York City firm* is tagged in Treebank notation as five separate words: *a/DT New/NNP York/NNP City/NNP firm/NN*. The C5 tagset, by contrast, allow prepositions like "*in terms of*" to be treated as a single word by adding numbers to each tag, as in *in/II31 terms/II32 of/II33*.

5.8.2 Unknown Words

words people
never use —
could be
only I
know them

Ishikawa Takuboku 1885–1912

All the tagging algorithms we have discussed require a dictionary that lists the possible parts-of-speech of every word. But the largest dictionary will still not contain every possible word, as we saw in Ch. 7. Proper names and acronyms are created very often, and even new common nouns and verbs enter the language at a surprising rate. Therefore in order to build a complete tagger we cannot always use a dictionary to give us $p(w_i|t_i)$. We need some method for guessing the tag of an unknown word.

The simplest possible unknown-word algorithm is to pretend that each unknown word is ambiguous among all possible tags, with equal probability. Then the tagger must rely solely on the contextual POS-trigrams to suggest the proper tag. A slightly more complex algorithm is based on the idea that the probability distribution of tags over unknown words is very similar to the distribution of tags over words that occurred only once in a training set, an idea that was suggested by both Baayen and Sproat (1996) and Dermatas and Kokkinakis (1995). These words that only occur once are known as **hapax legomena** (singular **hapax legomenon**). For example, unknown words and *hapax legomena* are similar in that they are both most likely to be nouns, followed by verbs, but are very unlikely to be determiners or interjections. Thus the likelihood $P(w_i|t_i)$ for an unknown word is determined by the average of the distribution over all singleton words in the training set. This idea of using "things we've seen once" as an estimator for "things we've never seen" will prove useful in the Good-Turing algorithm of Ch. 4.

HAPAX LEGOMENA

Most unknown-word algorithms, however, make use of a much more powerful source of information: the morphology of the words. For example, words that end in *-s* are likely to be plural nouns (NNS), words ending with *-ed* tend to be past participles (VBN), words ending with *able* tend to be adjectives (JJ), and so on. Even if we've never seen a word, we can use facts about its morphological form to guess its part-of-speech. Besides morphological knowledge, orthographic information can be very helpful. For example words starting with capital letters are likely to be proper nouns (NP). The presence of a hyphen is also a useful feature; hyphenated words in the Treebank version of Brown are most likely to be adjectives (JJ). This prevalence of JJs is caused by the labeling instructions for the Treebank, which specified that prenominal modifiers should be labeled as JJ if they contained a hyphen.

How are these features combined and used in part-of-speech taggers? One method is to train separate probability estimators for each feature, assume independence, and multiply the probabilities. Weischedel et al. (1993) built such a model, based on four specific kinds of morphological and orthographic features. They used 3 inflectional endings (*-ed*, *-s*, *-ing*), 32 derivational endings (such as *-ion*, *-al*, *-ive*, and *-ly*), 4 values of capitalization depending on whether a word is sentence-initial (+/- capitalization, +/- initial) and whether the word was hyphenated. For each feature, they trained maximum likelihood estimates of the probability of the feature given a tag from a labeled training set. They then combined the features to estimate the probability of an unknown word by assuming independence and multiplying:

$$P(w_i|t_i) = p(\text{unknown-word}|t_i) * p(\text{capital}|t_i) * p(\text{endings/hyph}|t_i) \tag{5.50}$$

Another HMM-based approach, due to Samuelsson (1993) and Brants (2000), generalizes this use of morphology in a data-driven way. In this approach, rather than pre-selecting certain suffixes by hand, all final letter sequences of all words are considered. They consider such suffixes of up to ten letters, computing for each suffix of length i the probability of the tag t_i given the suffix:

$$P(t_i|l_{n-i+1}\dots l_n) \tag{5.51}$$

These probabilities are smoothed using successively shorter and shorter suffixes. Separate suffix tries are kept for capitalized and uncapitalized words.

In general, most unknown word models try to capture the fact that unknown words are unlikely to be closed-class words like prepositions. Brants models this fact by only computing suffix probabilities from the training set for words whose frequency in the training set is ≤ 10. In the HMM tagging model of Thede and Harper (1999), this fact is modeled instead by only training on open-class words.

Note that (5.51) gives an estimate of $p(t_i|w_i)$; since for the HMM tagging approach we need the likelihood $p(w_i|t_i)$, this can be derived from (5.51) using Bayesian inversion (i.e. using Bayes rule and computation of the two priors $P(t_i)$ and $P(t_i|l_{n-i+1}\dots l_n)$).

In addition to using capitalization information for unknown words, Brants (2000) also uses capitalization information for tagging known words, by adding a capitalization feature to each tag. Thus instead of computing $P(t_i|t_{i-1},t_{i-2})$ as in (5.44), he actually computes the probability $P(t_i,c_i|t_{i-1},c_{i-1},t_{i-2},c_{i-2})$. This is equivalent to having a capitalized and uncapitalized version of each tag, essentially doubling the size of the tagset.

A non-HMM based approach to unknown word detection was that of Brill (1995) using the TBL algorithm, where the allowable templates were defined orthographically (the first N letters of the words, the last N letters of the word, etc.).

Most recent approaches to unknown word handling, however, combine these features in a third way: by using maximum entropy (**MaxEnt**) models such as the **Maximum Entropy Markov Model** (**MEMM**) first introduced by Ratnaparkhi (1996) and McCallum et al. (2000), and which we will study in Ch. 6. The maximum entropy approach is one a family of loglinear approaches to classification in which many features are computed for the word to be tagged, and all the features are combined in a model based on multinomial logistic regression. The unknown word model in the tagger of Toutanova et al. (2003) uses a feature set extended from Ratnaparkhi (1996), in which each feature represents a property of a word, including features like:

word contains a number
word contains an upper-case letter
word contains a hyphen
word is all upper-case
word contains a particular prefix (from the set of all prefixes of length ≤ 4)
word contains a particular suffix (from the set of all prefixes of length ≤ 4)
word is upper-case and has a digit and a dash (like *CFC-12*)
word is upper-case and followed within 3 word by Co., Inc., etc

Toutanova et al. (2003) found this last feature, implementing a simple company name detector, to be particularly useful. 3 words by a word like Co. or Inc. Note that the Ratnaparkhi (1996) model ignored all features with counts less than 10.

Loglinear models have also been applied to Chinese tagging by Tseng et al. (2005). Chinese words are very short (around 2.4 characters per unknown word compared with 7.7 for English), but Tseng et al. (2005) found that morphological features nonetheless gave a huge increase in tagging performance for unknown words. For example for each character in an unknown word and each POS tag, they added a binary feature indicating whether that character ever occurred with that tag in any training set word. There is also an interesting distributional difference in unknown words between Chinese and English. While English unknown words tend to be proper nouns (41% of unknown words in WSJ are NP), in Chinese the majority of unknown words are common nouns and verbs (61% in the Chinese TreeBank 5.0). These ratios are similar to German, and seem to be caused by the prevalence of compounding as a morphological device in Chinese and German.

5.8.3 Part-of-Speech Tagging for Other Languages

As the previous paragraph suggests, part-of-speech tagging algorithms have all been applied to many other languages as well. In some cases, the methods work well without large modifications; Brants (2000) showed the exact same performance for tagging on the German NEGRA corpus (96.7%) as on the English Penn Treebank. But a number of augmentations and changes become necessary when dealing with highly inflected or agglutinative languages.

One problem with these languages is simply the large number of words, when compared to English. Recall from Ch. 3 that agglutinative languages like Turkish (and to some extent mixed agglutinative-inflectional languages like Hungarian) are those in which words contain long strings of morphemes, where each morpheme has relatively few surface forms, and so it is often possible to clearly see the morphemes in the surface text. For example Megyesi (1999) gives the following typical example of a Hungarian word meaning "of their hits":

(5.52) találataiknak

talál	*-at*	*-a*	*-i*	*-k*	*-nak*
hit/find	nominalizer	his	poss.plur	their	dat/gen

“of their hits”

Similarly, the following list, excerpted from Hakkani-Tür et al. (2002), shows a few of the words producible in Turkish from the root *uyu-*, ’sleep’:

uyuyorum	‘I am sleeping’	uyuyorsun	‘you are sleeping’
uyuduk	‘we slept’	uyumadan	‘without sleeping’
uyuman	‘your sleeping’	uyurken	‘while (somebody) is sleeping’
uyutmak	‘to cause someone to sleep’	uyutturmak	‘to cause someone to cause another person to sleep’

These productive word-formation processes result in a large vocabulary for these languages. Oravecz and Dienes (2002), for example, show that a quarter-million word corpus of English has about 19,000 different words (i.e. word types); the same size corpus of Hungarian has almost 50,000 different words. This problem continues even with much larger corpora; note in the table below on Turkish from Hakkani-Tür et al. (2002) that the vocabulary size of Turkish is far bigger than that of English and is growing faster than English even at 10 million words.

	Vocabulary Size	
Corpus Size	**Turkish**	**English**
1M words	106,547	33,398
10M words	417,775	97,734

The large vocabulary size seems to cause a significant degradation in tagging performance when the HMM algorithm is applied directly to agglutinative languages. For example Oravecz and Dienes (2002) applied the exact same HMM software (called ‘TnT’) that Brants (2000) used to achieve 96.7% on both English and German, and achieved only 92.88% on Hungarian. The performance on known words (98.32%) was comparable to English results; the problem was the performance on unknown words: 67.07% on Hungarian, compared to around 84-85% for unknown words with a comparable amount of English training data. Hajič (2000) notes the same problem in a wide variety of other languages (including Czech, Slovene, Estonian, and Romanian); the performance of these taggers is hugely improved by adding a dictionary which essentially gives a better model of unknown words. In summary, one difficulty in tagging highly inflected and agglutinative languages is tagging of unknown words.

A second, related issue with such languages is the vast amount of information that is coded in the morphology of the word. In English, lots of information about syntactic function of a word is represented by word order, or neighboring function words. In highly inflectional languages, information such as the case (nominative, accusative, genitive) or gender (masculine, feminine) is marked on the words themselves, and word order plays less of a role in marking syntactic function. Since tagging is often used a preprocessing step for other NLP algorithms such as parsing or information extraction, this morphological information is crucial to extract. This means that a part-of-speech tagging output for Turkish or Czech needs to include information about the case and gender of each word in order to be as useful as parts-of-speech without case or gender are in English.

For this reason, tagsets for agglutinative and highly inflectional languages are usually much larger than the 50-100 tags we have seen for English. Tags in such enriched tagsets are sequences of morphological tags rather than a single primitive tag. Assigning tags from such a tagset to words means that we are jointly solving the problems of part-of-speech tagging and morphological disambiguation. Hakkani-Tür et al. (2002) give the following example of tags from Turkish, in which the word *izin* has three possible morphological/part-of-speech tags (and meanings):

1. Yerdeki **izin** temizlenmesi gerek. iz + `Noun+A3sg+Pnon+Gen`
   The trace on the floor should be cleaned.

2. Üzerinde parmak **izin** kalmiş iz + `Noun+A3sg+P2sg+Nom`
   Your finger **print** is left on (it).

3. Içeri girmek için **izin** alman gerekiyor. izin + `Noun+A3sg+Pnon+Nom`
   You need a **permission** to enter.

Using a morphological parse sequence like `Noun+A3sg+Pnon+Gen` as the part-of-speech tag greatly increases the number of parts-of-speech, of course. We can see this clearly in the morphologically tagged MULTEXT-East corpora, in English, Czech, Estonian, Hungarian, Romanian, and Slovene (Dimitrova et al., 1998; Erjavec, 2004). Hajič (2000) gives the following tagset sizes for these corpora:

Language	Tagset Size
English	139
Czech	970
Estonian	476
Hungarian	401
Romanian	486
Slovene	1033

With such large tagsets, it is generally necessary to perform morphological analysis on each word to generate the list of possible morphological tag sequences (i.e. the list of possible part-of-speech tags) for the word. The role of the tagger is then to disambiguate among these tags. The morphological analysis can be done in various ways. The Hakkani-Tür et al. (2002) model of Turkish morphological analysis is based on the two-level morphology we introduced in Ch. 3. For Czech and the MULTEXT-East languages, Hajič (2000) and Hajič and Hladká (1998) use a fixed external dictionary for each language which compiles out all the possible forms of each word, and lists possible tags for each wordform. The morphological parse also crucially helps address the problem of unknown words, since morphological parsers can accept unknown stems and still segment the affixes properly.

Given such a morphological parse, various methods for the tagging itself can be used. The Hakkani-Tür et al. (2002) model for Turkish uses a Markov model of tag sequences. The model assigns a probability to sequences of tags like `izin+Noun+A3sg+Pnon+Nom` by computing tag transition probabilities from a training set. Other models use similar techniques to those for English. Hajič (2000) and Hajič and Hladká (1998), for example, use a log-linear exponential tagger for the MULTEXT-East languages, Oravecz and Dienes (2002) and Džeroski et al. (2000) use the TnT HMM tagger (Brants, 2000), and so on.

5.8.4 Combining Taggers

The various part-of-speech tagging algorithms we have described can also be combined. The most common approach to tagger combination is to run multiple taggers in parallel on the same sentence, and then combine their output, either by voting or by training another classifier to choose which tagger to trust in a given context. Brill and Wu (1998), for example, combined unigram, HMM, TBL, and maximum-entropy taggers by voting via a higher-order classifier, and showed a small gain over the best of the four classifiers. In general, this kind of combination is only useful if the taggers have complementary errors, and so research on combination often begins by checking to see if the errors are indeed different from different taggers. Another option is to combine taggers in series. Hajič et al. (2001) apply this option for Czech, using the rule-based approach to remove some of the impossible tag possibilities for each word, and then an HMM tagger to choose the best sequence from the remaining tags.

5.9 Advanced: The Noisy Channel Model for Spelling

The Bayesian inference model introduced in Sec. 5.5 for tagging has another interpretation: as an implementation of the **noisy channel** model, a crucial tool in speech recognition and machine translation.

In this section we introduce this noisy channel model and show how to apply it to the task of correcting spelling errors. The noisy channel model is used in Microsoft Word and in many search engines, and in general is the most widely used algorithm for correcting any kind of single-word spelling error, including **non-word spelling errors** and for **real-word spelling errors**.

Recall that non-word spelling errors are those which are not English words (like *recieve* for *receive*), and we can **detect** these by simply looking for any word not in a dictionary. We saw in Sec. **??** that candidate corrections for some spelling errors could be found by looking for words that had a small **edit distance** to the misspelled word.

The Bayesian models we have seen in this chapter, and the noisy channel model, will give us a better way to find these corrections. Furthermore, we'll be able to use the noisy channel model for **contextual spell checking**, which is the task of correcting **real-word spelling errors** like the following:

They are leaving in about fifteen *minuets* to go to her house.
The study was conducted mainly *be* John Black.

Since these errors have real words, we can't find them by just flagging words not in the dictionary, and we can't correct them just using edit distance alone. But note that words around the candidate correction *in about fifteen minutes* make it a much more probable word sequence than the original *in about fifteen minuets*. The noisy channel model will implement this idea via N-gram models.

NOISY CHANNEL

The intuition of the **noisy channel** model (see Fig. 5.23) is to treat the misspelled word as if a correctly-spelled word had been 'distorted' by being passed through a noisy communication channel. This channel introduces "noise" in the form of substitutions or other changes to the letters which makes it hard to recognize the "true" word. Our

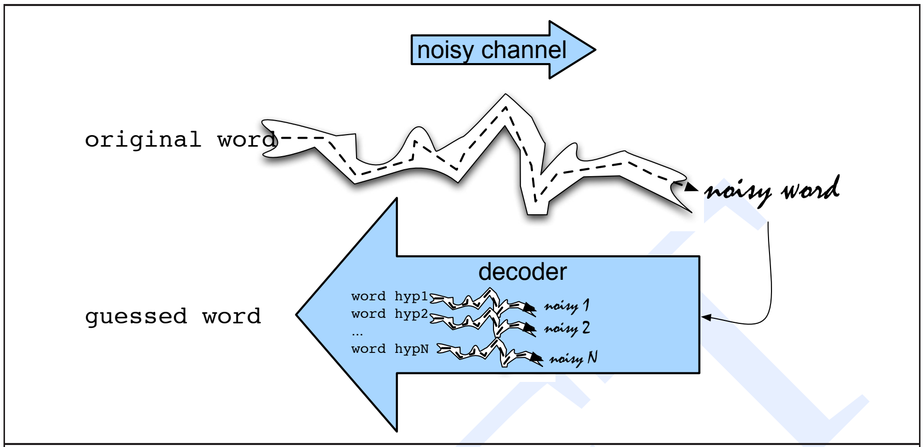


Figure 5.23 In the noisy channel model, we imagine that the surface form we see is actually a 'distorted' form of an original word passed through a noisy channel. The decoder passes each hypothesis through a model of this channel and picks the word that best matches the surface noisy word.

goal is then to build a model of the channel. Given this model, we then find the true word by taking every word of the language, passing each word through our model of the noisy channel, and seeing which one comes the closest to the misspelled word.

This noisy channel model, like the HMM tagging architecture we saw earlier, is a special case of **Bayesian inference**. We see an observation O (a misspelled word) and our job is to find the word w which generated this misspelled word. Out of all possible words in the vocabulary V we want to find the word w such that $P(w|O)$ is highest, or:

BAYESIAN

V

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(w|O) \tag{5.53}$$

As we saw for part-of-speech tagging, we will use Bayes rule to turn the problem around (and note that, as for tagging, we can ignore the denominator):

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} \frac{P(O|w)P(w)}{P(O)} = \underset{w \in V}{\operatorname{argmax}} P(O|w)\,P(w) \tag{5.54}$$

To summarize, the noisy channel model says that we have some true underlying word w, and we have a noisy channel which modifies the word into some possible misspelled surface form. The probability of the noisy channel producing any particular observation sequence O is modeled by $P(O|w)$. The probability distribution over possible hidden words is modeled by $P(w)$. The most probable word $\hat{w}$ given that we've seen some observed misspelling O can be computed by taking the product of the word prior $P(w)$ and the observation likelihood $P(O|w)$ and choosing the word for which this product is greatest.

Let's apply the noisy channel approach to correcting non-word spelling errors. This approach was first suggested by Kernighan et al. (1990); their program, `correct`, takes words rejected by the Unix `spell` program, generates a list of potential correct words, ranks them according to Eq. (5.54), and picks the highest-ranked one. We'll apply the algorithm to the example misspelling *acress*. The algorithm has two stages: *proposing candidate corrections* and *scoring the candidates*.

In order to propose candidate corrections Kernighan et al. make the reasonable (Damerau, 1964) simplifying assumption that the correct word will differ from the misspelling by a single insertion, deletion, substitution, or transposition. The list of candidate words is generated from the typo by applying any single transformation which results in a word in a large on-line dictionary. Applying all possible transformations to *acress* yields the list of candidate words in Fig. 5.24.

		Transformation			
		Correct	Error	Position	
Error	Correction	Letter	Letter	(Letter #)	Type
acress	actress	t	–	2	deletion
acress	cress	–	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
acress	across	o	e	3	substitution
acress	acres	–	2	5	insertion
acress	acres	–	2	4	insertion

Figure 5.24 Candidate corrections for the misspelling *acress*, together with the transformations that would have produced the error (after Kernighan et al. (1990)). "–" represents a null letter.

The second stage of the algorithm scores each correction by Equation 5.54. Let t represent the typo (the misspelled word), and let c range over the set C of candidate corrections. The most likely correction is then:

$$\hat{c} = \underset{c\in C}{\operatorname{argmax}} \; \overbrace{P(t|c)}^{\text{likelihood}} \; \overbrace{P(c)}^{\text{prior}} \tag{5.55}$$

The prior probability of each correction $P(c)$ is the language model probability of the word c in context; for in this section for pedagogical reasons we'll make the simplifying assumption that this is the unigram probability $P(c)$, but in practice in spelling correction this is extended to trigram or 4-gram probabilities. Let's use the corpus of Kernighan et al. (1990), which is the 1988 AP newswire corpus of 44 million words. Since in this corpus the word *actress* occurs 1343 times out of 44 million, the word *acres* 2879 times, and so on, the resulting unigram prior probabilities are as follows:

c	**freq(c)**	**p(c)**
actress	1343	.0000315
cress	0	.000000014
caress	4	.0000001
access	2280	.000058
across	8436	.00019
acres	2879	.000065

How can we estimate $P(t|c)$? It is very difficult to model the actual channel perfectly (i.e. computing the exact probability that a word will be mistyped) because it would require knowing who the typist was, whether they were left-handed or right-handed, and many other factors. Luckily, it turns out we can get a pretty reasonable estimate of $p(t|c)$ just by looking at simple local context factors, because the most important factors predicting an insertion, deletion, transposition are the identity of the correct letter itself, how the letter was misspelled, and the surrounding context. For example, the letters *m* and *n* are often substituted for each other; this is partly a fact about their identity (these two letters are pronounced similarly and they are next to each other on the keyboard), and partly a fact about context (because they are pronounced similarly, they occur in similar contexts). Kernighan et al. (1990) used a simple model of this sort. They estimated e.g. $p(acress|across)$ just using the number of times that the letter *e* was substituted for the letter *o* in some large corpus of errors. This is represented by a **confusion matrix**, a square 26×26 matrix which represents the number of times one letter was incorrectly used instead of another. For example, the cell labeled $[o,e]$ in a substitution confusion matrix would give the count of times that *e* was substituted for *o*. The cell labeled $[t,s]$ in an insertion confusion matrix would give the count of times that *t* was inserted after *s*. A confusion matrix can be computed by coding a collection of spelling errors with the correct spelling and then counting the number of times different errors occurred (Grudin, 1983). Kernighan et al. (1990) used four confusion matrices, one for each type of single error:

CONFUSION MATRIX

- del$[x,y]$ contains the number of times in the training set that the characters *xy* in the correct word were typed as *x*.
- ins$[x,y]$ contains the number of times in the training set that the character *x* in the correct word was typed as *xy*.
- sub$[x,y]$ the number of times that *x* was typed as *y*.
- trans$[x,y]$ the number of times that *xy* was typed as *yx*.

Note that they chose to condition their insertion and deletion probabilities on the previous character; they could also have chosen to condition on the following character. Using these matrices, they estimated $p(t|c)$ as follows (where c_p is the *p*th character of the word *c*):

$$P(t|c) = \begin{cases} \frac{\mathrm{del}[c_{p-1},c_p]}{\mathrm{count}[c_{p-1}c_p]}, \text{ if deletion} \\ \frac{\mathrm{ins}[c_{p-1},t_p]}{\mathrm{count}[c_{p-1}]}, \text{ if insertion} \\ \frac{\mathrm{sub}[t_p,c_p]}{\mathrm{count}[c_p]}, \text{ if substitution} \\ \frac{\mathrm{trans}[c_p,c_{p+1}]}{\mathrm{count}[c_p c_{p+1}]}, \text{ if transposition} \end{cases} \tag{5.56}$$

Fig. 5.25 shows the final probabilities for each of the potential corrections; the unigram prior is multiplied by the likelihood (computed using Equation (5.56) and the confusion matrices). The final column shows the "normalized percentage".

This implementation of the Bayesian algorithm predicts *acres* as the correct word (at a total normalized percentage of 45%), and *actress* as the second most likely word.

c	**freq(c)**	**p(c)**	**p(t|c)**	**p(t|c)p(c)**	**%**
actress	1343	.0000315	.000117	3.69×10^{-9}	**37%**
cress	0	.000000014	.00000144	2.02×10^{-14}	**0%**
caress	4	.0000001	.00000164	1.64×10^{-13}	**0%**
access	2280	.000058	.000000209	1.21×10^{-11}	**0%**
across	8436	.00019	.0000093	1.77×10^{-9}	**18%**
acres	2879	.000065	.0000321	2.09×10^{-9}	**21%**
acres	2879	.000065	.0000342	2.22×10^{-9}	**23%**

Figure 5.25 Computation of the ranking for each candidate correction. Note that the highest ranked word is not *actress* but *acres* (the two lines at the bottom of the table), since *acres* can be generated in two ways. The *del*[], *ins*[], *sub*[], and *trans*[] confusion matrices are given in full in Kernighan et al. (1990).

Unfortunately, the algorithm was wrong here: The writer's intention becomes clear from the context: . . . *was called a "stellar and versatile* **acress** *whose combination of sass and glamour has defined her. . .* ". The surrounding words make it clear that *actress* and not *acres* was the intended word. This is the reason that in practice we use trigram (or larger) language models in the noisy channel model, rather thaan unigrams. Seeing whether a **bigram** model of $P(c)$ correctly solves this problem is left as Exercise 5.10 for the reader.

The algorithm as we have described it requires hand-annotated data to train the confusion matrices. An alternative approach used by Kernighan et al. (1990) is to compute the matrices by iteratively using this very spelling error correction algorithm itself. The iterative algorithm first initializes the matrices with equal values; thus any character is equally likely to be deleted, equally likely to be substituted for any other character, etc. Next the spelling error correction algorithm is run on a set of spelling errors. Given the set of typos paired with their corrections, the confusion matrices can now be recomputed, the spelling algorithm run again, and so on. This clever method turns out to be an instance of the important **EM** algorithm (Dempster et al., 1977) that we will discuss in Ch. 6.

5.9.1 Contextual Spelling Error Correction

REAL-WORD ERROR DETECTION

CONTEXT-SENSITIVE SPELL CORRECTION

As we mentioned above, the noisy channel approach can also be applied to detect and correct **real-word spelling errors,** errors that result in an actual word of English. This can happen from typographical errors (insertion, deletion, transposition) that accidentally produce a real word (e.g., *there* for *three*), or because the writer substituted the wrong spelling of a homophone or near-homophone (e.g., *dessert* for *desert*, or *piece* for *peace*). The task of correcting these errors is also called **context-sensitive spell correction**. A number of studies suggest that between of 25% and 40% of spelling errors are valid English words (Kukich, 1992); some of Kukich's examples include:

They are leaving in about fifteen *minuets* to go to her house.
The design *an* construction of the system will take more than a year.
Can they *lave* him my messages?
The study was conducted mainly *be* John Black.

We can extend the noisy channel model to deal with real-word spelling errors by generating a *candidate spelling set* for every word in a sentence (Mays et al., 1991). The candidate set includes the word itself, plus every English word that would be generated from the word by either typographical modifications (letter insertion, deletion, substitution), or from a homophone list. The algorithm then chooses the spelling for each word that gives the whole sentence the highest probability. That is, given a sentence $W = \{w_1, w_2, \ldots, w_k, \ldots, w_n\}$, where w_k has alternative spelling w'_k, w''_k, etc., we choose the spelling among these possible spellings that maximizes $P(W)$, using the N-gram grammar to compute $P(W)$.

More recent research has focused on improving the channel model $P(t|c)$, such as by incorporating phonetic information, or allowing more complex errors (Brill and Moore, 2000; Toutanova and Moore, 2002). The most important improvement to the language model $P(c)$ is to use very large contexts, for example by using the very large set of 5-grams publicly released by Google in 2006 (Franz and Brants, 2006). See Norvig (2007) for a nice explanation and Python implementation of the noisy channel model; the end of the chapter has further pointers.

5.10 Summary

This chapter introduced the idea of **parts-of-speech** and **part-of-speech tagging**. The main ideas:

- Languages generally have a relatively small set of **closed class** words, which are often highly frequent, generally act as **function words**, and can be very ambiguous in their part-of-speech tags. Open class words generally include various kinds of **nouns**, **verbs**, **adjectives**. There are a number of part-of-speech coding schemes, based on **tagsets** of between 40 and 200 tags.
- **Part-of-speech tagging** is the process of assigning a part-of-speech label to each of a sequence of words. Rule-based taggers use hand-written rules to distinguish tag ambiguity. HMM taggers choose the tag sequence which maximizes the product of word likelihood and tag sequence probability. Other machine learning models used for tagging include maximum entropy and other log-linear models, decision trees, memory-based learning, and transformation-based learning.
- The probabilities in HMM taggers are trained on hand-labeled training corpora, combining different N-gram levels using deleted interpolation, and using sophisticated unknown word models.
- Given an HMM and an input string, the Viterbi algorithm is used to decode the optimal tag sequence.

- Taggers are evaluated by comparing their output from a test set to human labels for that test set. Error analysis can help pinpoint areas where a tagger doesn't perform well.

Bibliographical and Historical Notes

The earliest implemented part-of-speech assignment algorithm may have been part of the parser in Zellig Harris's Transformations and Discourse Analysis Project (TDAP), which was implemented between June 1958 and July 1959 at the University of Pennsylvania (Harris, 1962). Previous natural language processing systems had used dictionaries with part-of-speech information for words, but have not been described as performing part-of-speech disambiguation. As part of its parsing, TDAP did part-of-speech disambiguation via 14 hand-written rules, whose use of part-of-speech tag sequences prefigures all the modern algorithms, and which were run in an order based on the relative frequency of tags for a word. The parser/tagger was reimplemented recently and is described by Joshi and Hopely (1999) and Karttunen (1999), who note that the parser was essentially implemented (in a very modern way) as a cascade of finite-state transducers.

Soon after the TDAP parser was the Computational Grammar Coder (CGC) of Klein and Simmons (1963). The CGC had three components: a lexicon, a morphological analyzer, and a context disambiguator. The small 1500-word lexicon included exceptional words that could not be accounted for in the simple morphological analyzer, including function words as well as irregular nouns, verbs, and adjectives. The morphological analyzer used inflectional and derivational suffixes to assign part-of-speech classes. A word was run through the lexicon and morphological analyzer to produce a candidate set of parts-of-speech. A set of 500 context rules were then used to disambiguate this candidate set, by relying on surrounding islands of unambiguous words. For example, one rule said that between an ARTICLE and a VERB, the only allowable sequences were ADJ-NOUN, NOUN-ADVERB, or NOUN-NOUN. The CGC algorithm reported 90% accuracy on applying a 30-tag tagset to articles from the Scientific American and a children's encyclopedia.

The TAGGIT tagger (Greene and Rubin, 1971) was based on the Klein and Simmons (1963) system, using the same architecture but increasing the size of the dictionary and the size of the tagset (to 87 tags). For example the following sample rule, which states that a word x is unlikely to be a plural noun (NNS) before a third person singular verb (VBZ):

$$x \text{ VBZ} \rightarrow \textit{not} \text{ NNS}$$

TAGGIT was applied to the Brown corpus and, according to Francis and Kučera (1982, p. 9), "resulted in the accurate tagging of 77% of the corpus" (the remainder of the Brown corpus was tagged by hand).

In the 1970s the Lancaster-Oslo/Bergen (LOB) corpus was compiled as a British English equivalent of the Brown corpus. It was tagged with the CLAWS tagger (Marshall, 1983, 1987; Garside, 1987), a probabilistic algorithm which can be viewed as an approximation to the HMM tagging approach. The algorithm used tag bigram probabilities, but instead of storing the word-likelihood of each tag, tags were marked either as *rare* ($P(\text{tag}|\text{word}) < .01$) *infrequent* ($P(\text{tag}|\text{word}) < .10$), or *normally frequent* ($P(\text{tag}|\text{word}) > .10$),

The probabilistic PARTS tagger of Church (1988) was very close to a full HMM tagger. It extended the CLAWS idea to assign full lexical probabilities to each word/tag combination, and used Viterbi decoding to find a tag sequence. Like the CLAWS tagger, however, it stored the probability of the tag given the word:

$$P(\text{tag}|\text{word}) * P(\text{tag}|\text{previous } n \text{ tags}) \tag{5.57}$$

rather than using the probability of the word given the tag, as an HMM tagger does:

$$P(\text{word}|\text{tag}) * P(\text{tag}|\text{previous } n \text{ tags}) \tag{5.58}$$

Later taggers explicitly introduced the use of the Hidden Markov Model, often with the EM training algorithm (Kupiec, 1992; Merialdo, 1994; Weischedel et al., 1993), including the use of variable-length Markov models (Schütze and Singer, 1994).

Most recent tagging algorithms, like the HMM and TBL approaches we have discussed, are machine-learning classifiers which estimate the best tag-sequence for a sentence given various features such as the current word, neighboring parts-of-speech or words, and unknown word features such as orthographic and morphological features. Many kinds of classifiers have been used to combine these features, including decision trees (Jelinek et al., 1994; Magerman, 1995), maximum entropy models (Ratnaparkhi, 1996), other log-linear models (Franz, 1996), memory-based learning (Daelemans et al., 1996), and networks of linear separators (SNOW) (Roth and Zelenko, 1998). Most machine learning models seem to achieve relatively similar performance given similar features, roughly 96-97% on the Treebank 45-tag tagset on the Wall Street Journal corpus. As of the writing of this chapter, the highest performing published model on this WSJ Treebank task is a log-linear tagger that uses information about neighboring words as well as tags, and a sophisticated unknown-word model, achieving 97.24% accuracy (Toutanova et al., 2003). Most such models are supervised, although there is beginning to be work on unsupervised models (Schütze, 1995; Brill, 1997; Clark, 2000; Banko and Moore, 2004; Goldwater and Griffiths, 2007).

Readers interested in the history of parts-of-speech should consult a history of linguistics such as Robins (1967) or Koerner and Asher (1995), particularly the article by Householder (1995) in the latter. Sampson (1987) and Garside et al. (1997) give a detailed summary of the provenance and makeup of the Brown and other tagsets. More information on part-of-speech tagging can be found in van Halteren (1999).

Algorithms for spelling error detection and correction have existed since at least Blair (1960). Most early algorithm were based on similarity keys like the Soundex algorithm discussed in the exercises on page **??** (Odell and Russell, 1922; Knuth, 1973). Damerau (1964) gave a dictionary-based algorithm for error detection; most error-detection algorithms since then have been based on dictionaries. Damerau also gave a correction algorithm that worked for single errors. Most algorithms since then have relied on dynamic programming, beginning with Wagner and Fischer (1974). Kukich (1992) is the definitive survey article on spelling error detection and correction.

Modern algorithms are based on statistical or machine learning algorithm, following e.g., Kashyap and Oommen (1983) and Kernighan et al. (1990). Recent approaches to spelling include extensions to the noisy channel model (Brill and Moore, 2000; Toutanova and Moore, 2002) as well as many other machine learning architectures such as Bayesian classifiers, (Gale et al., 1993; Golding, 1997; Golding and Schabes, 1996), decision lists (Yarowsky, 1994), transformation based learning (Mangu and Brill, 1997) latent semantic analysis (Jones and Martin, 1997) and Winnow (Golding and Roth, 1999). Hirst and Budanitsky (2005) explore the use of word relatedness; see Ch. 20. Noisy channel spelling correction is used in a number of commercial applications, including the Microsoft Word contextual spell checker.

Exercises

5.1 Find one tagging error in each of the following sentences that are tagged with the Penn Treebank tagset:

a. I/PRP need/VBP a/DT flight/NN from/IN Atlanta/NN
b. Does/VBZ this/DT flight/NN serve/VB dinner/NNS
c. I/PRP have/VB a/DT friend/NN living/VBG in/IN Denver/NNP
d. What/WDT flights/NNS do/VBP you/PRP have/VB from/IN Milwaukee/NNP to/IN Tampa/NNP
e. Can/VBP you/PRP list/VB the/DT nonstop/JJ afternoon/NN flights/NNS

5.2 Use the Penn Treebank tagset to tag each word in the following sentences from Damon Runyon's short stories. You may ignore punctuation. Some of these are quite difficult; do your best.

a. It is a nice night.
b. This crap game is over a garage in Fifty-second Street...
c. ...Nobody ever takes the newspapers she sells ...
d. He is a tall, skinny guy with a long, sad, mean-looking kisser, and a mournful voice.
e. ...I am sitting in Mindy's restaurant putting on the gefillte fish, which is a dish I am very fond of, ...
f. When a guy and a doll get to taking peeks back and forth at each other, why there you are indeed.

5.3 Now compare your tags from the previous exercise with one or two friend's answers. On which words did you disagree the most? Why?

5.4 Now tag the sentences in Exercise 5.2 using the more detailed Brown tagset in Fig. 5.7.

5.5 Implement the TBL algorithm in Fig. 5.21. Create a small number of templates and train the tagger on any POS-tagged training set you can find.

5.6 Implement the "most-likely tag" baseline. Find a POS-tagged training set, and use it to compute for each word the tag which maximizes $p(t|w)$. You will need to implement a simple tokenizer to deal with sentence boundaries. Start by assuming all unknown words are NN and compute your error rate on known and unknown words. Now write at least 5 rules to do a better job of tagging unknown words, and show the difference in error rates.

5.7 Recall that the Church (1988) tagger is not an HMM tagger since it incorporates the probability of the tag given the word:

$$P(\text{tag}|\text{word}) * P(\text{tag}|\text{previous } n \text{ tags}) \tag{5.59}$$

rather than using the likelihood of the word given the tag, as an HMM tagger does:

$$P(\text{word}|\text{tag}) * P(\text{tag}|\text{previous } n \text{ tags}) \tag{5.60}$$

As a gedanken-experiment, construct a sentence, a set of tag transition probabilities, and a set of lexical tag probabilities that demonstrate a way in which the HMM tagger can produce a better answer than the Church tagger, and another example in which the Church tagger is better.

5.8 Build an HMM tagger. This requires (1) that you have implemented the Viterbi algorithm from this chapter and Ch. 6, (2) that you have a dictionary with part-of-speech information and (3) that you have either (a) a part-of-speech-tagged corpus or (b) an implementation of the Forward Backward algorithm. If you have a labeled corpus, train the transition and observation probabilities of an HMM tagger directly on the hand-tagged data. If you have an unlabeled corpus, train using Forward Backward.

5.9 Now run your algorithm on a small test set that you have hand-labeled. Find five errors and analyze them.

5.10 Compute a bigram grammar on a large corpus and reestimate the spelling correction probabilities shown in Fig. 5.25 given the correct sequence *... was called a "stellar and versatile* **acress** *whose combination of sass and glamour has defined her...* ". Does a bigram grammar prefer the correct word *actress*?

5.11 Read Norvig (2007) and implement one of the extensions he suggests to his Python noisy channel spell checker.

Baayen, R. H., Piepenbrock, R., and Gulikers, L. (1995). *The CELEX Lexical Database (Release 2) [CD-ROM]*. Linguistic Data Consortium, University of Pennsylvania [Distributor], Philadelphia, PA.

Baayen, R. H. and Sproat, R. (1996). Estimating lexical priors for low-frequency morphologically ambiguous forms. *Computational Linguistics*, *22*(2), 155–166.

Bahl, L. R. and Mercer, R. L. (1976). Part of speech assignment by a statistical decision algorithm. In *Proceedings IEEE International Symposium on Information Theory*, pp. 88–89.

Banko, M. and Moore, R. C. (2004). A study of unsupervised part-of-speech tagging. In *COLING-04*.

Bayes, T. (1763). *An Essay Toward Solving a Problem in the Doctrine of Chances*, Vol. 53. Reprinted in *Facsimiles of two papers by Bayes*, Hafner Publishing Company, New York, 1963.

Blair, C. R. (1960). A program for correcting spelling errors. *Information and Control*, *3*, 60–67.

Brants, T. (2000). TnT: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, Seattle, WA, pp. 224–231. Morgan Kaufmann.

Brill, E. (1995). Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, *21*(4), 543–566.

Brill, E. (1997). Unsupervised learning of disambiguation rules for part of speech tagging. Unpublished manuscript.

Brill, E. and Moore, R. C. (2000). An improved error model for noisy channel spelling correction. In *ACL-00*, Hong Kong, pp. 286–293.

Brill, E. and Wu, J. (1998). Classifier combination for improved lexical disambiguation. In *COLING/ACL-98*, Montreal, Canada, pp. 191–195.

Broschart, J. (1997). Why Tongan does it differently. *Linguistic Typology*, *1*, 123–165.

Charniak, E., Hendrickson, C., Jacobson, N., and Perkowitz, M. (1993). Equations for part-of-speech tagging. In *AAAI-93*, Washington, D.C., pp. 784–789. AAAI Press.

Church, K. W. (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *Second Conference on Applied Natural Language Processing*, pp. 136–143. ACL.

Clark, A. (2000). Inducing syntactic categories by context distribution clustering. In *CoNLL-00*.

Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press.

Cutler, A. (1986). Forbear is a homophone: Lexical prosody does not constrain lexical access. *Language and Speech*, *29*, 201–219.

Cutting, D., Kupiec, J., Pedersen, J., and Sibun, P. (1992). A practical part-of-speech tagger. In *Third Conference on Applied Natural Language Processing*, pp. 133–140.

Daelemans, W., Zavrel, J., Berck, P., and Gillis, S. (1996). MBT: A memory based part of speech tagger-generator. In Ejerhed, E. and Dagan, I. (Eds.), *Proceedings of the Fourth Workshop on Very Large Corpora*, pp. 14–27.

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, *7*(3), 171–176.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the *EM* algorithm. *Journal of the Royal Statistical Society*, *39*(1), 1–21.

Dermatas, E. and Kokkinakis, G. (1995). Automatic stochastic tagging of natural language texts. *Computational Linguistics*, *21*(2), 137–164.

DeRose, S. J. (1988). Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, *14*, 31–39.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, *10*(7), 1895–1924.

Dimitrova, L., Ide, N., Petkevic, V., Erjavec, T., Kaalep, H. J., and Tufis, D. (1998). Multext-East: parallel and comparable corpora and lexicons for six Central and Eastern European languages. In *COLING/ACL-98*, Montreal, Canada.

Džeroski, S., Erjavec, T., and Zavrel, J. (2000). Morphosyntactic Tagging of Slovene: Evaluating PoS Taggers and Tagsets. In *LREC-00*, Paris, pp. 1099–1104.

Erjavec, T. (2004). Multext-east version 3: Multilingual morphosyntactic specifications, lexicons and corpora. In *LREC-04*, pp. 1535–1538. ELRA.

Evans, N. (2000). Word classes in the world's languages. In Booij, G., Lehmann, C., and Mugdan, J. (Eds.), *Morphology: a Handbook on Inflection and Word Formation*, pp. 708–732. Mouton, Berlin.

Francis, W. N. (1979). A tagged corpus – problems and prospects. In Greenbaum, S., Leech, G., and Svartvik, J. (Eds.), *Studies in English linguistics for Randolph Quirk*, pp. 192–209. Longman.

Francis, W. N. and Kučera, H. (1982). *Frequency Analysis of English Usage*. Houghton Mifflin, Boston.

Franz, A. and Brants, T. (2006). All our n-gram are belong to you. `http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html`.

Franz, A. (1996). *Automatic Ambiguity Resolution in Natural Language Processing*. Springer-Verlag, Berlin.

Gale, W. A., Church, K. W., and Yarowsky, D. (1992). Estimating upper and lower bounds on the performance of word-sense disambiguation programs. In *Proceedings of the 30th ACL*, Newark, DE, pp. 249–256.

Gale, W. A., Church, K. W., and Yarowsky, D. (1993). A method for disambiguating word senses in a large corpus. *Computers and the Humanities*, *26*, 415–439.

Garside, R. (1987). The CLAWS word-tagging system. In Garside, R., Leech, G., and Sampson, G. (Eds.), *The Computational Analysis of English*, pp. 30–41. Longman, London.

Garside, R., Leech, G., and McEnery, A. (1997). *Corpus Annotation*. Longman, London and New York.

Gil, D. (2000). Syntactic categories, cross-linguistic variation and universal grammar. In Vogel, P. M. and Comrie, B. (Eds.), *Approaches to the Typology of Word Classes*, pp. 173–216. Mouton.

Golding, A. R. and Roth, D. (1999). A winnow based approach to context-sensitive spelling correction. *Machine Learning*, *34*(1-3), 107–130. Special Issue on Machine Learning and Natural Language.

Golding, A. R. (1997). A bayesian hybrid method for context-sensitive spelling correction. In *Proceedings of the Third Workshop on Very Large Corpora*, Boston, MA, pp. 39–53.

Golding, A. R. and Schabes, Y. (1996). Combining trigram-based and feature-based methods for context-sensitive spelling COrrection. In *ACL-96*, Santa Cruz, CA, pp. 71–78.

Goldwater, S. and Griffiths, T. L. (2007). A fully Bayesian approach to unsupervised part-of-speech tagging. In *ACL-07*, Prague, Czech Republic.

Greene, B. B. and Rubin, G. M. (1971). Automatic grammatical tagging of English. Department of Linguistics, Brown University, Providence, Rhode Island.

Grudin, J. T. (1983). Error patterns in novice and skilled transcription typing. In Cooper, W. E. (Ed.), *Cognitive Aspects of Skilled Typewriting*, pp. 121–139. Springer-Verlag.

Hajič, J. (2000). Morphological tagging: Data vs. dictionaries. In *Proceedings of ANLP-NAACL Conference*. Seattle.

Hajič, J. and Hladká, B. (1998). Tagging inflective languages: Prediction of morphological categories for a rich, structured tagset. In *COLING/ACL-98*, Montreal, Canada.

Hajič, J., Krbec, P., Květoň, P., Oliva, K., and Petkevič, V. (2001). Serial Combination of Rules and Statistics: A Case Study in Czech Tagging. In *ACL-01*, Toulouse, France, pp. –.

Hakkani-Tür, D., Oflazer, K., and Tür, G. (2002). Statistical morphological disambiguation for agglutinative languages. *Journal of Computers and Humanities*, *36*(4).

Harris, Z. S. (1962). *String Analysis of Sentence Structure*. Mouton, The Hague.

Heikkilä, J. (1995). A TWOL-based lexicon and feature system for English. In Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (Eds.), *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*, pp. 103–131. Mouton de Gruyter, Berlin.

Hirst, G. and Budanitsky, A. (2005). Correcting real-word spelling errors by restoring lexical cohesion. *Natural Language Engineering*, *11*, 87–111.

Householder, F. W. (1995). Dionysius Thrax, the *technai*, and Sextus Empiricus. In Koerner, E. F. K. and Asher, R. E. (Eds.), *Concise History of the Language Sciences*, pp. 99–103. Elsevier Science, Oxford.

Jelinek, F., Lafferty, J. D., Magerman, D. M., Mercer, R. L., Ratnaparkhi, A., and Roukos, S. (1994). Decision tree parsing using a hidden derivation model. In *ARPA Human Language Technologies Workshop*, Plainsboro, N.J., pp. 272–277. Morgan Kaufmann.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In Gelsema, E. S. and Kanal, L. N. (Eds.), *Proceedings, Workshop on Pattern Recognition in Practice*, pp. 381–397. North Holland, Amsterdam.

Jones, M. P. and Martin, J. H. (1997). Contextual spelling correction using latent semantic analysis. In *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP'97)*, Washington, D.C., pp. 166–173.

Joshi, A. K. and Hopely, P. (1999). A parser from antiquity. In Kornai, A. (Ed.), *Extended Finite State Models of Language*, pp. 6–15. Cambridge University Press.

Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (1995a). *Constraint Grammar — A language-independent system for parsing unrestricted text*. Mouton de Gruyter, Berlin.

Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (Eds.). (1995b). *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, Berlin.

Karttunen, L. (1999). Comments on Joshi. In Kornai, A. (Ed.), *Extended Finite State Models of Language*, pp. 16–18. Cambridge University Press.

Kashyap, R. L. and Oommen, B. J. (1983). Spelling correction using probabilistic methods. *Pattern Recognition Letters*, *2*, 147–154.

Kernighan, M. D., Church, K. W., and Gale, W. A. (1990). A spelling correction program base on a noisy channel model. In *COLING-90*, Helsinki, Vol. II, pp. 205–211.

Klein, S. and Simmons, R. F. (1963). A computational approach to grammatical coding of English words. *Journal of the Association for Computing Machinery*, *10*(3), 334–347.

Knuth, D. E. (1973). *Sorting and Searching: The Art of Computer Programming Volume 3*. Addison-Wesley, Reading, MA.

Koerner, E. F. K. and Asher, R. E. (Eds.). (1995). *Concise History of the Language Sciences*. Elsevier Science, Oxford.

Kruskal, J. B. (1983). An overview of sequence comparison. In Sankoff, D. and Kruskal, J. B. (Eds.), *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pp. 1–44. Addison-Wesley, Reading, MA.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, *24*(4), 377–439.

Kupiec, J. (1992). Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, *6*, 225–242.

Kučera, H. and Francis, W. N. (1967). *Computational analysis of present-day American English*. Brown University Press, Providence, RI.

Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *ACL-95*, pp. 276–283.

Mangu, L. and Brill, E. (1997). Automatic rule acquisition for spelling correction. In *ICML 1997*, Nashville, TN, pp. 187–194.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, *19*(2), 313–330.

Marshall, I. (1983). Choice of grammatical word-class without GLobal syntactic analysis: Tagging words in the LOB corpus. *Computers and the Humanities*, *17*, 139–150.

Marshall, I. (1987). Tag selection using probabilistic methods. In Garside, R., Leech, G., and Sampson, G. (Eds.), *The Computational Analysis of English*, pp. 42–56. Longman, London.

Mays, E., Damerau, F. J., and Mercer, R. L. (1991). Context based spelling correction. *Information Processing and Management*, *27*(5), 517–522.

McCallum, A., Freitag, D., and Pereira, F. C. N. (2000). Maximum Entropy Markov Models for Information Extraction and Segmentation. In *ICML 2000*, pp. 591–598.

Megyesi, B. (1999). Improving brill's pos tagger for an agglutinative language. In *EMNLP/VLC-99*, College Park, MA.

Merialdo, B. (1994). Tagging English text with a probabilistic model. *Computational Linguistics*, *20*(2), 155–172.

Mosteller, F. and Wallace, D. L. (1964). *Inference and Disputed Authorship: The Federalist*. Springer-Verlag. A second edition appeared in 1984 as *Applied Bayesian and Classical Inference*.

Norvig, P. (2007). How to write a spelling corrector. `http://www.norvig.com/spell-correct.html`.

Odell, M. K. and Russell, R. C. (1918/1922). U.S. Patents 1261167 (1918), 1435663 (1922)†. Cited in Knuth (1973).

Oravecz, C. and Dienes, P. (2002). Efficient stochastic part-of-speech tagging for Hungarian. In *LREC-02*, Las Palmas, Canary Islands, Spain, pp. 710–717.

Quirk, R., Greenbaum, S., Leech, G., and Svartvik, J. (1985). *A Comprehensive Grammar of the English Language*. Longman, London.

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. In *EMNLP 1996*, Philadelphia, PA, pp. 133–142.

Robins, R. H. (1967). *A Short History of Linguistics*. Indiana University Press, Bloomington.

Roche, E. and Schabes, Y. (1997). Deterministic part-of-speech tagging with finite-state transducers. In Roche, E. and Schabes, Y. (Eds.), *Finite-State Language Processing*, pp. 205–239. MIT Press.

Roth, D. and Zelenko, D. (1998). Part of speech tagging using a network of linear separators. In *COLING/ACL-98*, Montreal, Canada, pp. 1136–1142.

Sampson, G. (1987). Alternative grammatical coding systems. In Garside, R., Leech, G., and Sampson, G. (Eds.), *The Computational Analysis of English*, pp. 165–183. Longman, London and New York.

Samuel, K., Carberry, S., and Vijay-Shanker, K. (1998). Computing dialogue acts from features with transformation-based learning. In Chu-Carroll, J. and Green, N. (Eds.), *Applying Machine Learning to Discourse Processing. Papers from the 1998 AAAI Spring Symposium*, pp. 90–97. Technical Report SS-98-01.

Samuelsson, C. (1993). Morphological tagging based entirely on Bayesian inference. In *9th Nordic Conference on Computational Linguistics NODALIDA-93*. Stockholm.

Santorini, B. (1990). Part-of-speech tagging guidelines for the Penn Treebank project. 3rd revision, 2nd printing.

Schachter, P. (1985). Parts-of-speech systems. In Shopen, T. (Ed.), *Language Typology and Syntactic Description, Volume 1*, pp. 3–61. Cambridge University Press.

Schütze, H. (1995). Distributional part-of-speech tagging. In *EACL-95*.

Schütze, H. and Singer, Y. (1994). Part-of-speech tagging using a variable memory Markov model. In *Proceedings of the 32nd ACL*, Las Cruces, NM, pp. 181–187.

Stolz, W. S., Tannenbaum, P. H., and Carstensen, F. V. (1965). A stochastic approach to the grammatical coding of English. *Communications of the ACM*, *8*(6), 399–405.

Thede, S. M. and Harper, M. P. (1999). A second-order Hidden Markov Model for part-of-speech tagging. In *ACL-99*, College Park, MA, pp. 175–182.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL-03*.

Toutanova, K. and Moore, R. C. (2002). Pronunciation modeling for improved spelling correction. In *ACL-02*, Philadelphia, PA, pp. 144–151.

Tseng, H., Jurafsky, D., and Manning, C. D. (2005). Morphological features help pos tagging of unknown words across language varieties. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*.

van Halteren, H. (Ed.). (1999). *Syntactic Wordclass Tagging*. Kluwer, Dordrecht.

Vintsyuk, T. K. (1968). Speech discrimination by dynamic programming. *Cybernetics*, *4*(1), 52–57. Russian Kibernetika 4(1):81-88 (1968).

Voutilainen, A. (1995). Morphological disambiguation. In Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (Eds.), *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*, pp. 165–284. Mouton de Gruyter, Berlin.

Voutilainen, A. (1999). Handcrafted rules. In van Halteren, H. (Ed.), *Syntactic Wordclass Tagging*, pp. 217–246. Kluwer, Dordrecht.

Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery*, *21*, 168–173.

Weischedel, R., Meteer, M., Schwartz, R., Ramshaw, L. A., and Palmucci, J. (1993). Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, *19*(2), 359–382.

Yarowsky, D. (1994). Decision lists for lexical ambiguity resolution: Application to accent restoration in Spanish and French. In *Proceedings of the 32nd ACL*, Las Cruces, NM, pp. 88–95. ACL.

6 HIDDEN MARKOV AND MAXIMUM ENTROPY MODELS

Numquam ponenda est pluralitas sine necessitat
'Plurality should never be proposed unless needed'
William of Occam

Her sister was called Tatiana.
For the first time with such a name
the tender pages of a novel,
we'll whimsically grace.
Pushkin, *Eugene Onegin*, in the Nabokov translation

Alexander Pushkin's novel in verse, *Eugene Onegin*, serialized in the early 19th century, tells of the young dandy Onegin, his rejection of the love of young Tatiana, his duel with his friend Lenski, and his later regret for both mistakes. But the novel is mainly beloved for its style and structure rather than its plot. Among other interesting structural innovations, the novel is written in a form now known as the *Onegin stanza*, iambic tetrameter with an unusual rhyme scheme. These elements have caused complications and controversy in its translation into other languages. Many of the translations have been in verse, but Nabokov famously translated it strictly literally into English prose. The issue of its translation, and the tension between literal and verse translations have inspired much commentary (see for example Hofstadter (1997)).

In 1913 A. A. Markov asked a less controversial question about Pushkin's text: could we use frequency counts from the text to help compute the probability that the next letter in sequence would be a vowel. In this chapter we introduce two important classes of statistical models for processing text and speech, both descendants of Markov's models. One of them is the **Hidden Markov Model** (**HMM**). The other, is the **Maximum Entropy** model (**MaxEnt**), and particularly a Markov-related variant of MaxEnt called the **Maximum Entropy Markov Model** (**MEMM**). All of these are **machine learning** models. We have already touched on some aspects of machine learning; indeed we briefly introduced the Hidden Markov Model in the previous chapter, and we have introduced the *N*-gram model in the chapter before. In this chapter we give a more complete and formal introduction to these two important models.

SEQUENCE CLASSIFIERS

HMMs and MEMMs are both **sequence classifiers**. A sequence classifier or **sequence labeler** is a model whose job is to assign some label or class to each unit in a sequence. The finite-state transducer we studied in Ch. 3 is a kind of non-probabilistic sequence classifier, for example transducing from sequences of words to sequences of morphemes. The HMM and MEMM extend this notion by being probabilistic sequence classifiers; given a sequence of units (words, letters, morphemes, sentences, whatever) their job is to compute a probability distribution over possible labels and choose the best label sequence.

We have already seen one important sequence classification task: part-of-speech tagging, where each word in a sequence has to be assigned a part-of-speech tag. Sequence labeling tasks come up throughout speech and language processing, a fact that isn't too surprising if we consider that language consists of sequences at many representational levels. Besides part-of-speech tagging, in this book we will see the application of these sequence models to tasks like speech recognition (Ch. 9), sentence segmentation and grapheme-to-phoneme conversion (Ch. 8), partial parsing/chunking (Ch. 13), and named entity recognition and information extraction (Ch. 22).

This chapter is roughly divided into two sections: Hidden Markov Models followed by Maximum Entropy Markov Models. Our discussion of the Hidden Markov Model extends what we said about HMM part-of-speech tagging. We begin in the next section by introducing the Markov Chain, then give a detailed overview of HMMs and the forward and Viterbi algorithms with more formalization, and finally introduce the important EM algorithm for unsupervised (or semi-supervised) learning of a Hidden Markov Model.

In the second half of the chapter, we introduce Maximum Entropy Markov Models gradually, beginning with techniques that may already be familiar to you from statistics: linear regression and logistic regression. We next introduce MaxEnt. MaxEnt by itself is not a sequence classifier; it is used to assign a class to a single element. The name Maximum Entropy comes from the idea that the classifier finds the probabilistic model which follows Occam's Razor in being the simplest (least constrained; has the maximum entropy) yet still consistent with some specific constraints. The Maximum Entropy Markov Model is the extension of MaxEnt to the sequence labeling task, adding components such as the Viterbi algorithm.

Although this chapter introduces MaxEnt, which is a classifier, we will not focus in general on non-sequential classification. Non-sequential classification will be addressed in later chapters with the introduction of classifiers like the **Gaussian Mixture Model** in (Ch. 9) and the **Naive Bayes** and **decision list** classifiers in (Ch. 20).

6.1 Markov Chains

The Hidden Markov Model is one of the most important machine learning models in speech and language processing. In order to define it properly, we need to first introduce the **Markov chain**, sometimes called the **observed Markov model**. Markov chains and Hidden Markov Models are both extensions of the finite automata of Ch. 2.

WEIGHTED

Recall that a finite automaton is defined by a set of states, and a set of transitions between states that are taken based on the input observations. A **weighted finite-state automaton** is a simple augmentation of the finite automaton in which each arc is associated with a probability, indicating how likely that path is to be taken. The probability on all the arcs leaving a node must sum to 1.

MARKOV CHAIN

A **Markov chain** is a special case of a weighted automaton in which the input sequence uniquely determines which states the automaton will go through. Because it can't represent inherently ambiguous problems, a Markov chain is only useful for assigning probabilities to unambiguous sequences.

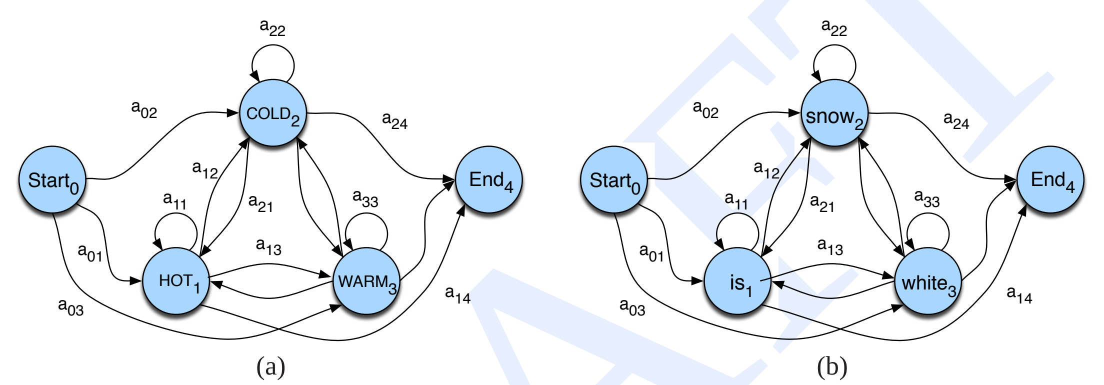


Figure 6.1 A Markov chain for weather (a) and one for words (b). A Markov chain is specified by the structure, the transition between states, and the start and end states.

Fig. 6.1a shows a Markov chain for assigning a probability to a sequence of weather events, where the vocabulary consists of HOT, COLD, and RAINY. Fig. 6.1b shows another simple example of a Markov chain for assigning a probability to a sequence of words $w_1 ... w_n$. This Markov chain should be familiar; in fact it represents a bigram language model. Given the two models in Figure 6.1 we can assign a probability to any sequence from our vocabulary. We'll go over how to do this shortly.

First, let's be more formal. We'll view a Markov chain as a kind of probabilistic **graphical model**; a way of representing probabilistic assumptions in a graph. A Markov chain is specified by the following components:

$Q = q_1 q_2 \dots q_N$	a set of N **states**
$A = a_{01} a_{02} \dots a_{n1} \dots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
q_0, q_F	a special **start state** and **end (final) state** which are not associated with observations.

Fig. 6.1 shows that we represent the states (including start and end states) as nodes in the graph, and the transitions as edges between nodes.

A Markov chain embodies an important assumption about these probabilities. In a

FIRST-ORDER

first-order Markov chain, the probability of a particular state is dependent only on the previous state:

$$\textbf{Markov Assumption:}\quad P(q_i|q_1...q_{i-1}) = P(q_i|q_{i-1}) \tag{6.1}$$

Note that because each a_{ij} expresses the probability $p(q_j|q_i)$, the laws of probability require that the values of the outgoing arcs from a given state must sum to 1:

$$\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i \tag{6.2}$$

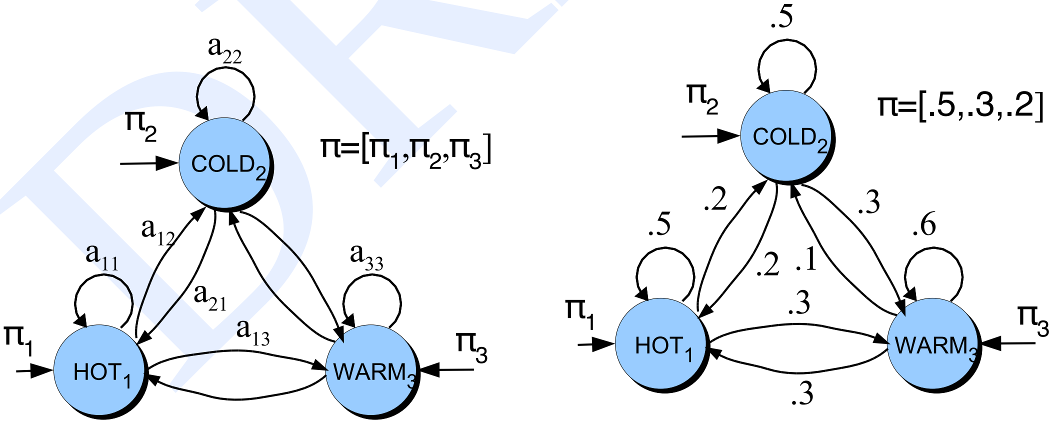
An alternate representation that is sometimes used for Markov chains doesn't rely on a start or end state, instead representing the distribution over initial states and accepting states explicitly:

$\pi = \pi_1, \pi_2, ..., \pi_N$ an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

$QA = \{q_x, q_y...\}$ a set $QA \subset Q$ of legal **accepting states**

Thus the probability of state 1 being the first state can be represented either as a_{01} or as π_1. Note that because each π_i expresses the probability $p(q_i|START)$, all the π probabilities must sum to 1:

$$\sum_{i=1}^{n} \pi_i = 1 \tag{6.3}$$



(a) (b)

Figure 6.2 Another representation of the same Markov chain for weather shown in Fig. 6.1. Instead of using a special start state with a_{01} transition probabilities, we use the π vector, which represents the distribution over starting state probabilities. The figure in (b) shows sample probabilities.

Before you go on, use the sample probabilities in Fig. 6.2b to compute the probability of each of the following sequences:

(6.4) hot hot hot hot

(6.5) cold hot cold hot

What does the difference in these probabilities tell you about a real-world weather fact encoded in Fig. 6.2b?

6.2 The Hidden Markov Model

A Markov chain is useful when we need to compute a probability for a sequence of events that we can observe in the world. In many cases, however, the events we are interested in may not be directly observable in the world. For example, in part-of-speech tagging (Ch. 5) we didn't observe part of speech tags in the world; we saw words, and had to infer the correct tags from the word sequence. We call the part-of-speech tags **hidden** because they are not observed. The same architecture will come up in speech recognition; in that case we'll see acoustic events in the world, and have to infer the presence of 'hidden' words that are the underlying causal source of the acoustics. A **Hidden Markov Model** (**HMM**) allows us to talk about both *observed* events (like words that we see in the input) and *hidden* events (like part-of-speech tags) that we think of as causal factors in our probabilistic model.

HIDDEN MARKOV MODEL

To exemplify these models, we'll use a task conceived of by Jason Eisner (2002). Imagine that you are a climatologist in the year 2799 studying the history of global warming. You cannot find any records of the weather in Baltimore, Maryland, for the summer of 2007, but you do find Jason Eisner's diary, which lists how many ice creams Jason ate every day that summer. Our goal is to use these observations to estimate the temperature every day. We'll simplify this weather task by assuming there are only two kinds of days: cold (C) and hot (H). So the Eisner task is as follows:

> Given a sequence of observations O, each observation an integer corresponding to the number of ice creams eaten on a given day, figure out the correct 'hidden' sequence Q of weather states (H or C) which caused Jason to eat the ice cream.

Let's begin with a formal definition of a Hidden Markov Model, focusing on how it differs from a Markov chain. An **HMM** is specified by the following components:

HMM

$Q = q_1 q_2 \ldots q_N$ a set of N **states**

$A = a_{11} a_{12} \ldots a_{n1} \ldots a_{nn}$ a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$

$O = o_1 o_2 \ldots o_T$ a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$.

$B = b_i(o_t)$ a sequence of **observation likelihoods:**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i.

q_0, q_F a special **start state** and **end (final) state** which are not associated with observations, together with transition probabilities $a_{01} a_{02} .. a_{0n}$ out of the start state and $a_{1F} a_{2F} ... a_{nF}$ into the end state.

As we noted for Markov chains, an alternate representation that is sometimes used for HMMs doesn't rely on a start or end state, instead representing the distribution over initial and accepting states explicitly. We won't be using the π notation in this textbook, but you may see it in the literature:

$\pi = \pi_1, \pi_2, ..., \pi_N$ an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

$QA = \{q_x, q_y ...\}$ a set $QA \subset Q$ of legal **accepting states**

A first-order Hidden Markov Model instantiates two simplifying assumptions. First, as with a first-order Markov chain, the probability of a particular state is dependent only on the previous state:

$$\textbf{Markov Assumption:} \quad P(q_i | q_1 ... q_{i-1}) = P(q_i | q_{i-1}) \tag{6.6}$$

Second, the probability of an output observation o_i is dependent only on the state that produced the observation q_i, and not on any other states or any other observations:

$$\textbf{Output Independence Assumption:} \; P(o_i | q_1 \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i | q_i) \tag{6.7}$$

Fig. 6.3 shows a sample HMM for the ice cream task. The two hidden states (H and C) correspond to hot and cold weather, while the observations (drawn from the alphabet $O = \{1, 2, 3\}$) correspond to the number of ice creams eaten by Jason on a given day.

FULLY-CONNECTED
ERGODIC HMM
LEFT-TO-RIGHT
BAKIS

Notice that in the HMM in Fig. 6.3, there is a (non-zero) probability of transitioning between any two states. Such an HMM is called a **fully-connected** or **ergodic HMM**. Sometimes, however, we have HMMs in which many of the transitions between states have zero probability. For example, in **left-to-right** (also called **Bakis**) HMMs, the state transitions proceed from left to right, as shown in Fig. 6.4. In a Bakis HMM,

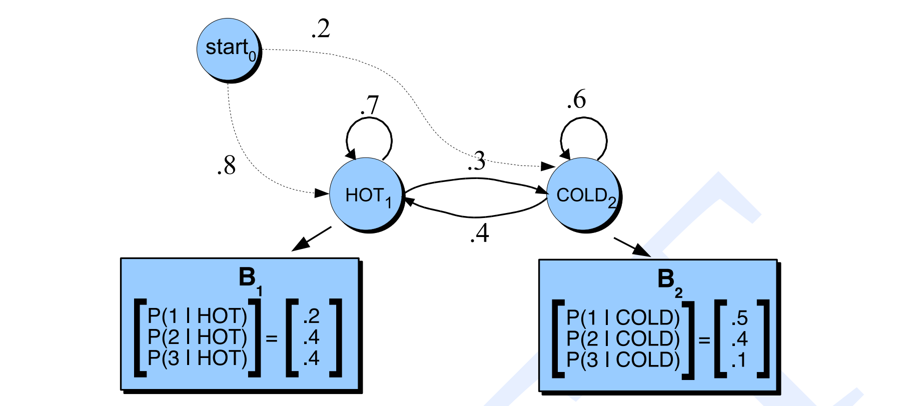


Figure 6.3 A Hidden Markov Model for relating numbers of ice creams eaten by Jason (the observations) to the weather (H or C, the hidden variables). For this example we are not using an end-state, instead allowing both states 1 and 2 to be a final (accepting) state.

there are no transitions going from a higher-numbered state to a lower-numbered state (or, more accurately, any transitions from a higher-numbered state to a lower-numbered state have zero probability). Bakis HMMs are generally used to model temporal processes like speech; we will see more of them in Ch. 9.

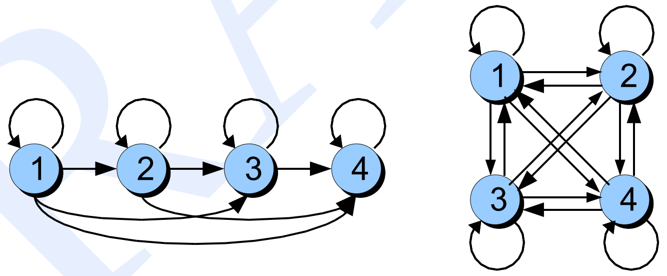


Figure 6.4 Two 4-state Hidden Markov Models; a left-to-right (Bakis) HMM on the left, and a fully-connected (ergodic) HMM on the right. In the Bakis model, all transitions not shown have zero probability.

Now that we have seen the structure of an HMM, we turn to algorithms for computing things with them. An influential tutorial by Rabiner (1989), based on tutorials by Jack Ferguson in the 1960s, introduced the idea that Hidden Markov Models should be characterized by **three fundamental problems**:

Problem 1 (Computing Likelihood): Given an HMM $\lambda = (A,B)$ and an observation sequence O, determine the likelihood $P(O|\lambda)$.

Problem 2 (Decoding): Given an observation sequence O and an HMM $\lambda = (A,B)$, discover the best hidden state sequence Q.

> **Problem 3 (Learning):** Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B.

We already saw an example of problem (2) in Ch. 5. In the next three sections we introduce all three problems more formally.

6.3 Computing Likelihood: The Forward Algorithm

Our first problem is to compute the likelihood of a particular observation sequence. For example, given the HMM in Fig. 6.2b, what is the probability of the sequence *3 1 3*? More formally:

> **Computing Likelihood:** Given an HMM $\lambda = (A,B)$ and an observation sequence O, determine the likelihood $P(O|\lambda)$.

For a Markov chain, where the surface observations are the same as the hidden events, we could compute the probability of *3 1 3* just by following the states labeled *3 1 3* and multiplying the probabilities along the arcs. For a Hidden Markov Model, things are not so simple. We want to determine the probability of an ice-cream observation sequence like *3 1 3*, but we don't know what the hidden state sequence is!

Let's start with a slightly simpler situation. Suppose we already knew the weather, and wanted to predict how much ice cream Jason would eat. This is a useful part of many HMM tasks. For a given hidden state sequence (e.g. *hot hot cold*) we can easily compute the output likelihood of *3 1 3*.

Let's see how. First, recall that for Hidden Markov Models, each hidden state produces only a single observation. Thus the sequence of hidden states and the sequence of observations have the same length. [1]

Given this one-to-one mapping, and the Markov assumptions expressed in Eq. 6.6, for a particular hidden state sequence $Q = q_0, q_1, q_2, ..., q_T$ and an observation sequence $O = o_1, o_2, ..., o_T$, the likelihood of the observation sequence is:

$$P(O|Q) = \prod_{i=1}^{T} P(o_i|q_i) \tag{6.8}$$

The computation of the forward probability for our ice-cream observation *3 1 3* from one possible hidden state sequence *hot hot cold* is as follows (Fig. 6.5 shows a graphic representation of this):

$$P(3\ 1\ 3|\text{hot hot cold}) \;=\; P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \tag{6.9}$$

But of course, we don't actually know what the hidden state (weather) sequence was. We'll need to compute the probability of ice-cream events *3 1 3* instead by summing over all possible weather sequences, weighted by their probability. First, let's

[1] There are variants of HMMs called **segmental HMMs** (in speech recognition) or **semi-HMMs** (in natural language processing) in which this one-to-one mapping between the length of the hidden state sequence and the length of the observation sequence does not hold.

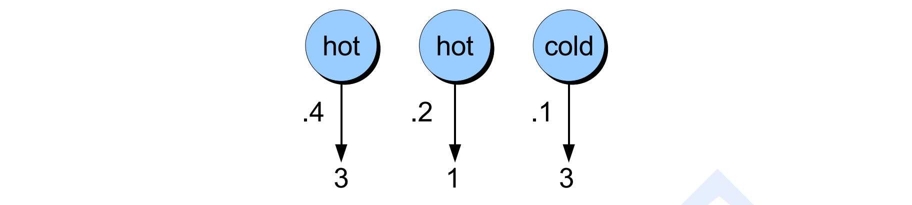


Figure 6.5 The computation of the observation likelihood for the ice-cream events *3 1 3* given the hidden state sequence *hot hot cold*.

compute the joint probability of being in a particular weather sequence Q and generating a particular sequence O of ice-cream events. In general, this is:

(6.10)
$$P(O,Q) = P(O|Q) \times P(Q) = \prod_{i=1}^{n} P(o_i|q_i) \times \prod_{i=1}^{n} P(q_i|q_{i-1})$$

The computation of the joint probability of our ice-cream observation *3 1 3* and one possible hidden state sequence *hot hot cold* is as follows (Fig. 6.6 shows a graphic representation of this):

(6.11)
$$\begin{aligned} P(3\ 1\ 3, \text{hot hot cold}) &= P(\text{hot}|\text{start}) \times P(\text{hot}|\text{hot}) \times P(\text{cold}|\text{hot}) \\ &\quad \times P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \end{aligned}$$

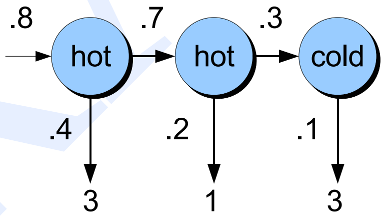


Figure 6.6 The computation of the joint probability of the ice-cream events *3 1 3* and the hidden state sequence *hot hot cold*.

Now that we know how to compute the joint probability of the observations with a particular hidden state sequence, we can compute the total probability of the observations just by summing over all possible hidden state sequences:

(6.12)
$$P(O) = \sum_{Q} P(O,Q) = \sum_{Q} P(O|Q)P(Q)$$

For our particular case, we would sum over the 8 three-event sequences *cold cold cold*, *cold cold hot*, i.e.:

(6.13)
$$P(3\ 1\ 3) = P(3\ 1\ 3, \text{cold cold cold}) + P(3\ 1\ 3, \text{cold cold hot}) + P(3\ 1\ 3, \text{hot hot cold}) + \ldots$$

For an HMM with N hidden states and an observation sequence of T observations, there are N^T possible hidden sequences. For real tasks, where N and T are both large, N^T is a very large number, and so we cannot compute the total observation likelihood by computing a separate observation likelihood for each hidden state sequence and then summing them up.

FORWARD ALGORITHM

Instead of using such an extremely exponential algorithm, we use an efficient ($O(N^2T)$) algorithm called the **forward algorithm**. The forward algorithm is a kind of **dynamic programming** algorithm, i.e., an algorithm that uses a table to store intermediate values as it builds up the probability of the observation sequence. The forward algorithm computes the observation probability by summing over the probabilities of all possible hidden state paths that could generate the observation sequence, but it does so efficiently by implicitly folding each of these paths into a single **forward trellis**.

Fig. 6.7 shows an example of the forward trellis for computing the likelihood of *3 1 3* given the hidden state sequence *hot hot cold*.

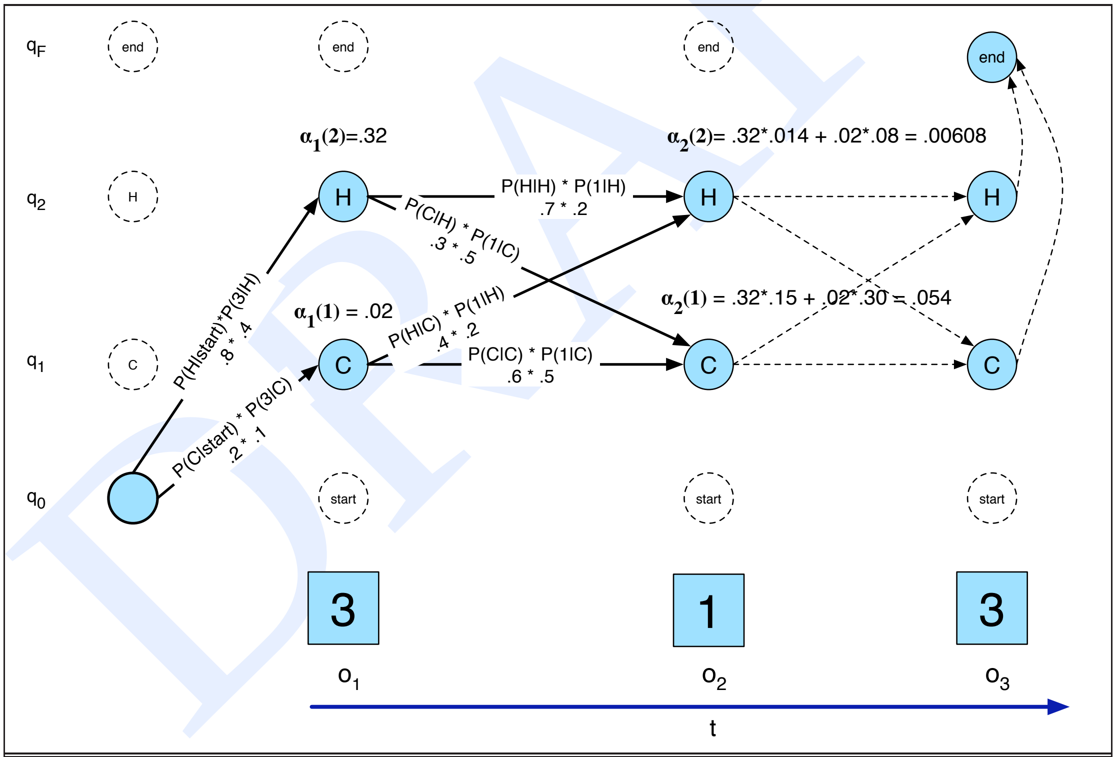


Figure 6.7 The forward trellis for computing the total observation likelihood for the ice-cream events *3 1 3*. Hidden states are in circles, observations in squares. White (unfilled) circles indicate illegal transitions. The figure shows the computation of $\alpha_t(j)$ for two states at two time steps. The computation in each cell follows Eq˙ 6.15: $\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i)a_{ij}b_j(o_t)$. The resulting probability expressed in each cell is Eq˙ 6.14: $\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda)$.

Each cell of the forward algorithm trellis $\alpha_t(j)$ represents the probability of being in state j after seeing the first t observations, given the automaton λ. The value of each cell $\alpha_t(j)$ is computed by summing over the probabilities of every path that could lead us to this cell. Formally, each cell expresses the following probability:

$$\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda) \tag{6.14}$$

Here $q_t = j$ means “the probability that the tth state in the sequence of states is state j”. We compute this probability by summing over the extensions of all the paths that lead to the current cell. For a given state q_j at time t, the value $\alpha_t(j)$ is computed as:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \tag{6.15}$$

The three factors that are multiplied in Eq˙ 6.15 in extending the previous paths to compute the forward probability at time t are:

$\alpha_{t-1}(i)$ the **previous forward path probability** from the previous time step

a_{ij} the **transition probability** from previous state q_i to current state q_j

$b_j(o_t)$ the **state observation likelihood** of the observation symbol o_t given the current state j

Consider the computation in Fig. 6.7 of $\alpha_2(1)$, the forward probability of being at time step 2 in state 1 having generated the partial observation *3 1*. This is computed by extending the α probabilities from time step 1, via two paths, each extension consisting of the three factors above: $\alpha_1(1) \times P(H|H) \times P(1|H)$ and $\alpha_1(2) \times P(H|C) \times P(1|H)$.

Fig. 6.8 shows another visualization of this induction step for computing the value in one new cell of the trellis.

We give two formal definitions of the forward algorithm; the pseudocode in Fig. 6.9 and a statement of the definitional recursion here:

1. Initialization:

$$\alpha_1(j) \;=\; a_{0j} b_j(o_1) \;\; 1 \le j \le N \tag{6.16}$$

2. Recursion (since states 0 and F are non-emitting):

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \tag{6.17}$$

3. Termination:

$$P(O|\lambda) = \alpha_T(q_F) = \sum_{i=1}^{N} \alpha_T(i)\, a_{iF} \tag{6.18}$$

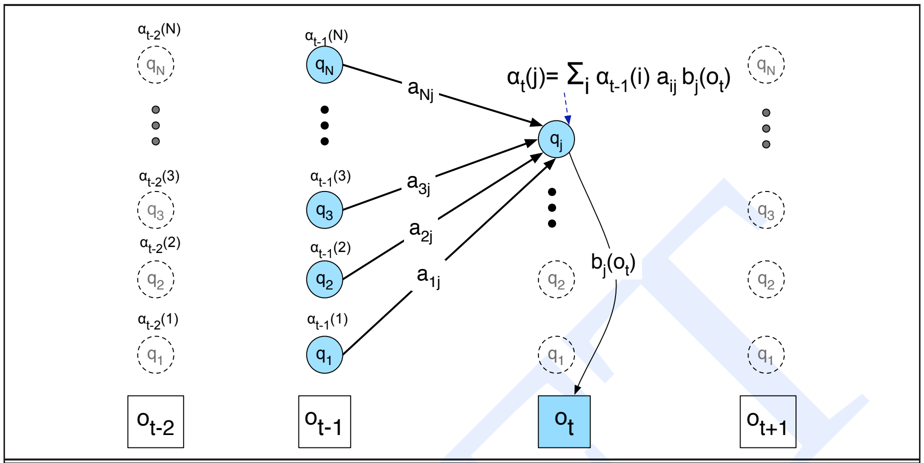


Figure 6.8 Visualizing the computation of a single element $\alpha_t(i)$ in the trellis by summing all the previous values α_{t-1} weighted by their transition probabilities a and multiplying by the observation probability $b_i(o_{t+1})$. For many applications of HMMs, many of the transition probabilities are 0, so not all previous states will contribute to the forward probability of the current state. Hidden states are in circles, observations in squares. Shaded nodes are included in the probability computation for $\alpha_t(i)$. Start and end states are not shown.

```
function FORWARD(observations of len T, state-graph of len N) returns forward-prob

  create a probability matrix forward[N+2,T]
  for each state s from 1 to N do                              ;initialization step
      $forward[s,1] \leftarrow a_{0,s} * b_s(o_1)$
  for each time step t from 2 to T do                          ;recursion step
    for each state s from 1 to N do
                $forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$
  $forward[q_F,\mathrm{T}] \leftarrow \sum_{s=1}^{N} forward[s,T] * a_{s,q_F}$          ; termination step
  return $forward[q_F,T]$
```

Figure 6.9 The forward algorithm. We've used the notation $forward[s,t]$ to represent $\alpha_t(s)$.

6.4 DECODING: THE VITERBI ALGORITHM

For any model, such as an HMM, that contains hidden variables, the task of determining which sequence of variables is the underlying source of some sequence of observations is called the **decoding** task. In the ice cream domain, given a sequence of ice cream observations *3 1 3* and an HMM, the task of the **decoder** is to find the best hidden

DECODING
DECODER

weather sequence (*H H H*). More formally,

> **Decoding**: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, ..., o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \ldots q_T$.

We might propose to find the best sequence as follows: for each possible hidden state sequence (*HHH*, *HHC*, *HCH*, etc.), we could run the forward algorithm and compute the likelihood of the observation sequence given that hidden state sequence. Then we could choose the hidden state sequence with the max observation likelihood. It should be clear from the previous section that we cannot do this because there are an exponentially large number of state sequences!

VITERBI ALGORITHM

Instead, the most common decoding algorithms for HMMs is the **Viterbi algorithm**. Like the forward algorithm, **Viterbi** is a kind of **dynamic programming**, and makes uses of a dynamic programming trellis. Viterbi also strongly resembles another dynamic programming variant, the **minimum edit distance** algorithm of Ch. 3.

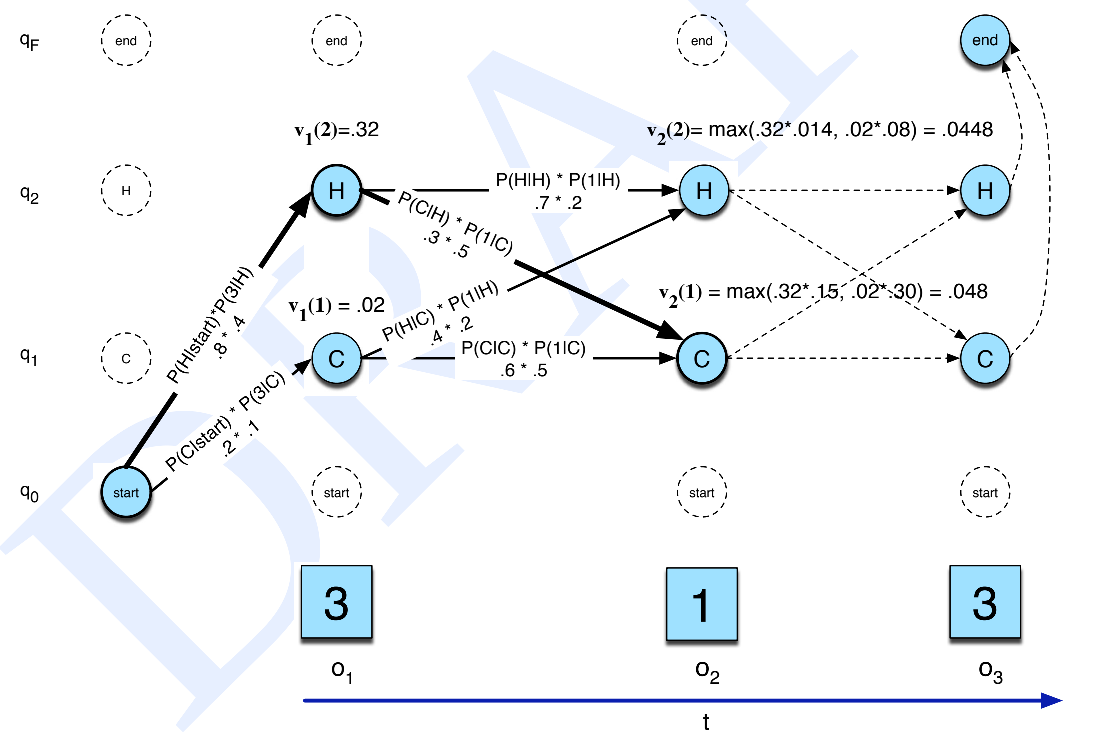


Figure 6.10 The Viterbi trellis for computing the best path through the hidden state space for the ice-cream eating events *3 1 3*. Hidden states are in circles, observations in squares. White (unfilled) circles indicate illegal transitions. The figure shows the computation of $v_t(j)$ for two states at two time steps. The computation in each cell follows Eq˙ 6.20: $v_t(j) = \max_{1 \leq i \leq N-1} v_{t-1}(i)\; a_{ij}\; b_j(o_t)$ The resulting probability expressed in each cell is Eq˙ 6.19: $v_t(j) = P(q_0, q_1, \ldots, q_{t-1}, o_1, o_2, \ldots, o_t, q_t = j|\lambda)$.

Fig. 6.10 shows an example of the Viterbi trellis for computing the best hidden state sequence for the observation sequence *3 1 3*. The idea is to process the observation sequence left to right, filling out the trellis. Each cell of the Viterbi trellis, $v_t(j)$ represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence $q_0, q_1, ..., q_{t-1}$, given the automaton λ. The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that could lead us to this cell. Formally, each cell expresses the following probability:

$$v_t(j) = \max_{q_0,q_1,...,q_{t-1}} P(q_0,q_1...q_{t-1},o_1,o_2\ldots o_t,q_t = j|\lambda) \tag{6.19}$$

Note that we represent the most probable path by taking the maximum over all possible previous state sequences $\max\limits_{q_0,q_1,...,q_{t-1}}$. Like other dynamic programming algorithms, Viterbi fills each cell recursively. Given that we had already computed the probability of being in every state at time $t-1$, We compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell. For a given state q_j at time t, the value $v_t(j)$ is computed as:

$$v_t(j) \;=\; \max_{i=1}^{N} v_{t-1}(i)\; a_{ij}\; b_j(o_t) \tag{6.20}$$

The three factors that are multiplied in Eq. 6.20 for extending the previous paths to compute the Viterbi probability at time t are:

$v_{t-1}(i)$ the **previous Viterbi path probability** from the previous time step

a_{ij} the **transition probability** from previous state q_i to current state q_j

$b_j(o_t)$ the **state observation likelihood** of the observation symbol o_t given the current state j

Fig. 6.11 shows pseudocode for the Viterbi algorithm. Note that the Viterbi algorithm is identical to the forward algorithm except that it takes the **max** over the previous path probabilities where the forward algorithm takes the **sum**. Note also that the Viterbi algorithm has one component that the forward algorithm doesn't have: **backpointers**. This is because while the forward algorithm needs to produce an observation likelihood, the Viterbi algorithm must produce a probability and also the most likely state sequence. We compute this best state sequence by keeping track of the path of hidden states that led to each state, as suggested in Fig. 6.12, and then at the end tracing back
BACKTRACE the best path to the beginning (the Viterbi **backtrace**).

Finally, we can give a formal definition of the Viterbi recursion as follows:

1. **Initialization:**

$$v_1(j) \;=\; a_{0j}b_j(o_1) \;\; 1 \le j \le N \tag{6.21}$$

$$bt_1(j) \;=\; 0 \tag{6.22}$$

```
function VITERBI(observations of len T, state-graph of len N) returns best-path

  create a path probability matrix viterbi[N+2,T]
  for each state s from 1 to N do                              ;initialization step
      $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
      $backpointer[s,1] \leftarrow 0$
  for each time step t from 2 to T do                          ;recursion step
    for each state s from 1 to N do
      $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
      $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
  $viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$              ; termination step
  $backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$      ; termination step
    return the backtrace path by following backpointers to states back in time from
  $backpointer[q_F,T]$
```

Figure 6.11 Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state-path through the HMM which assigns maximum likelihood to the observation sequence. Note that states 0 and q_F are non-emitting.

2. **Recursion** (recall states 0 and q_F are non-emitting):

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \tag{6.23}$$

$$bt_t(j) = \operatorname{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \tag{6.24}$$

3. **Termination:**

$$\text{The best score:} \quad P* = v_t(q_F) = \max_{i=1}^{N} v_T(i) * a_{i,F} \tag{6.25}$$

$$\text{The start of backtrace:} \quad q_T* = bt_T(q_F) = \operatorname{argmax}_{i=1}^{N} v_T(i) * a_{i,F} \tag{6.26}$$

6.5 TRAINING HMMS: THE FORWARD-BACKWARD ALGORITHM

We turn to the third problem for HMMs: learning the parameters of an HMM, i.e., the A and B matrices. Formally,

> **Learning:** Given an observation sequence O and the set of possible states in the HMM, learn the HMM parameters A and B.

The input to such a learning algorithm would be an unlabeled sequence of observations O and a vocabulary of potential hidden states Q. Thus for the ice cream task,

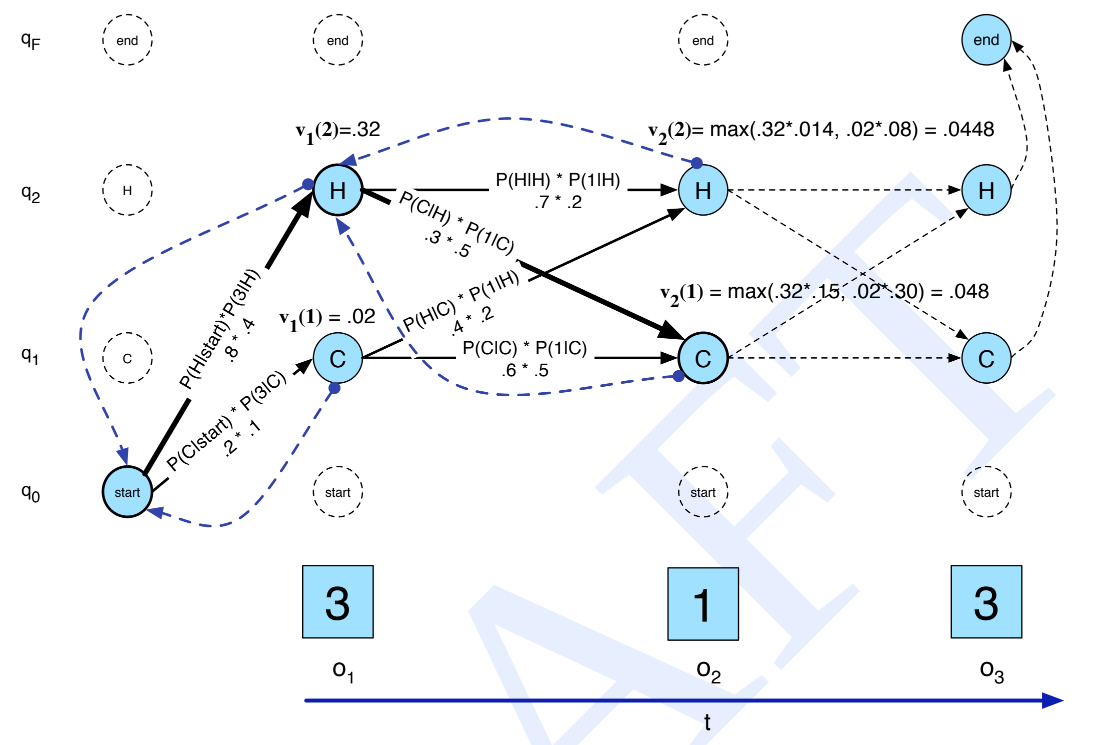


Figure 6.12 The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken blue lines) to the best path that led us to this state.

we would start with a sequence of observations $O = \{1,3,2,...,\}$, and the set of hidden states H and C. For the part-of-speech tagging task we would start with a sequence of observations $O = \{w_1, w_2, w_3 \ldots\}$ and a set of hidden states *NN*, *NNS*, *VBD*, *IN*,... and so on.

FORWARD-BACKWARD

BAUM-WELCH

EM

The standard algorithm for HMM training is the **forward-backward** or **Baum-Welch** algorithm (Baum, 1972), a special case of the **Expectation-Maximization** or **EM** algorithm (Dempster et al., 1977). The algorithm will let us train both the transition probabilities A and the emission probabilities B of the HMM.

Let us begin by considering the much simpler case of training a Markov chain rather than a Hidden Markov Model. Since the states in a Markov chain are observed, we can run the model on the observation sequence and directly see which path we took through the model, and which state generated each observation symbol. A Markov chain of course has no emission probabilities B (alternatively we could view a Markov chain as a degenerate Hidden Markov Model where all the b probabilities are 1.0 for the observed symbol and 0 for all other symbols.). Thus the only probabilities we need to train are the transition probability matrix A.

We get the maximum likelihood estimate of the probability a_{ij} of a particular transition between states i and j by counting the number of times the transition was taken, which we could call $C(i \rightarrow j)$, and then normalizing by the total count of all times we took any transition from state i:

$$a_{ij} = \frac{C(i \rightarrow j)}{\sum_{q \in Q} C(i \rightarrow q)} \tag{6.27}$$

We can directly compute this probability in a Markov chain because we know which states we were in. For an HMM we cannot compute these counts directly from an observation sequence since we don't know which path of states was taken through the machine for a given input. The Baum-Welch algorithm uses two neat intuitions to solve this problem. The first idea is to *iteratively* estimate the counts. We will start with an estimate for the transition and observation probabilities, and then use these estimated probabilities to derive better and better probabilities. The second idea is that we get our estimated probabilities by computing the forward probability for an observation and then dividing that probability mass among all the different paths that contributed to this forward probability.

BACKWARD PROBABILITY

In order to understand the algorithm, we need to define a useful probability related to the forward probability, called the **backward probability**.

The backward probability β is the probability of seeing the observations from time $t+1$ to the end, given that we are in state i at time t (and of course given the automaton λ):

$$\beta_t(i) = P(o_{t+1}, o_{t+2} \ldots o_T | q_t = i, \lambda) \tag{6.28}$$

It is computed inductively in a similar manner to the forward algorithm.

1. **Initialization:**

$$\beta_T(i) \;=\; a_{i,F}, \quad 1 \leq i \leq N \tag{6.29}$$

2. **Recursion** (again since states 0 and q_F are non-emitting):

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j), \quad 1 \leq i \leq N, 1 \leq t < T \tag{6.30}$$

3. **Termination:**

$$P(O|\lambda) = \alpha_T(q_F) = \beta_1(0) = \sum_{j=1}^{N} a_{0j}\, b_j(o_1)\, \beta_1(j) \tag{6.31}$$

Fig. 6.13 illustrates the backward induction step.

We are now ready to understand how the forward and backward probabilities can help us compute the transition probability a_{ij} and observation probability $b_i(o_t)$ from an observation sequence, even though the actual path taken through the machine is hidden.

Let's begin by showing how to estimate $\hat{a}_{ij}$ by a variant of (6.27):

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \tag{6.32}$$

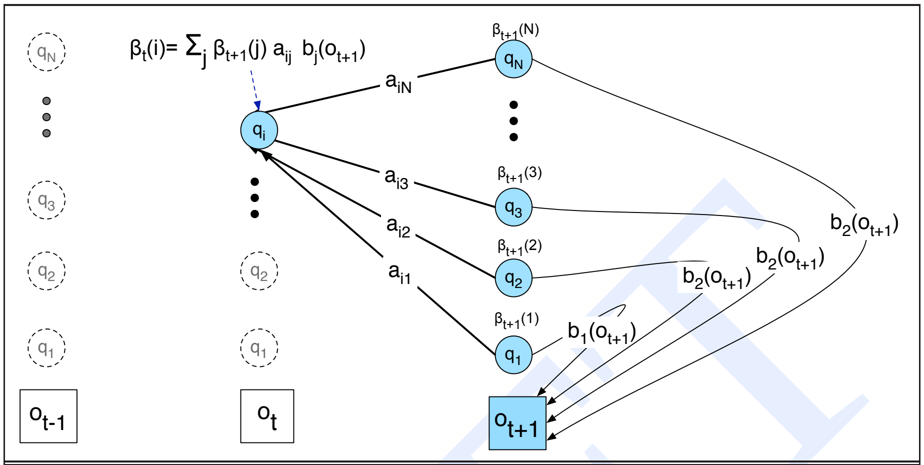


Figure 6.13 The computation of $\beta_t(i)$ by summing all the successive values $\beta_{t+1}(j)$ weighted by their transition probabilities a_{ij} and their observation probabilities $b_j(o_{t+1})$. Start and end states not shown.

How do we compute the numerator? Here's the intuition. Assume we had some estimate of the probability that a given transition $i \to j$ was taken at a particular point in time t in the observation sequence. If we knew this probability for each particular time t, we could sum over all times t to estimate the total count for the transition $i \to j$.

More formally, let's define the probability ξ_t as the probability of being in state i at time t and state j at time $t+1$, given the observation sequence and of course the model:

(6.33) $$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda)$$

In order to compute ξ_t, we first compute a probability which is similar to ξ_t, but differs in including the probability of the observation; note the different conditioning of O from Equation (6.33):

(6.34) $$\text{not-quite-}\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O | \lambda)$$

Fig. 6.14 shows the various probabilities that go into computing not-quite-ξ_t: the transition probability for the arc in question, the α probability before the arc, the β probability after the arc, and the observation probability for the symbol just after the arc. These four are multiplied together to produce *not-quite-*ξ_t as follows:

(6.35) $$\text{not-quite-}\xi_t(i,j) = \alpha_t(i)\, a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$

In order to compute ξ_t from *not-quite-*ξ_t, the laws of probability instruct us to divide by $P(O|\lambda)$, since:

(6.36) $$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)}$$

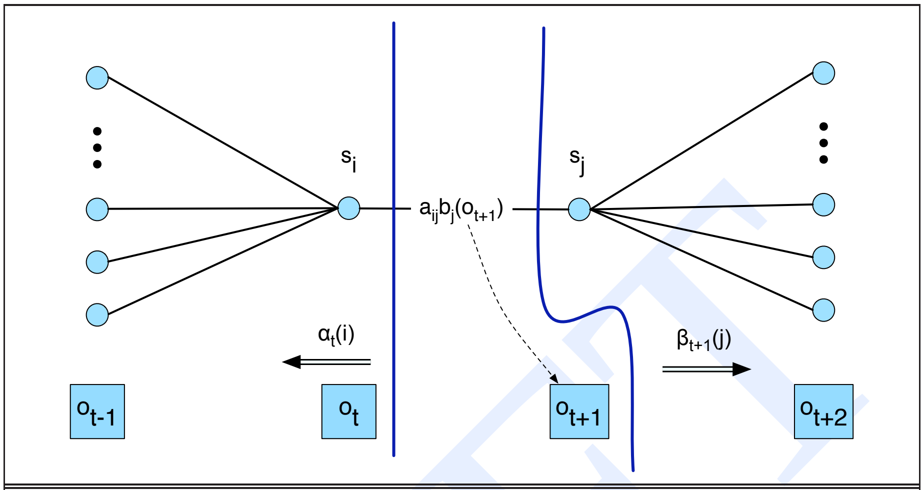


Figure 6.14 Computation of the joint probability of being in state i at time t and state j at time $t+1$. The figure shows the various probabilities that need to be combined to produce $P(q_t = i, q_{t+1} = j, O|\lambda)$: the α and β probabilities, the transition probability a_{ij} and the observation probability $b_j(o_{t+1})$. After Rabiner (1989).

The probability of the observation given the model is simply the forward probability of the whole utterance, (or alternatively the backward probability of the whole utterance!), which can thus be computed in a number of ways:

$$P(O|\lambda) = \alpha_T(N) = \beta_T(1) = \sum_{j=1}^{N} \alpha_t(j)\beta_t(j) \tag{6.37}$$

So, the final equation for ξ_t is:

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(N)} \tag{6.38}$$

The expected number of transitions from state i to state j is then the sum over all t of ξ. For our estimate of a_{ij} in (6.32), we just need one more thing: the total expected number of transitions from state i. We can get this by summing over all transitions out of state i. Here's the final formula for $\hat{a}_{ij}$:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)} \tag{6.39}$$

We also need a formula for recomputing the observation probability. This is the probability of a given symbol v_k from the observation vocabulary V, given a state j: $\hat{b}_j(v_k)$. We will do this by trying to compute:

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j} \tag{6.40}$$

For this we will need to know the probability of being in state j at time t, which we will call $\gamma_t(j)$:

$$\gamma_t(j) = P(q_t = j|O,\lambda) \tag{6.41}$$

Once again, we will compute this by including the observation sequence in the probability:

$$\gamma_t(j) = \frac{P(q_t = j,O|\lambda)}{P(O|\lambda)} \tag{6.42}$$

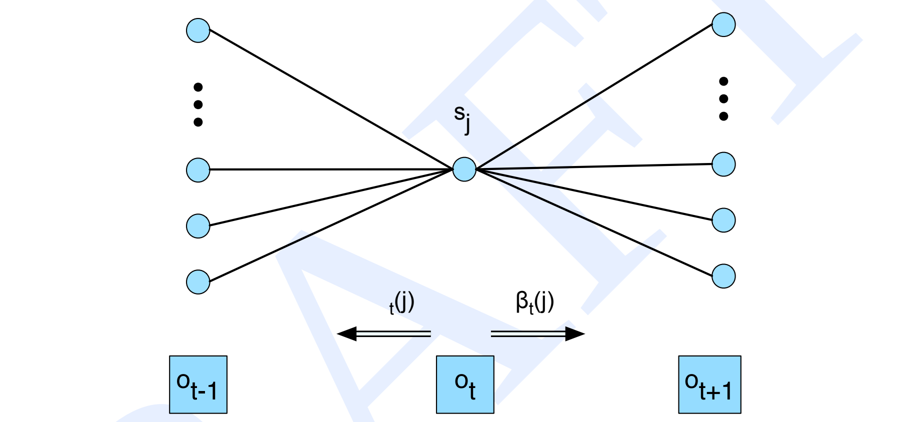


Figure 6.15 The computation of $\gamma_t(j)$, the probability of being in state j at time t. Note that γ is really a degenerate case of ξ and hence this figure is like a version of Fig. 6.14 with state i collapsed with state j. After Rabiner (1989).

As Fig. 6.15 shows, the numerator of (6.42) is just the product of the forward probability and the backward probability:

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \tag{6.43}$$

We are ready to compute b. For the numerator, we sum $\gamma_t(j)$ for all time steps t in which the observation o_t is the symbol v_k that we are interested in. For the denominator, we sum $\gamma_t(j)$ over all time steps t. The result will be the percentage of the times that we were in state j and we saw symbol v_k (the notation $\sum_{t=1 s.t. O_t=v_k}^{T}$ means "sum over all t for which the observation at time t was v_k"):

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{6.44}$$

We now have ways in (6.39) and (6.44) to *re-estimate* the transition A and observation B probabilities from an observation sequence O assuming that we already have a previous estimate of A and B.

These re-estimations form the core of the iterative forward-backward algorithm.

The forward-backward algorithm starts with some initial estimate of the HMM parameters $\lambda = (A,B)$. We then iteratively run two steps. Like other cases of the EM (expectation-maximization) algorithm, the forward-backward algorithm has two steps: the **expectation** step, or **E-step**, and the **maximization** step, or **M-step**.

EXPECTATION
E-STEP
MAXIMIZATION
M-STEP

In the E-step, we compute the expected state occupancy count γ and the expected state transition count ξ, from the earlier A and B probabilities. In the M-step, we use γ and ξ to recompute new A and B probabilities.

```
function FORWARD-BACKWARD( observations of len T, output vocabulary V, hidden
state set Q) returns HMM=(A,B)

  initialize A and B
  iterate until convergence
    E-step
        $\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \;\forall\, t \text{ and } j$
        $\xi_t(i,j) = \frac{\alpha_t(i)\,a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(N)} \;\forall\, t,\ i, \text{ and } j$
    M-step
        $\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{j=1}^{N}\xi_t(i,j)}$
        $\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)}$
  return A, B
```

Figure 6.16 The forward-backward algorithm.

Although in principle the forward-backward algorithm can do completely unsupervised learning of the A and B parameters, in practice the initial conditions are very important. For this reason the algorithm is often given extra information. For example, for speech recognition, in practice the HMM structure is very often set by hand, and only the emission (B) and (non-zero) A transition probabilities are trained from a set of observation sequences O. Sec. **??** in Ch. 9 will also discuss how initial A and B estimates are derived in speech recognition. We will also see that for speech that the forward-backward algorithm can be extended to inputs which are non-discrete ("continuous observation densities").

6.6 Maximum Entropy Models: Background

We turn now to a second probabilistic machine learning framework called **Maximum Entropy** modeling, **MaxEnt** for short. MaxEnt is more widely known as **multinomial logistic regression**.

Our goal in this chapter is to introduce the use of MaxEnt for sequence classification. Recall that the task of sequence classification or sequence labelling is to assign a label to each element in some sequence, such as assigning a part-of-speech tag to a word. The most common MaxEnt sequence classifier is the **Maximum Entropy Markov Model** or **MEMM**, to be introduced in Sec. 6.8. But before we see this use of MaxEnt as a sequence classifier, we need to introduce non-sequential classification.

The task of classification is to take a single observation, extract some useful features describing the observation, and then based on these features, to **classify** the observation into one of a set of discrete classes. A **probabilistic** classifier does slightly more than this; in addition to assigning a label or class, it gives the **probability** of the observation being in that class; indeed, for a given observation a probabilistic classifier gives a probability distribution over all classes.

Such non-sequential classification tasks occur throughout speech and language processing. For example, in **text classification** we might need to decide whether a particular email should be classified as spam or not. In **sentiment analysis** we have to determine whether a particular sentence or document expresses a positive or negative **opinion**. In many tasks, we'll need to know where the sentence boundaries are, and so we'll need to classify a period character ('.') as either a sentence boundary or not. We'll see more examples of the need for classification throughout this book.

EXPONENTIAL
LOG-LINEAR

MaxEnt belongs to the family of classifiers known as the **exponential** or **log-linear** classifiers. MaxEnt works by extracting some set of features from the input, combining them **linearly** (meaning that we multiply each by a weight and then add them up), and then, for reasons we will see below, using this sum as an exponent.

Let's flesh out this intuition just a bit more. Assume that we have some input x (perhaps it is a word that needs to be tagged, or a document that needs to be classified) from which we extract some features. A feature for tagging might be *this word ends in -ing* or *the previous word was 'the'*. For each such feature f_i, we have some weight w_i.

Given the features and weights, our goal is to choose a class (for example a part-of-speech tag) for the word. MaxEnt does this by choosing the most probable tag; the probability of a particular class c given the observation x is:

$$p(c|x) \;=\; \frac{1}{Z}\exp(\sum_i w_i f_i) \tag{6.45}$$

Here Z is a normalizing factor, used to make the probabilities correctly sum to 1; and as usual $\exp(x) = e^x$. As we'll see later, this is a simplified equation in various ways; for example in the actual MaxEnt model the features f and weights w are both dependent on the class c (i.e., we'll have different features and weights for different classes).

In order to explain the details of the MaxEnt classifier, including the definition of the normalizing term Z and the intuition of the exponential function, we'll need to understand first **linear regression**, which lays the groundwork for prediction using features, and **logistic regression**, which is our introduction to exponential models. We cover these areas in the next two sections. Readers who have had a grounding in these kinds of regression may want to skip the next two sections. Then in Sec. 6.7 we introduce the details of the MaxEnt classifier. Finally in Sec. 6.8 we show how the MaxEnt classifier is used for sequence classification in the **Maximum Entropy Markov Model** or **MEMM**.

6.6.1 Linear Regression

In statistics we use two different names for tasks that map some input features into some output value: we use the word **regression** when the output is real-valued, and **classification** when the output is one of a discrete set of classes.

You may already be familiar with linear regression from a statistics class. The idea is that we are given a set of observations, each observation associated with some features, and we want to predict some real-valued outcome for each observation. Let's see an example from the domain of predicting housing prices. Levitt and Dubner (2005) showed that the words used in a real estate ad can be used as a good predictor of whether a house will sell for more or less than its asking price. They showed, for example, that houses whose real estate ads had words like *fantastic*, *cute*, or *charming*, tended to sell for lower prices, while houses whose ads had words like *maple* and *granite* tended to sell for higher prices. Their hypothesis was that real estate agents used vague positive words like *fantastic* to mask the lack of any specific positive qualities in the house. Just for pedagogical purposes, we created the fake data in Fig. 6.17.

Number of vague adjectives	**Amount house sold over asking price**
4	0
3	\$1000
2	\$1500
2	\$6000
1	\$14000
0	\$18000

Figure 6.17 Some made-up data on the number of vague adjectives (*fantastic*, *cute*, *charming*) in a real estate ad, and the amount the house sold for over the asking price.

REGRESSION LINE

Fig. 6.18 shows a graph of these points, with the feature (# of adjectives) on the x-axis, and the price on the y-axis. We have also plotted a **regression line**, which is the line that best fits the observed data. The equation of any line is $y = mx + b$; as we show on the graph, the slope of this line is $m = -4900$, while the intercept is 16550. We can think of these two parameters of this line (slope m and intercept b) as a set of weights that we use to map from our features (in this case x, numbers of adjectives) to our output value y (in this case price). We can represent this linear function using w to refer to weights as follows:

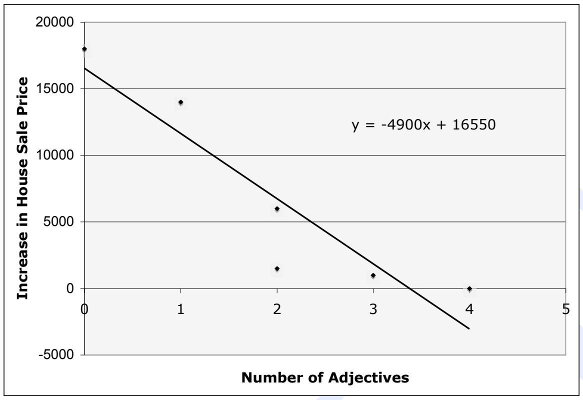


Figure 6.18 A plot of the (made-up) points in Fig. 6.17 and the regression line that best fits them, with the equation $y = -4900x + 16550$.

$$\text{price} = w_0 + w_1 * \text{Num\_Adjectives} \tag{6.46}$$

Thus Eq. 6.46 gives us a linear function that lets us estimate the sales price for any number of these adjectives. For example, how much would we expect a house whose ad has 5 adjectives to sell for?

The true power of linear models comes when we use more than one feature (technically we call this **multiple linear regression**). For example, the final house price probably depends on many factors such as the average mortgage rate that month, the number of unsold houses on the market, and many other such factors. We could encode each of these as a variable, and the importance of each factor would be the weight on that variable, as follows:

$$\text{price} = w_0 + w_1 * \text{Num\_Adjectives} + w_2 * \text{Mortgage Rate} + w_3 * \text{Num\_Unsold\_Houses} \tag{6.47}$$

FEATURE

In speech and language processing, we often call each of these predictive factors like the number of adjectives or the mortgage rate a **feature**. We represent each observation (each house for sale) by a vector of these features. Suppose a house has 1 adjective in its ad, and the mortgage rate was 6.5 and there were 10,000 unsold houses in the city. The feature vector for the house would be $\vec{f} = (1, 6.5, 10000)$. Suppose the weight vector that we had previously learned for this task was $\vec{w} = (w_0, w_1, w_2, w_3) = (18000, -5000, -3000, -1.8)$. Then the predicted value for this house would be computed by multiplying each feature by its weight:

$$\text{price} = w_0 + \sum_{i=1}^{N} w_i \times f_i \tag{6.48}$$

In general we will pretend that there is an extra feature f_0 which has the value 1, an **intercept feature**, which make the equations simpler with regard to that pesky w_0, and so in general we can represent a linear regression for estimating the value of y as:

$$\textbf{linear regression:} \qquad y = \sum_{i=0}^{N} w_i \times f_i \tag{6.49}$$

DOT PRODUCT

Taking two vectors and creating a scalar by multiplying each element in a pairwise fashion and summing the results is called the **dot product**. Recall that the dot product $a \cdot b$ between two vectors a and b is defined as:

$$\textbf{dot product:} \qquad a \cdot b = \sum_{i=1}^{N} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{6.50}$$

Thus Eq. 6.49 is equivalent to the dot product between the weights vector and the feature vector:

$$y = w \cdot f \tag{6.51}$$

Vector dot products occur very frequently in speech and language processing; we will often rely on the dot product notation to avoid the messy summation signs.

Learning in linear regression

How do we learn the weights for linear regression? Intuitively we'd like to choose weights that make the estimated values y as close as possible to the actual values that we saw in the training set.

Consider a particular instance $x^{(j)}$ from the training set (we'll use superscripts in parentheses to represent training instances), which has an observed label in the training set $y^{(j)}_{obs}$. Our linear regression model predicts a value for $y^{(j)}$ as follows:

$$y^{(j)}_{pred} = \sum_{i=0}^{N} w_i \times f^{(j)}_i \tag{6.52}$$

We'd like to choose the whole set of weights W so as to minimize the difference between the predicted value $y^{(j)}_{pred}$ and the observed value $y^{(j)}_{obs}$, and we want this difference minimized over all the M examples in our training set. Actually we want to minimize the absolute value of the difference (since we don't want a negative distance in one example to cancel out a positive difference in another example), so for simplicity (and differentiability) we minimize the square of the difference. Thus the total value we want to minimize, which we call the **sum-squared error**, is this cost function of the current set of weights W:

SUM-SQUARED ERROR

$$\text{cost}(W) = \sum_{j=0}^{M} \left(y^{(j)}_{pred} - y^{(j)}_{obs} \right)^2 \tag{6.53}$$

We won't give here the details of choosing the optimal set of weights to minimize the sum-squared error. But, briefly, it turns out that if we put the entire training set into a single matrix X with each row in the matrix consisting of the vector of features associated with each observation $x^{(i)}$, and put all the observed y values in a vector $\vec{y}$, that there is a closed-form formula for the optimal weight values W which will minimize cost(W):

$$W = (X^TX)^{-1}X^T\vec{y} \tag{6.54}$$

Implementations of this equation are widely available in statistical packages like SPSS or R.

6.6.2 Logistic regression

Linear regression is what we want when we are predicting a real-valued outcome. But somewhat more commonly in speech and language processing we are doing **classification**, in which the output y we are trying to predict takes on one from a small set of discrete values.

Consider the simplest case of binary classification, where we want to classify whether some observation x is in the class (true) or not in the class (false). In other words y can only take on the values 1 (true) or 0 (false), and we'd like a classifier that can take features of x and return true or false. Furthermore, instead of just returning the 0 or 1 value, we'd like a model that can give us the **probability** that a particular observation is in class 0 or 1. This is important because in most real-world tasks we're passing the results of this classifier onto some further classifier to accomplish some task. Since we are rarely completely certain about which class an observation falls in, we'd prefer not to make a hard decision at this stage, ruling out all other classes. Instead, we'd like to pass on to the later classifier as much information as possible: the entire set of classes, with the probability value that we assign to each class.

Could we modify our linear regression model to use it for this kind of probabilistic classification? Suppose we just tried to train a linear model to predict a probability as follows:

$$\begin{aligned} P(y = true|x) &= \sum_{i=0}^{N} w_i \times f_i && (6.55)\\ &= w \cdot f && (6.56) \end{aligned}$$

We could train such a model by assigning each training observation the target value $y = 1$ if it was in the class (true) and the target value $y = 0$ if it was not (false). Each observation x would have a feature vector f, and we would train the weight vector w to minimize the predictive error from 1 (for observations in the class) or 0 (for observations not in the class). After training, we would compute the probability of a class given an observation by just taking the dot product of the weight vector with the features for that observation.

The problem with this model is that there is nothing to force the output to be a legal probability, i.e. to lie between zero and 1. The expression $\sum_{i=0}^{N} w_i \times f_i$ produces values from $-\infty$ to ∞. How can we fix this problem? Suppose that we keep our linear predictor $w \cdot f$, but instead of having it predict a probability, we have it predict a *ratio* of two probabilities. Specifically, suppose we predict the ratio of the probability of being in the class to the probability of not being in the class. This ratio is called the **odds**. If an event has probability .75 of occurring and probability .25 of not occurring, we say

ODDS

the **odds** of occurring is $.75/.25 = 3$. We could use the linear model to predict the odds of y being true:

$$\frac{p(y = true)|x}{1 - p(y = true|x)} = w \cdot f \tag{6.57}$$

This last model is close: a ratio of probabilities can lie between 0 and ∞. But we need the left-hand side of the equation to lie between $-\infty$ and ∞. We can achieve this by taking the natural log of this probability:

$$\ln\left(\frac{p(y = true|x)}{1 - p(y = true|x)}\right) = w \cdot f \tag{6.58}$$

LOGIT FUNCTION

Now both the left and right hand lie between $-\infty$ and ∞. This function on the left (the log of the odds) is known as the **logit function**:

$$\text{logit}\,(p(x)) = \ln\left(\frac{p(x)}{1 - p(x)}\right) \tag{6.59}$$

LOGISTIC REGRESSION

The model of regression in which we use a linear function to estimate, not the probability, but the logit of the probability, is known as **logistic regression**. If the linear function is estimating the logit, what is the actual formula in logistic regression for the probability $P(y = true)$? You should stop here and take Equation (6.58) and apply some simple algebra to solve for the probability $P(y = true)$.

Hopefully when you solved for $P(y = true)$ you came up with a derivation something like the following:

$$\begin{aligned}
\ln\left(\frac{p(y = \text{true}|x)}{1 - p(y = \text{true}|x)}\right) &= w \cdot f \\
\frac{p(y = \text{true}|x)}{1 - p(y = \text{true}|x)} &= e^{w \cdot f} \qquad (6.60) \\
p(y = \text{true}|x) &= (1 - p(y = \text{true}|x))e^{w \cdot f} \\
p(y = \text{true}|x) &= e^{w \cdot f} - p(y = \text{true}|x)e^{w \cdot f} \\
p(y = \text{true}|x) + p(y = \text{true}|x)e^{w \cdot f} &= e^{w \cdot f} \\
p(y = \text{true}|x)(1 + e^{w \cdot f}) &= e^{w \cdot f}
\end{aligned}$$

$$p(y = \text{true}|x) = \frac{e^{w \cdot f}}{1 + e^{w \cdot f}} \tag{6.61}$$

Once we have this probability, we can easily state the probability of the observation not belonging to the class, $p(y = false|x)$, as the two must sum to 1:

$$p(y = false|x) = \frac{1}{1 + e^{w \cdot f}} \tag{6.62}$$

Here are the equations again using explicit summation notation:

$$p(y=\text{true}|x) = \frac{\exp(\sum_{i=0}^{N} w_i f_i)}{1+\exp(\sum_{i=0}^{N} w_i f_i)} \tag{6.63}$$

$$p(y=\text{false}|x) = \frac{1}{1+\exp(\sum_{i=0}^{N} w_i f_i)} \tag{6.64}$$

We can express the probability $P(y=\text{true}|x)$ in a slightly different way, by dividing the numerator and denominator in (6.61) by $e^{-w\cdot f}$:

$$p(y=\text{true}|x) = \frac{e^{w\cdot f}}{1+e^{w\cdot f}} \tag{6.65}$$

$$= \frac{1}{1+e^{-w\cdot f}} \tag{6.66}$$

LOGISTIC FUNCTION

These last equation is now in the form of what is called the **logistic function**, (the function that gives logistic regression its name). The general form of the logistic function is:

$$\frac{1}{1+e^{-x}} \tag{6.67}$$

The logistic function maps values from $-\infty$ and ∞ to lie between 0 and 1

Again, we can express $P(y=\text{false}|x)$ so as to make the probabilities sum to one:

$$p(y=\text{false}|x) = \frac{e^{-w\cdot f}}{1+e^{-w\cdot f}} \tag{6.68}$$

6.6.3 Logistic regression: Classification

CLASSIFICATION

INFERENCE

Given a particular observation, how do we decide which of the two classes ('true' or 'false') it belongs to? This is the task of **classification**, also called **inference**. Clearly the correct class is the one with the higher probability. Thus we can safely say that our observation should be labeled 'true' if:

$$p(y=true|x) > p(y=false|x)$$
$$\frac{p(y=true|x)}{p(y=false|x)} > 1$$
$$\frac{p(y=true|x)}{1-p(y=true|x)} > 1$$

and substituting from Eq. 6.60 for the odds ratio:

$$e^{w\cdot f} > 1$$
$$w\cdot f > 0 \tag{6.69}$$

or with the explicit sum notation:

$$\sum_{i=0}^{N} w_i f_i > 0 \tag{6.70}$$

Thus in order to decide if an observation is a member of the class we just need to compute the linear function, and see if its value is positive; if so, the observation is in the class.

A more advanced point: the equation $\sum_{i=0}^{N} w_i f_i = 0$ is the equation of a **hyperplane** (a generalization of a line to N dimensions). The equation $\sum_{i=0}^{N} w_i f_i > 0$ is thus the part of N-dimensional space above this hyperplane. Thus we can see the logistic regression function as learning a hyperplane which separates points in space which are in the class ('true') from points which are not in the class.

6.6.4 Advanced: Learning in logistic regression

CONDITIONAL MAXIMUM LIKELIHOOD ESTIMATION

In linear regression, learning consisted of choosing the weights w which minimized the sum-squared error on the training set. In logistic regression, by contrast, we generally use **conditional maximum likelihood estimation**. What this means is that we choose the parameters w which makes the probability of the observed y values in the training data to be the highest, given the observations x. In other words, for an individual training observation x, we want to choose the weights as follows:

$$\hat{w} = \underset{w}{\operatorname{argmax}}\, P(y^{(i)}|x^{(i)}) \tag{6.71}$$

And we'd like to choose the optimal weights for the entire training set:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \prod_i P(y^{(i)}|x^{(i)}) \tag{6.72}$$

We generally work with the log likelihood:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_i \log P(y^{(i)}|x^{(i)}) \tag{6.73}$$

So, more explicitly:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_i \log \begin{cases} P(y^{(i)} = 1|x^{(i)})) & \text{for } y^{(i)} = 1 \\ P(y^{(i)} = 0|x^{(i)})) & \text{for } y^{(i)} = 0 \end{cases} \tag{6.74}$$

This equation is unwieldy, and so we usually apply a convenient representational trick. Note that if $y = 0$ the first term goes away, while if $y = 1$ the second term goes away:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_i y^{(i)} \log P(y^{(i)} = 1|x^{(i)})) + (1 - y^{(i)}) \log P(y^{(i)} = 0|x^{(i)}) \tag{6.75}$$

Now if we substitute in (6.66) and (6.68), we get:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_i y^{(i)} \log \frac{1}{1+e^{-w \cdot f}} + (1-y^{(i)}) \log \frac{e^{-w \cdot f}}{1+e^{-w \cdot f}} \tag{6.76}$$

CONVEX OPTIMIZATION

Finding the weights which result in the maximum log-likelihood according to (6.76) is a problem in the field known as **convex optimization**. Among the most commonly used algorithms are **quasi-Newton** methods like L-BFGS, as well as gradient ascent, conjugate gradient, and various iterative scaling algorithms (Darroch and Ratcliff, 1972; Della Pietra et al., 1997; Malouf, 2002). These learning algorithms are available in the various MaxEnt modeling toolkits but are too complex to define here; interested readers should see the machine learning textbooks suggested at the end of the chapter.

6.7 Maximum Entropy Modeling

MULTINOMIAL LOGISTIC REGRESSION
MAXENT

We showed above how logistic regression can be used to classify an observation into one of two classes. But most of the time the kinds of classification problems that come up in language processing involve larger numbers of classes (such as the set of part-of-speech classes). Logistic regression can also be defined for such functions with many discrete values. In such cases it is called **multinomial logistic regression**. As we mentioned above, multinomial logistic regression is called **MaxEnt** in speech and language processing (see Sec. 6.7.1 on the intuition behind the name 'maximum entropy').

The equations for computing the class probabilities for a MaxEnt classifier are a generalization of Eqs. 6.63-6.64 above. Let's assume that the target value y is a random variable which can take on C different values corresponding to the classes c_1, c_2,...,c_C.

We said earlier in this chapter that in a MaxEnt model we estimate the probability that y is a particular class c as:

$$p(c|x) \quad = \quad \frac{1}{Z} \exp \sum_i w_i f_i \tag{6.77}$$

Let's now add some details to this schematic equation. First we'll flesh out the normalization factor Z, specify the number of features as N, and make the value of the weight dependent on the class c. The final equation is:

$$p(c|x) \quad = \quad \frac{\exp\left(\sum_{i=0}^{N} w_{ci} f_i\right)}{\sum_{c' \in C} \exp\left(\sum_{i=0}^{N} w_{c'i} f_i\right)} \tag{6.78}$$

Note that the normalization factor Z is just used to make the exponential into a true probability;

$$Z = \sum_{C} p(c|x) = \sum_{c' \in C} \exp\left(\sum_{i=0}^{N} w_{c'i} f_i\right) \tag{6.79}$$

We need to make one more change to see the final MaxEnt equation. So far we've been assuming that the features f_i are real-valued. It is more common in speech and language processing, however, to use binary-valued features. A feature that only takes on the values 0 and 1 is also called an **indicator function**. In general, the features we use are indicator functions of some property of the observation and the class we are considering assigning. Thus in MaxEnt, instead of the notation f_i, we will often use the notation $f_i(c,x)$, meaning a feature i for a particular class c for a given observation x.

INDICATOR FUNCTION

The final equation for computing the probability of y being of class c given x in MaxEnt is:

$$p(c|x) \;=\; \frac{\exp\left(\sum_{i=0}^{N} w_{ci} f_i(c,x)\right)}{\sum_{c' \in C} \exp\left(\sum_{i=0}^{N} w_{c'i} f_i(c',x)\right)} \tag{6.80}$$

To get a clearer intuition of this use of binary features, let's look at some sample features for the task of part-of-speech tagging. Suppose we are assigning a part-of-speech tag to the word *race* in (6.81), repeated from (**??**):

(6.81) Secretariat/NNP is/BEZ expected/VBN to/TO **race**/<u>??</u> tomorrow/

Again, for now we're just doing classification, not sequence classification, so let's consider just this single word. We'll discuss in Sec. 6.8 how to perform tagging for a whole sequence of words.

We would like to know whether to assign the class *VB* to *race* (or instead assign some other class like *NN*). One useful feature, we'll call it f_1, would be the fact that the current word is *race*. We can thus add a binary feature which is true if this is the case:

$$f_1(c,x) \;=\; \begin{cases} 1 & \text{if } \; word_i = \text{“race”} \;\;\&\;\; c = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

Another feature would be whether the previous word has the tag *TO*:

$$f_2(c,x) \;=\; \begin{cases} 1 & \text{if } \; t_{i-1} = \text{TO} \;\;\&\;\; c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

Two more part-of-speech tagging features might focus on aspects of a word's spelling and case:

$$f_3(c,x) \;=\; \begin{cases} 1 & \text{if } \; \text{suffix}(word_i) = \text{“ing”} \;\;\&\;\; c = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$

$$f_4(c,x) \;=\; \begin{cases} 1 & \text{if } \text{is\_lower\_case}(word_i) \;\; \& \;\; c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

Since each feature is dependent on both a property of the observation and the class being labeled, we would need to have separate feature for, e.g, the link between *race* and VB, or the link between a previous TO and NN:

$$f_5(c,x) \;=\; \begin{cases} 1 & \text{if } word_i = \text{"race"} \;\; \& \;\; c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

$$f_6(c,x) \;=\; \begin{cases} 1 & \text{if } t_{i-1} = \text{TO} \;\; \& \;\; c = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

Each of these features has a corresponding weight. Thus the weight $w_1(c,x)$ would indicate how strong a cue the word *race* is for the tag VB, the weight $w_2(c,x)$ would indicate how strong a cue the previous tag *TO* is for the current word being a VB, and so on.

		f1	f2	f3	f4	f5	f6
VB	f	0	1	0	1	1	0
VB	w		.8		.01	.1	
NN	f	1	0	0	0	0	1
NN	w	.8					-1.3

Figure 6.19 Some sample feature values and weights for tagging the word *race* in (6.81).

Let's assume that the feature weights for the two classes VB and VN are as shown in Fig. 6.19. Let's call the current input observation (where the current word is *race*) x. We can now compute $P(NN|x)$ and $P(VB|x)$, using Eq. 6.80:

$$P(NN|x) \;=\; \frac{e^{.8}e^{-1.3}}{e^{.8}e^{-1.3}+e^{.8}e^{.01}e^{.1}} = .20 \tag{6.82}$$

$$P(VB|x) \;=\; \frac{e^{.8}e^{.01}e^{.1}}{e^{.8}e^{-1.3}+e^{.8}e^{.01}e^{.1}} = .80 \tag{6.83}$$

Notice that when we use MaxEnt to perform **classification**, MaxEnt naturally gives us a probability distribution over the classes. If we want to do a hard-classification and choose the single-best class, we can choose the class that has the highest probability, i.e.:

$$\hat{c} = \operatorname*{argmax}_{c \in C} P(c|x) \tag{6.84}$$

Classification in MaxEnt is thus a generalization of classification in (boolean) logistic regression. In boolean logistic regression, classification involves building one linear expression which separates the observations in the class from the observations not in the class. Classification in MaxEnt, by contrast, involves building a separate linear expression for each of C classes.

But as we'll see later in Sec. 6.8, we generally don't use MaxEnt for hard classification. Usually we want to use MaxEnt as part of sequence classification, where we want not the best single class for one unit, but the best total sequence. For this task, it's useful to exploit the entire probability distribution for each individual unit, to help find the best sequence. Indeed even in many non-sequence applications a probability distribution over the classes is more useful than a hard choice.

The features we have described so far express a single binary property of an observation. But it is often useful to create more complex features that express combinations of properties of a word. Some kinds of machine learning models, like Support Vector Machines (SVMs), can automatically model the interactions between primitive properties, but in MaxEnt any kind of complex feature has to be defined by hand. For example a word starting with a capital letter (like the word *Day*) is more likely to be a proper noun (NNP) than a common noun (for example in the expression *United Nations Day*). But a word which is capitalized but which occurs at the beginning of the sentence (the previous word is `<s>`), as in *Day after day....*, is not more likely to be a proper noun. Even if each of these properties were already a primitive feature, MaxEnt would not model their combination, so this boolean combination of properties would need to be encoded as a feature by hand:

$$f_{125}(c,x) \;=\; \begin{cases} 1 & \text{if } word_{i-1} = \texttt{<s>} \;\;\&\;\; \text{isupperfirst}(word_i) \;\;\&\;\; c = \text{NNP} \\ 0 & \text{otherwise} \end{cases}$$

A key to successful use of MaxEnt is thus the design of appropriate features and feature combinations.

Learning Maximum Entropy Models

Learning a MaxEnt model can be done via a generalization of the logistic regression learning algorithms described in Sec. 6.6.4; as we saw in (6.73), we want to find the parameters w which maximize the log likelihood of the M training samples:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_i \log P(y^{(i)}|x^{(i)}) \tag{6.85}$$

As with binary logistic regression, we use some convex optimization algorithm to find the weights which maximize this function.

REGULARIZATION

A brief note: one important aspect of MaxEnt training is a kind of smoothing of the weights called **regularization**. The goal of regularization is to penalize large weights; it turns out that otherwise a MaxEnt model will learn very high weights which overfit the training data. Regularization is implemented in training by changing the likelihood function that is optimized. Instead of the optimization in (6.85), we optimize the following:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_i \log P(y^{(i)}|x^{(i)}) - \alpha R(w) \tag{6.86}$$

where $R(w)$ is a **regularization** term used to penalize large weights. It is common to make the regularization term $R(w)$ be a quadratic function of the weight values:

$$R(W) = \sum_{j=1}^{N} w_j^2 \tag{6.87}$$

Subtracting squares of the weights will thus result in preferring smaller weights:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_i \log P(y^{(i)}|x^{(i)}) - \alpha \sum_{j=1}^{N} w_j^2 \tag{6.88}$$

It turns that this kind of regularization corresponds to assuming that weights are distributed according to a Gaussian distribution with mean $\mu = 0$. In a Gaussian or normal distribution, the further away a value is from the mean, the lower its probability (scaled by the variance σ). By using a Gaussian prior on the weights, we are saying that weights prefer to have the value zero. A Gaussian for a weight w_j is:

$$\frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(w_j - \mu_j)^2}{2\sigma_j^2} \right) \tag{6.89}$$

If we multiply each weight by a Gaussian prior on the weight, we are thus maximizing the following constraint:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \prod_i^M P(y^{(i)}|x^{(i)}) \times \prod_{j=1}^{N} \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(w_j - \mu_j)^2}{2\sigma_j^2} \right) \tag{6.90}$$

which in log space, with $\mu = 0$, corresponds to

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_i \log P(y^{(i)}|x^{(i)}) - \sum_{j=1}^{N} \frac{w_j^2}{2\sigma_j^2} \tag{6.91}$$

which is in the same form as Eq. 6.88.

There is a vast literature on the details of learning in MaxEnt; see the end of the chapter for pointers to further details.

6.7.1 Why do we call it Maximum Entropy?

Why do we refer to multinomial logistic regression models as MaxEnt or Maximum Entropy models? Let's give the intuition of this interpretation in the context of part-of-speech tagging. Suppose we want to assign a tag to the word *zzfish* (a word we made up for this example). What is the probabilistic tagging model (the distribution of part-of-speech tags across words) that makes the fewest assumptions, imposing no constraints at all? Intuitively it would be the equiprobable distribution:

NN	JJ	NNS	VB	NNP	IN	MD	UH	SYM	VBG	POS	PRP	CC	CD	...
$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	$\frac{1}{45}$	...

Now suppose we had some training data labeled with part-of-speech tags, and from this data we learned only one fact: the set of possible tags for *zzfish* are NN, JJ, NNS, and VB (so *zzfish* is a word something like *fish*, but which can also be an adjective). What is the tagging model which relies on this constraint, but makes no further assumptions at all? Since one of these must be the correct tag, we know that

(6.92)
$$P(NN)+P(JJ)+P(NNS)+P(VB)=1$$

Since we have no further information, a model which makes no further assumptions beyond what we know would simply assign equal probability to each of these words:

NN	JJ	NNS	VB	NNP	IN	MD	UH	SYM	VBG	POS	PRP	CC	CD	...
$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	0	0	0	0	0	0	0	0	0	0	...

In the first example, where we wanted an uninformed distribution over 45 parts-of-speech, and in this case, where we wanted an uninformed distribution over 4 parts-of-speech, it turns out that of all possible distributions, the equiprobable distribution has the **maximum entropy**. Recall from Sec. **??** that the entropy of the distribution of a random variable x is computed as:

(6.93)
$$H(x)=-\sum_x P(x)\log_2 P(x)$$

An equiprobable distribution in which all values of the random variable have the same probability has a higher entropy than one in which there is more information. Thus of all distributions over four variables the distribution $\{\frac{1}{4},\frac{1}{4},\frac{1}{4},\frac{1}{4}\}$ has the maximum entropy. (To have an intuition for this, use Eq. 6.93 to compute the entropy for a few other distributions such as the distribution $\{\frac{1}{4},\frac{1}{2},\frac{1}{8},\frac{1}{8}\}$, and make sure they are all lower than the equiprobable distribution.)

The intuition of MaxEnt modeling is that the probabilistic model we are building should follow whatever constraints we impose on it, but beyond these constraints it should follow Occam's Razor, i.e., make the fewest possible assumptions.

Let's add some more constraints into our tagging example. Suppose we looked at our tagged training data and noticed that 8 times out of 10, *zzfish* was tagged as some sort of common noun, either NN or NNS. We can think of this as specifying the feature 'word is *zzfish* and t_i = NN or t_i = NNS'. We might now want to modify our distribution so that we give $\frac{8}{10}$ of our probability mass to nouns, i.e. now we have 2 constraints

$$\begin{gathered} P(NN)+P(JJ)+P(NNS)+P(VB)=1 \\ P(\text{word is } \textit{zzfish} \text{ and } t_i=\text{NN or } t_i=\text{NNS})=\frac{8}{10} \end{gathered}$$

but make no further assumptions (keep JJ and VB equiprobable, and NN and NNS equiprobable).

NN	JJ	NNS	VB	NNP	...
$\frac{4}{10}$	$\frac{1}{10}$	$\frac{4}{10}$	$\frac{1}{10}$	0	...

Now suppose we don't have have any more information about *zzfish*. But we notice in the training data that for all English words (not just *zzfish*) verbs (VB) occur as 1 word in 20. We can now add this constraint (corresponding to the feature $t_i = VB$):

$$P(NN)+P(JJ)+P(NNS)+P(VB) = 1$$

$$P(\text{word is } \textit{zzfish} \text{ and } t_i = \text{NN or } t_i = \text{NNS}) = \frac{8}{10}$$

$$P(VB) = \frac{1}{20}$$

The resulting maximum entropy distribution is now as follows:

NN	JJ	NNS	VB
$\frac{4}{10}$	$\frac{3}{20}$	$\frac{4}{10}$	$\frac{1}{20}$

In summary, the intuition of maximum entropy is to build a distribution by continuously adding features. Each feature is an indicator function, which picks out a subset of the training observations. For each feature we add a constraint on our total distribution, specifying that our distribution for this subset should match the empirical distribution we saw in our training data. We then choose the maximum entropy distribution which otherwise accords with these constraints. Berger et al. (1996) pose the optimization problem of finding this distribution as follows:

> *"To select a model from a set $\mathcal{C}$ of allowed probability distributions, choose the model $p^* \in \mathcal{C}$ with maximum entropy $H(p)$":*

$$p^* = \underset{p \in \mathcal{C}}{\operatorname{argmax}} H(p) \tag{6.94}$$

Now we come to the important conclusion. Berger et al. (1996) show that the solution to this optimization problem turns out to be exactly the probability distribution of a multinomial logistic regression model whose weights W maximize the likelihood of the training data! Thus the exponential model for multinomial logistic regression, when trained according to the maximum likelihood criterion, also finds the maximum entropy distribution subject to the constraints from the feature functions.

6.8 Maximum Entropy Markov Models

We began our discussion of MaxEnt by pointing out that the basic MaxEnt model is not in itself a classifier for sequences. Instead, it is used to classify a single observation into one of a set of discrete classes, as in text classification (choosing between possible authors of an anonymous text, or classifying an email as spam), or tasks like deciding whether a period marks the end of a sentence.

We turn in this section to the **Maximum Entropy Markov Model** or **MEMM**, which is an augmentation of the basic MaxEnt classifier so that it can be applied to assign a class to each element in a sequence, just as we do with HMMs. Why would we want a sequence classifier built on MaxEnt? How might such a classifier be better than an HMM?

Consider the HMM approach to part-of-speech tagging. The HMM tagging model is based on probabilities of the form $P(\text{tag}|\text{tag})$ and $P(\text{word}|\text{tag})$. That means that if we want to include some source of knowledge into the tagging process, we must find a way to encode the knowledge into one of these two probabilities. But many knowledge sources are hard to fit into these models. For example, we saw in Sec. **??** that for tagging unknown words, useful features include capitalization, the presence of hyphens, word endings, and so on. There is no easy way to fit probabilities like $P(\text{capitalization}|\text{tag})$, $P(\text{hyphen}|\text{tag})$, $P(\text{suffix}|\text{tag})$, and so on into an HMM-style model.

We gave the initial part of this intuition in the previous section, when we discussed applying MaxEnt to part-of-speech tagging. Part-of-speech tagging is definitely a sequence labeling task, but we only discussed assigning a part-of-speech tag to a single word.

How can we take this single local classifier and turn it into a general sequence classifier? When classifying each word we can rely on features from the current word, features from surrounding words, as well as the output of the classifier from previous words. For example the simplest method is to run our local classifier left-to-right, first making a hard classification of the first word in the sentence, then the second word, and so on. When classifying each word, we can rely on the output of the classifier from the previous word as a feature. For example, we saw in tagging the word *race* that a useful feature was the tag of the previous word; a previous TO is a good indication that *race* is a VB, whereas a previous DT is a good indication that *race* is a NN. Such a strict left-to-right sliding window approach has been shown to yield surprisingly good results across a wide range of applications.

While it is possible to perform part-of-speech tagging in this way, this simple left-to-right classifier has an important flaw: it makes a hard decision on each word before moving on to the next word. This means that the classifier is unable to use information from later words to inform its decision early on. Recall that in Hidden Markov Models, by contrast, we didn't have to make a hard decision at each word; we used Viterbi decoding to find the sequence of part-of-speech tags which was optimal for the whole sentence.

The Maximum Entropy Markov Model (or MEMM) allows us to achieve this same advantage, by mating the Viterbi algorithm with MaxEnt. Let's see how it works, again looking at part-of-speech tagging. It is easiest to understand an MEMM when comparing it to an HMM. Remember that in using an HMM to model the most probable part-of-speech tag sequence we rely on Bayes rule, computing $P(W|T)P(W)$ instead of directly computing $P(T|W)$:

$$\begin{aligned}\hat{T} &= \underset{T}{\operatorname{argmax}} P(T|W)\\ &= \underset{T}{\operatorname{argmax}} P(W|T)P(T)\end{aligned}$$

$$= \underset{T}{\operatorname{argmax}} \prod_i P(word_i|tag_i) \prod_i P(tag_i|tag_{i-1}) \tag{6.95}$$

That is, an HMM as we've described it is a generative model that optimizes the likelihood $P(W|T)$, and we estimate the posterior by combining the likelihood and the prior $P(T)$.

DISCRIMINATIVE MODEL

In an MEMM, by contrast, we compute the posterior $P(T|W)$ directly. Because we train the model directly to discriminate among the possible tag sequences, we call an MEMM a **discriminative model** rather than a generative model. In an MEMM, we break down the probabilities as follows:

$$\hat{T} = \underset{T}{\operatorname{argmax}} P(T|W)$$

$$= \underset{T}{\operatorname{argmax}} \prod_i P(tag_i|word_i, tag_{i-1}) \tag{6.96}$$

Thus in an MEMM instead of having a separate model for likelihoods and priors, we train a single probabilistic model to estimate $P(tag_i|word_i, tag_{i-1})$. We will use MaxEnt for this last piece, estimating the probability of each local tag given the previous tag, the observed word, and, as we will see, any other features we want to include.

We can see the HMM versus MEMM intuitions of the POS tagging task in Fig. 6.20, which repeats the HMM model of Fig. **??**a from Ch. 5, and adds a new model for the MEMM. Note that the HMM model includes distinct probability estimates for each transition and observation, while the MEMM gives one probability estimate per hidden state, which is the probability of the next tag given the previous tag and the observation.

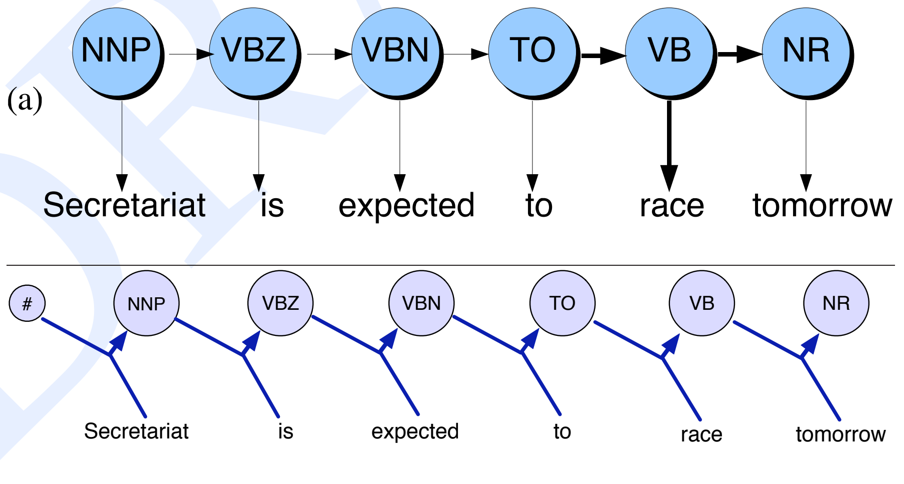


Figure 6.20 The HMM (top) and MEMM (bottom) representation of the probability computation for the correct sequence of tags for the Secretariat sentence. Each arc would be associated with a probability; the HMM computes two separate probabilities for the observation likelihood and the prior, while the MEMM computes a single probability function at each state, conditioned on the previous state and current observation.

Fig. 6.21 emphasizes another advantage of MEMMs over HMMs not shown in Fig. 6.20: unlike the HMM, the MEMM can condition on any useful feature of the input observation. In the HMM this wasn't possible because the HMM is likelihood-based, hence would have needed to compute the likelihood of each feature of the observation.

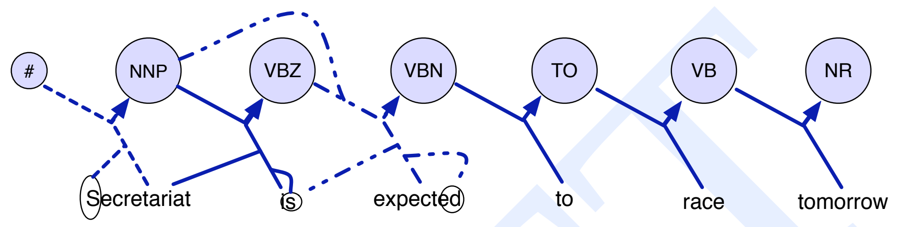


Figure 6.21 An MEMM for part-of-speech tagging, augmenting the description in Fig. 6.20 by showing that an MEMM can condition on many features of the input, such as capitalization, morphology (ending in *-s* or *-ed*), as well as earlier words or tags. We have shown some potential additional features for the first three decisions, using different line styles for each class.

More formally, in the HMM we compute the probability of the state sequence given the observations as:

$$P(Q|O) = \prod_{i=1}^{n} P(o_i|q_i) \times \prod_{i=1}^{n} P(q_i|q_{i-1}) \tag{6.97}$$

In the MEMM, we compute the probability of the state sequence given the observations as:

$$P(Q|O) = \prod_{i=1}^{n} P(q_i|q_{i-1}, o_i) \tag{6.98}$$

In practice, however, an MEMM can also condition on many more features than the HMM, so in general we condition the right-hand side on many more factors.

To estimate the individual probability of a transition from a state q' to a state q producing an observation o, we build a MaxEnt model as follows:

$$P(q|q', o) = \frac{1}{Z(o, q')} \exp\left(\sum_i w_i f_i(o, q)\right) \tag{6.99}$$

6.8.1 Decoding and Learning in MEMMs

Like HMMs, the MEMM uses the Viterbi algorithm to perform the task of decoding (inference). Concretely, this involves filling an $N \times T$ array with the appropriate values for $P(t_i|t_{i-1}, word_i)$, maintaining backpointers as we proceed. As with HMM Viterbi, when the table is filled we simply follow pointers back from the maximum value in the final column to retrieve the desired set of labels. The requisite changes from the HMM-style application of Viterbi only have to do with how we fill each cell. Recall from Eq. 6.23 that the recursive step of the Viterbi equation computes the Viterbi value of time t for state j as:

$$v_t(j) \;=\; \max_{i=1}^{N} \; v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \tag{6.100}$$

which is the HMM implementation of

$$v_t(j) \;=\; \max_{i=1}^{N} \; v_{t-1}(i)\; P(s_j|s_i)\; P(o_t|s_j) \quad 1 \leq j \leq N, 1 < t \leq T \tag{6.101}$$

The MEMM requires only a slight change to this latter formula, replacing the a and b prior and likelihood probabilities with the direct posterior:

$$v_t(j) \;=\; \max_{i=1}^{N} \; v_{t-1}(i)\; P(s_j|s_i, o_t) \quad 1 \leq j \leq N, 1 < t \leq T \tag{6.102}$$

Fig. 6.22 shows an example of the Viterbi trellis for an MEMM applied to the ice-cream task from Sec. 6.4. Recall that the task is figuring out the hidden weather (Hot or Cold) from observed numbers of ice-creams eaten in Jason Eisner's diary. Fig. 6.22 shows the abstract Viterbi probability calculation assuming that we have a MaxEnt model which computes $P(s_i|s_{i-1}, o_i)$ for us.

Learning in MEMMs relies on the same supervised learning algorithms we presented for logistic regression and MaxEnt. Given a sequence of observations, feature functions, and corresponding hidden states, we train the weights so as maximize the log-likelihood of the training corpus. As with HMMs, it is also possible to train MEMMs in semi-supervised modes, for example when the sequence of labels for the training data is missing or incomplete in some way: a version of the EM algorithm can be used for this purpose.

6.9 Summary

This chapter described two important models for probabilistic **sequence classification**: the **Hidden Markov Model** and the **Maximum Entropy Markov Model**. Both models are widely used throughout speech and language processing.

- Hidden Markov Models (**HMMs**) are a way of relating a sequence of **observations** to a sequence of **hidden classes** or **hidden states** which explain the observations.
- The process of discovering the sequence of hidden states given the sequence of observations is known as **decoding** or **inference**. The **Viterbi** algorithm is commonly used for decoding.
- The parameters of an HMM are the A transition probability matrix and the B observation likelihood matrix. Both can be trained using the **Baum-Welch** or **forward-backward** algorithm.

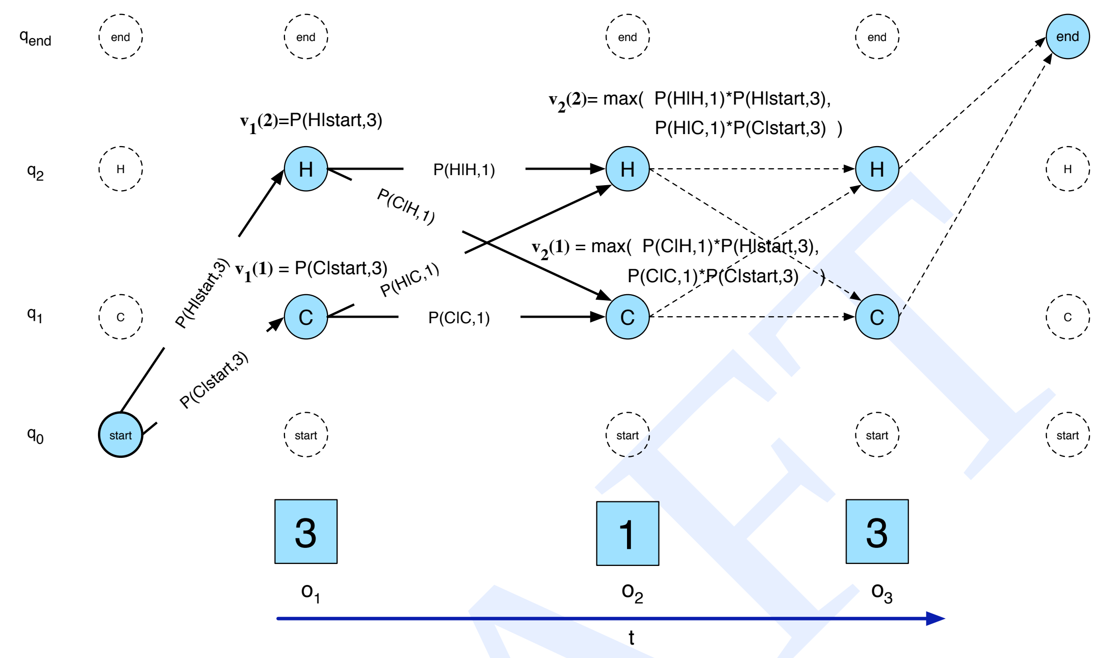


Figure 6.22 Inference from ice-cream eating computed by an MEMM instead of an HMM. The Viterbi trellis for computing the best path through the hidden state space for the ice-cream eating events *3 1 3*, modified from the HMM figure in Fig. 6.10.

- A **MaxEnt** model is a classifier which assigns a **class** to an **observation** by computing a probability from an exponential function of a **weighted** set of **features** of the observation.
- MaxEnt models can be trained using methods from the field of **convex optimization** although we don't give the details in this textbook.
- A **Maximum Entropy Markov Model** or **MEMM** is a sequence model augmentation of MaxEnt which makes use of the Viterbi decoding algorithm.
- MEMMs can be trained by augmenting MaxEnt training with a version of EM.

Bibliographical and Historical Notes

As we discussed at the end of Ch. 4, Markov chains were first used by Markov (1913, 2006), to predict whether an upcoming letter in Pushkin's *Eugene Onegin* would be a vowel or a consonant.

The Hidden Markov Model was developed by Baum and colleagues at the Institute for Defense Analyses in Princeton (Baum and Petrie, 1966; Baum and Eagon, 1967).

The **Viterbi** algorithm was first applied to speech and language processing in the context of speech recognition by Vintsyuk (1968), but has what Kruskal (1983) calls a ‘remarkable history of multiple independent discovery and publication’.[2] Kruskal and others give at least the following independently-discovered variants of the algorithm published in four separate fields:

Citation	Field
Viterbi (1967)	information theory
Vintsyuk (1968)	speech processing
Needleman and Wunsch (1970)	molecular biology
Sakoe and Chiba (1971)	speech processing
Sankoff (1972)	molecular biology
Reichert et al. (1973)	molecular biology
Wagner and Fischer (1974)	computer science

The use of the term **Viterbi** is now standard for the application of dynamic programming to any kind of probabilistic maximization problem in speech and language processing. For non-probabilistic problems (such as for minimum edit distance) the plain term **dynamic programming** is often used. Forney Jr. (1973) is an early survey paper which explores the origin of the Viterbi algorithm in the context of information and communications theory.

Our presentation of the idea that Hidden Markov Models should be characterized by three fundamental problems was modeled after an influential tutorial by Rabiner (1989), which was itself based on tutorials by Jack Ferguson of IDA in the 1960s. Jelinek (1997) and Rabiner and Juang (1993) give very complete descriptions of the forward-backward algorithm, as applied to the speech recognition problem. Jelinek (1997) also shows the relationship between forward-backward and EM. See also the description of HMMs in other textbooks such as Manning and Schütze (1999). Bilmes (1997) is a tutorial on EM.

While logistic regression and other log-linear models have been used in many fields since the middle of the 20th century, the use of Maximum Entropy/multinomial logistic regression in natural language processing dates from work in the early 1990s at IBM (Berger et al., 1996; Della Pietra et al., 1997). This early work introduced the maximum entropy formalism, proposed a learning algorithm (improved iterative scaling), and proposed the use of regularization. A number of applications of MaxEnt followed. For further discussion of regularization and smoothing for maximum entropy models see (*inter alia*) Chen and Rosenfeld (2000), Goodman (2004), and Dudík and Schapire (2006).

Although the second part of this chapter focused on MaxEnt-style classification, numerous other approaches to classification are used throughout speech and language processing. Naive Bayes (Duda et al., 2000) is often employed as a good baseline method (often yielding results that are sufficiently good for practical use); we’ll cover naive Bayes in Ch. 20. Support Vector Machines (Vapnik, 1995) have been successfully used in text classification and in a wide variety of sequence processing applications. Decision lists have been widely used in word sense discrimination, and decision trees (Breiman et al., 1984; Quinlan, 1986) have been used in many applications in speech processing. Good references to supervised machine learning approaches to classifica-

[2] Seven is pretty remarkable, but see page **??** for a discussion of the prevalence of multiple discovery.

tion include Duda et al. (2000), Hastie et al. (2001), and Witten and Frank (2005).

Maximum Entropy Markov Models (MEMMs) were introduced by Ratnaparkhi (1996) and McCallum et al. (2000).

CONDITIONAL RANDOM FIELD

CRF

There are many sequence models that augment the MEMM, such as the **Conditional Random Field** (**CRF**) (Lafferty et al., 2001; Sutton and McCallum, 2006). In addition, there are various generalizations of **maximum margin** methods (the insights that underlie SVM classifiers) to sequence tasks.

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In Shisha, O. (Ed.), *Inequalities III: Proceedings of the Third Symposium on Inequalities*, University of California, Los Angeles, pp. 1–8. Academic Press.

Baum, L. E. and Eagon, J. A. (1967). An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, *73*(3), 360–363.

Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite-state Markov chains. *Annals of Mathematical Statistics*, *37*(6), 1554–1563.

Berger, A., Della Pietra, S. A., and Della Pietra, V. J. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, *22*(1), 39–71.

Bilmes, J. (1997). A gentle tutorial on the EM algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Tech. rep. ICSI-TR-97-021, ICSI, Berkeley.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth & Brooks, Pacific Grove, CA.

Chen, S. F. and Rosenfeld, R. (2000). A survey of smoothing techniques for ME models. *IEEE Transactions on Speech and Audio Processing*, *8*(1), 37–50.

Darroch, J. N. and Ratcliff, D. (1972). Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, *43*(5), 1470–1480.

Della Pietra, S. A., Della Pietra, V. J., and Lafferty, J. D. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(4), 380–393.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the *EM* algorithm. *Journal of the Royal Statistical Society*, *39*(1), 1–21.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification*. Wiley-Interscience Publication.

Dudík, M. and Schapire, R. E. (2006). Maximum entropy distribution estimation with generalized regularization. In Lugosi, G. and Simon, H. U. (Eds.), *COLT 2006*, Berlin, pp. 123–138. Springer-Verlag.

Eisner, J. (2002). An interactive spreadsheet for teaching the forward-backward algorithm. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*, pp. 10–18.

Forney Jr., G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, *61*(3), 268–278.

Goodman, J. (2004). Exponential priors for maximum entropy models. In *ACL-04*.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2001). *The Elements of Statistical Learning*. Springer.

Hofstadter, D. R. (1997). *Le ton beau de marot*. Basic Books.

Jelinek, F. (1997). *Statistical Methods for Speech Recognition*. MIT Press.

Kruskal, J. B. (1983). An overview of sequence comparison. In Sankoff, D. and Kruskal, J. B. (Eds.), *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pp. 1–44. Addison-Wesley, Reading, MA.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML 2001*, Stanford, CA.

Levitt, S. D. and Dubner, S. J. (2005). *Freakonomics*. Morrow.

Malouf, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *CoNLL-2002*, pp. 49–55.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Markov, A. A. (1913). Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain'). *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, *7*, 153–162. English translation by Morris Halle, 1956.

Markov, A. A. (2006). Classical text in translation: A. A. Markov, an example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains. *Science in Context*, *19*(4), 591–600. Translated by David Link.

McCallum, A., Freitag, D., and Pereira, F. C. N. (2000). Maximum Entropy Markov Models for Information Extraction and Segmentation. In *ICML 2000*, pp. 591–598.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino-acid sequence of two proteins. *Journal of Molecular Biology*, *48*, 443–453.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286.

Rabiner, L. R. and Juang, B. H. (1993). *Fundamentals of Speech Recognition*. Prentice Hall.

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. In *EMNLP 1996*, Philadelphia, PA, pp. 133–142.

Reichert, T. A., Cohen, D. N., and Wong, A. K. C. (1973). An application of information theory to genetic mutations and the matching of polypeptide sequences. *Journal of Theoretical Biology*, *42*, 245–261.

Sakoe, H. and Chiba, S. (1971). A dynamic programming approach to continuous speech recognition. In *Proceedings of the Seventh International Congress on Acoustics, Budapest*, Budapest, Vol. 3, pp. 65–69. Akadémiai Kiadó.

Sankoff, D. (1972). Matching sequences under deletion-insertion constraints. *Proceedings of the Natural Academy of Sciences of the U.S.A.*, *69*, 4–6.

Sutton, C. and McCallum, A. (2006). An introduction to conditional random fields for relational learning. In Getoor, L. and Taskar, B. (Eds.), *Introduction to Statistical Relational Learning*. MIT Press.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag.

Vintsyuk, T. K. (1968). Speech discrimination by dynamic programming. *Cybernetics*, *4*(1), 52–57. Russian Kibernetika 4(1):81-88 (1968).

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, *IT-13*(2), 260–269.

Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery*, *21*, 168–173.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann. 2nd ed.

7 PHONETICS

(Upon being asked by Director George Cukor to teach Rex Harrison, the star of the 1964 film ”My Fair Lady”, how to behave like a phonetician:)

> *“My immediate answer was, ‘I don’t have a singing butler and three maids who sing, but I will tell you what I can as an assistant professor.’”*
>
> Peter Ladefoged, quoted in his obituary, LA Times, 2004

The debate between the “whole language” and “phonics” methods of teaching reading to children seems at very glance like a purely modern educational debate. Like many modern debates, however, this one recapitulates an important historical dialectic, in this case in writing systems. The earliest independently-invented writing systems (Sumerian, Chinese, Mayan) were mainly logographic: one symbol represented a whole word. But from the earliest stages we can find, most such systems contain elements of syllabic or phonemic writing systems, in which symbols are used to represent the sounds that make up the words. Thus the Sumerian symbol pronounced *ba* and meaning “ration” could also function purely as the sound /ba/. Even modern Chinese, which remains primarily logographic, uses sound-based characters to spell out foreign words. Purely sound-based writing systems, whether syllabic (like Japanese *hiragana* or *katakana*), alphabetic (like the Roman alphabet used in this book), or consonantal (like Semitic writing systems), can generally be traced back to these early logo-syllabic systems, often as two cultures came together. Thus the Arabic, Aramaic, Hebrew, Greek, and Roman systems all derive from a West Semitic script that is presumed to have been modified by Western Semitic mercenaries from a cursive form of Egyptian hieroglyphs. The Japanese syllabaries were modified from a cursive form of a set of Chinese characters which were used to represent sounds. These Chinese characters themselves were used in Chinese to phonetically represent the Sanskrit in the Buddhist scriptures that were brought to China in the Tang dynasty.

Whatever its origins, the idea implicit in a sound-based writing system, that the spoken word is composed of smaller units of speech, is the Ur-theory that underlies all our modern theories of **phonology**. This idea of decomposing speech and words into smaller units also underlies the modern algorithms for **speech recognition** (transcrbining acoustic waveforms into strings of text words) and **speech synthesis** or **text-to-speech** (converting strings of text words into acoustic waveforms).

In this chapter we introduce **phonetics** from a computational perspective. Phonetics is the study of linguistic sounds, how they are produced by the articulators of the human vocal tract, how they are realized acoustically, and how this acoustic realization can be digitized and processed.

We begin with a key element of both speech recognition and text-to-speech systems: how words are pronounced in terms of individual speech units called **phones**. A speech recognition system needs to have a pronunciation for every word it can recognize, and a text-to-speech system needs to have a pronunciation for every word it can say. The first section of this chapter will introduce **phonetic alphabets** for describing these pronunciations. We then introduce the two main areas of phonetics, **articulatory phonetics**, the study of how speech sounds are produced by articulators in the mouth, and **acoustic phonetics**, the study of the acoustic analysis of speech sounds.

We also briefly touch on **phonology**, the area of linguistics that describes the systematic way that sounds are differently realized in different environments, and how this system of sounds is related to the rest of the grammar. In doing so we focus on the crucial fact of **variation** in modeling speech; phones are pronounced differently in different contexts.

7.1 Speech Sounds and Phonetic Transcription

PHONETICS

PHONES

The study of the pronunciation of words is part of the field of **phonetics**, the study of the speech sounds used in the languages of the world. We model the pronunciation of a word as a string of symbols which represent **phones** or **segments**. A phone is a speech sound; phones are represented with phonetic symbols that bear some resemblance to a letter in an alphabetic language like English.

IPA

This section surveys the different phones of English, particularly American English, showing how they are produced and how they are represented symbolically. We will be using two different alphabets for describing phones. The **International Phonetic Alphabet** (**IPA**) is an evolving standard originally developed by the International Phonetic Association in 1888 with the goal of transcribing the sounds of all human languages. The IPA is not just an alphabet but also a set of principles for transcription, which differ according to the needs of the transcription, so the same utterance can be transcribed in different ways all according to the principles of the IPA. The ARPAbet (Shoup, 1980) is another phonetic alphabet, but one that is specifically designed for American English and which uses ASCII symbols; it can be thought of as a convenient ASCII representation of an American-English subset of the IPA. ARPAbet symbols are often used in applications where non-ASCII fonts are inconvenient, such as in on-line pronunciation dictionaries. Because the ARPAbet is very common for computational representations of pronunciations, we will rely on it rather than the IPA in the remainder of this book. Fig. 7.1 and Fig. 7.2 show the ARPAbet symbols for transcribing consonants and vowels, respectively, together with their IPA equivalents.

[1] The phone [ux] is rare in general American English and not generally used in speech systems. It is used to represent the fronted [uw] which appeared in (at least) Western and Northern Cities dialects of American English starting in the late 1970s (Labov, 1994). This fronting was first called to public by imitations

ARPAbet	IPA		ARPAbet
Symbol	Symbol	Word	Transcription
[p]	[p]	parsley	[p aa r s l iy]
[t]	[t]	tea	[t iy]
[k]	[k]	cook	[k uh k]
[b]	[b]	bay	[b ey]
[d]	[d]	dill	[d ih l]
[g]	[g]	garlic	[g aa r l ix k]
[m]	[m]	mint	[m ih n t]
[n]	[n]	nutmeg	[n ah t m eh g]
[ng]	[ŋ]	baking	[b ey k ix ng]
[f]	[f]	flour	[f l aw axr]
[v]	[v]	clove	[k l ow v]
[th]	[θ]	thick	[th ih k]
[dh]	[ð]	those	[dh ow z]
[s]	[s]	soup	[s uw p]
[z]	[z]	eggs	[eh g z]
[sh]	[ʃ]	squash	[s k w aa sh]
[zh]	[ʒ]	ambrosia	[ae m b r ow zh ax]
[ch]	[tʃ]	cherry	[ch eh r iy]
[jh]	[dʒ]	jar	[jh aa r]
[l]	[l]	licorice	[l ih k axr ix sh]
[w]	[w]	kiwi	[k iy w iy]
[r]	[r]	rice	[r ay s]
[y]	[j]	yellow	[y eh l ow]
[h]	[h]	honey	[h ah n iy]
Less commonly used phones and allophones			
[q]	[ʔ]	uh-oh	[q ah q ow]
[dx]	[ɾ]	butter	[b ah dx axr]
[nx]	[ɾ̃]	winner	[w ih nx axr]
[el]	[l̩]	table	[t ey b el]

Figure 7.1 ARPAbet symbols for transcription of English consonants, with IPA equivalents. Note that some rarer symbols like the flap [dx], nasal flap [nx], glottal stop [q] and the syllabic consonants, are used mainly for narrow transcriptions.

Many of the IPA and ARPAbet symbols are equivalent to the Roman letters used in the orthography of English and many other languages. So for example the ARPAbet phone [p] represents the consonant sound at the beginning of *platypus*, *puma*, and *pachyderm*, the middle of *leopard*, or the end of *antelope*. In general, however, the mapping between the letters of English orthography and phones is relatively **opaque**; a single letter can represent very different sounds in different contexts. The English letter *c* corresponds to phone [k] in *cougar* [k uw g axr], but phone [s] in *cell* [s eh

and recordings of ‘Valley Girls’ speech by Moon Zappa (Zappa and Zappa, 1982). Nevertheless, for most speakers [uw] is still much more common than [ux] in words like *dude*.

ARPAbet Symbol	IPA Symbol	Word	ARPAbet Transcription
[iy]	[i]	lily	[l ih l iy]
[ih]	[ɪ]	lily	[l ih l iy]
[ey]	[eɪ]	daisy	[d ey z iy]
[eh]	[ɛ]	pen	[p eh n]
[ae]	[æ]	aster	[ae s t axr]
[aa]	[ɑ]	poppy	[p aa p iy]
[ao]	[ɔ]	orchid	[ao r k ix d]
[uh]	[ʊ]	wood	[w uh d]
[ow]	[oʊ]	lotus	[l ow dx ax s]
[uw]	[u]	tulip	[t uw l ix p]
[ah]	[ʌ]	buttercup	[b ah dx axr k ah p]
[er]	[ɝ]	bird	[b er d]
[ay]	[aɪ]	iris	[ay r ix s]
[aw]	[aʊ]	sunflower	[s ah n f l aw axr]
[oy]	[oɪ]	soil	[s oy l]
Reduced and uncommon phones			
[ax]	[ə]	lotus	[l ow dx ax s]
[axr]	[ɚ]	heather	[h eh dh axr]
[ix]	[ɨ]	tulip	[t uw l ix p]
[ux]	[ʉ]	dude[1]	[d ux d]

Figure 7.2 ARPAbet symbols for transcription of English vowels, with IPA equivalents. Note again the list of rarer phones and reduced vowels (see Sec. 7.2.4); for example [ax] is the reduced vowel schwa, [ix] is the reduced vowel corresponding to [ih], and [axr] is the reduced vowel corresponding to [er].

l]. Besides appearing as *c* and *k*, the phone [k] can appear as part of *x* (*fox* [f aa k s]), as *ck* (*jackal* [jh ae k el] and as *cc* (*raccoon* [r ae k uw n]). Many other languages, for example Spanish, are much more **transparent** in their sound-orthography mapping than English.

7.2 Articulatory Phonetics

ARTICULATORY PHONETICS

The list of ARPAbet phones is useless without an understanding of how each phone is produced. We thus turn to **articulatory phonetics**, the study of how phones are produced, as the various organs in the mouth, throat, and nose modify the airflow from the lungs.

7.2.1 The Vocal Organs

Sound is produced by the rapid movement of air. Most sounds in human spoken languages are produced by expelling air from the lungs through the windpipe (technically the **trachea**) and then out the mouth or nose. As it passes through the trachea,

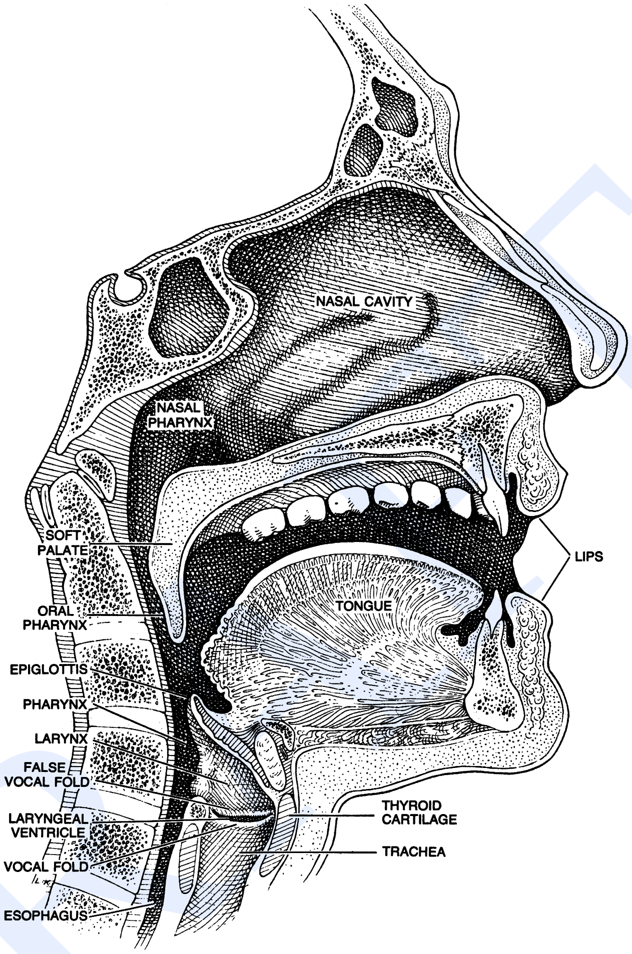


Figure 7.3 The vocal organs, shown in side view. Drawing by Laszlo Kubinyi from Sundberg (1977), ©Scientific American, used by permission.

the air passes through the **larynx**, commonly known as the Adam's apple or voicebox. The larynx contains two small folds of muscle, the **vocal folds** (often referred to non-technically as the **vocal cords**) which can be moved together or apart. The space between these two folds is called the **glottis**. If the folds are close together (but not tightly closed), they will vibrate as air passes through them; if they are far apart, they won't vibrate. Sounds made with the vocal folds together and vibrating are called **voiced**; sounds made without this vocal cord vibration are called **unvoiced** or **voiceless**. Voiced sounds include [b], [d], [g], [v], [z], and all the English vowels, among others. Unvoiced sounds include [p], [t], [k], [f], [s], and others.

GLOTTIS
VOICED
UNVOICED
VOICELESS

The area above the trachea is called the **vocal tract**, and consists of the **oral tract** and the **nasal tract**. After the air leaves the trachea, it can exit the body through the

NASAL SOUNDS

mouth or the nose. Most sounds are made by air passing through the mouth. Sounds made by air passing through the nose are called **nasal sounds**; nasal sounds use both the oral and nasal tracts as resonating cavities; English nasal sounds include *m*, and *n*, and *ng*.

CONSONANTS
VOWELS

Phones are divided into two main classes: **consonants** and **vowels**. Both kinds of sounds are formed by the motion of air through the mouth, throat or nose. Consonants are made by restricting or blocking the airflow in some way, and may be voiced or unvoiced. Vowels have less obstruction, are usually voiced, and are generally louder and longer-lasting than consonants. The technical use of these terms is much like the common usage; [p], [b], [t], [d], [k], [g], [f], [v], [s], [z], [r], [l], etc., are consonants; [aa], [ae], [ao], [ih], [aw], [ow], [uw], etc., are vowels. **Semivowels** (such as [y] and [w]) have some of the properties of both; they are voiced like vowels, but they are short and less syllabic like consonants.

7.2.2 Consonants: Place of Articulation

PLACE

Because consonants are made by restricting the airflow in some way, consonants can be distinguished by where this restriction is made: the point of maximum restriction is called the **place of articulation** of a consonant. Places of articulation, shown in Fig. 7.4, are often used in automatic speech recognition as a useful way of grouping phones together into equivalence classes:

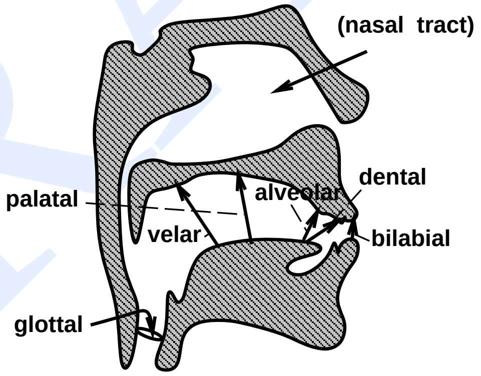


Figure 7.4 Major English places of articulation.

LABIAL

labial: Consonants whose main restriction is formed by the two lips coming together have a **bilabial** place of articulation. In English these include [p] as in *possum*, [b] as in *bear*, and [m] as in *marmot*. The English **labiodental** consonants [v] and [f] are made by pressing the bottom lip against the upper row of teeth and letting the air flow through the space in the upper teeth.

DENTAL

dental: Sounds that are made by placing the tongue against the teeth are dentals. The main dentals in English are the [th] of *thing* or the [dh] of *though*, which are made by placing the tongue behind the teeth with the tip slightly between the teeth.

ALVEOLAR
CORONAL

alveolar: The alveolar ridge is the portion of the roof of the mouth just behind the upper teeth. Most speakers of American English make the phones [s], [z], [t], and [d] by placing the tip of the tongue against the alveolar ridge. The word **coronal** is often used to refer to both dental and alveolar.

PALATAL
PALATE

palatal: The roof of the mouth (the **palate**) rises sharply from the back of the alveolar ridge. The **palato-alveolar** sounds [sh] (*shrimp*), [ch] (*china*), [zh] (*Asian*), and [jh] (*jar*) are made with the blade of the tongue against this rising back of the alveolar ridge. The palatal sound [y] of *yak* is made by placing the front of the tongue up close to the palate.

VELAR
VELUM

velar: The **velum** or soft palate is a movable muscular flap at the very back of the roof of the mouth. The sounds [k] (*cuckoo*), [g] (*goose*), and [ŋ] (*kingfisher*) are made by pressing the back of the tongue up against the velum.

GLOTTAL

glottal: The glottal stop [q] (IPA [ʔ]) is made by closing the glottis (by bringing the vocal folds together).

7.2.3 Consonants: Manner of Articulation

MANNER

Consonants are also distinguished by *how* the restriction in airflow is made, for example whether there is a complete stoppage of air, or only a partial blockage, etc. This feature is called the **manner of articulation** of a consonant. The combination of place and manner of articulation is usually sufficient to uniquely identify a consonant. Following are the major manners of articulation for English consonants:

STOP

A **stop** is a consonant in which airflow is completely blocked for a short time. This blockage is followed by an explosive sound as the air is released. The period of blockage is called the **closure** and the explosion is called the **release**. English has voiced stops like [b], [d], and [g] as well as unvoiced stops like [p], [t], and [k]. Stops are also called **plosives**. Some computational systems use a more narrow (detailed) transcription style that has separate labels for the closure and release parts of a stop. In one version of the ARPAbet, for example, the closure of a [p], [t], or [k] is represented as [pcl], [tcl], or [kcl] (respectively), while the symbols [p], [t], and [k] are used to mean only the release portion of the stop. In another version the symbols [pd], [td], [kd], [bd], [dd], [gd] are used to mean unreleased stops (stops at the end of words or phrases often are missing the explosive release), while [p], [t], [k], etc are used to mean normal stops with a closure and a release. The IPA uses a special symbol to mark unreleased stops: [p̚], [t̚], or [k̚]. We will not be using these narrow transcription styles in this chapter; we will always use [p] to mean a full stop with both a closure and a release.

NASAL

The **nasal** sounds [n], [m], and [ng] are made by lowering the velum and allowing air to pass into the nasal cavity.

FRICATIVES

In **fricatives**, airflow is constricted but not cut off completely. The turbulent airflow that results from the constriction produces a characteristic "hissing" sound. The English labiodental fricatives [f] and [v] are produced by pressing the lower lip against the upper teeth, allowing a restricted airflow between the upper teeth. The dental fricatives [th] and [dh] allow air to flow around the tongue between the teeth. The alveolar fricatives [s] and [z] are produced with the tongue against the alveolar ridge, forcing air over the edge of the teeth. In the palato-alveolar fricatives [sh] and [zh] the tongue is at the back of the alveolar ridge forcing air through a groove formed in the tongue. The higher-pitched fricatives (in English [s], [z], [sh] and [zh] are called **sibilants**. Stops that are followed immediately by fricatives are called **affricates**; these include English [ch] (*chicken*) and [jh] (*giraffe*).

SIBILANTS

APPROXIMANTS

In **approximants**, the two articulators are close together but not close enough to cause turbulent airflow. In English [y] (*yellow*), the tongue moves close to the roof of the mouth but not close enough to cause the turbulence that would characterize a fricative. In English [w] (*wood*), the back of the tongue comes close to the velum. American [r] can be formed in at least two ways; with just the tip of the tongue extended and close to the palate or with the whole tongue bunched up near the palate. [l] is formed with the tip of the tongue up against the alveolar ridge or the teeth, with one or both sides of the tongue lowered to allow air to flow over it. [l] is called a **lateral** sound because of the drop in the sides of the tongue.

TAP

FLAP

A **tap** or **flap** [dx] (or IPA [ɾ]) is a quick motion of the tongue against the alveolar ridge. The consonant in the middle of the word *lotus* ([l ow dx ax s]) is a tap in most dialects of American English; speakers of many UK dialects would use a [t] instead of a tap in this word.

7.2.4 Vowels

Like consonants, vowels can be characterized by the position of the articulators as they are made. The three most relevant parameters for vowels are what is called vowel **height**, which correlates roughly with the height of the highest part of the tongue, vowel **frontness** or **backness**, which indicates whether this high point is toward the front or back of the oral tract, and the shape of the lips (**rounded** or not). Fig. 7.5 shows the position of the tongue for different vowels.

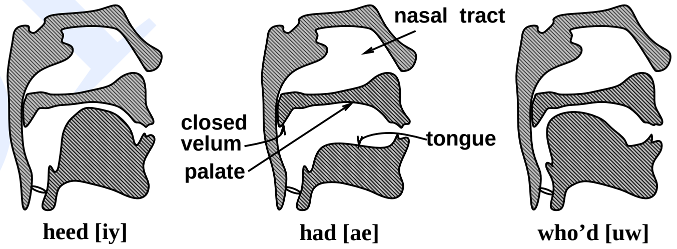


Figure 7.5 Positions of the tongue for three English vowels, high front [iy], low front [ae] and high back [uw]; tongue positions modeled after Ladefoged (1996).

In the vowel [iy], for example, the highest point of the tongue is toward the front of the mouth. In the vowel [uw], by contrast, the high-point of the tongue is located toward the back of the mouth. Vowels in which the tongue is raised toward the front are called **front vowels**; those in which the tongue is raised toward the back are called

FRONT

BACK HIGH

back vowels. Note that while both [ih] and [eh] are front vowels, the tongue is higher for [ih] than for [eh]. Vowels in which the highest point of the tongue is comparatively high are called **high vowels**; vowels with mid or low values of maximum tongue height are called **mid vowels** or **low vowels**, respectively.

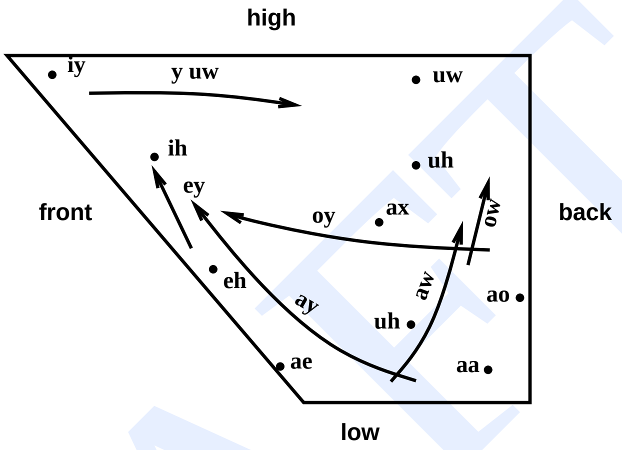


Figure 7.6 Qualities of English vowels (after Ladefoged (1993)).

DIPHTHONG

Fig. 7.6 shows a schematic characterization of the vowel height of different vowels. It is schematic because the abstract property **height** only correlates roughly with actual tongue positions; it is in fact a more accurate reflection of acoustic facts. Note that the chart has two kinds of vowels: those in which tongue height is represented as a point and those in which it is represented as a vector. A vowel in which the tongue position changes markedly during the production of the vowel is a **diphthong**. English is particularly rich in diphthongs.

ROUNDED

The second important articulatory dimension for vowels is the shape of the lips. Certain vowels are pronounced with the lips rounded (the same lip shape used for whistling). These **rounded** vowels include [uw], [ao], and [ow].

Syllables

SYLLABLE NUCLEUS ONSET CODA RIME RHYME

Consonants and vowels combine to make a **syllable**. There is no completely agreed-upon definition of a syllable; roughly speaking a syllable is a vowel-like (or **sonorant**) sound together with some of the surrounding consonants that are most closely associated with it. The word *dog* has one syllable, [d aa g], while the word *catnip* has two syllables, [k ae t] and [n ih p], We call the vowel at the core of a syllable the **nucleus**. The optional initial consonant or set of consonants is called the **onset**. If the onset has more than one consonant (as in the word *strike* [s t r ay k]), we say it has a **complex onset**. The **coda**. is the optional consonant or sequence of consonants following the nucleus. Thus [d] is the onset of *dog*, while [g] is the coda. The **rime**. or **rhyme**. is the nucleus plus coda. Fig. 7.7 shows some sample syllable structures.

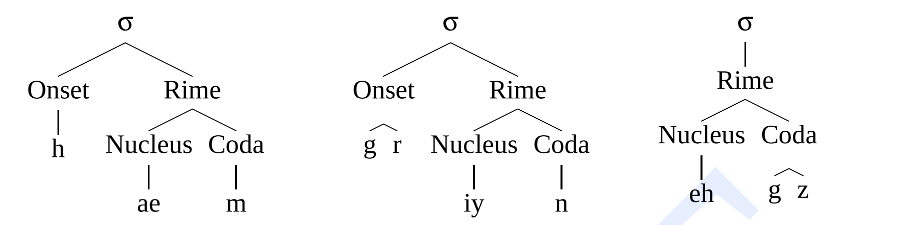


Figure 7.7 Syllable structure of *ham*, *green*, *eggs*. σ=syllable.

SYLLABIFICATION

The task of automatically breaking up a word into syllables is called **syllabification**, and will be discussed in Sec. **??**.

PHONOTACTICS

Syllable structure is also closely related to the **phonotactics** of a language. The term **phonotactics** means the constraints on which phones can follow each other in a language. For example, English has strong constraints on what kinds of consonants can appear together in an onset; the sequence [zdr], for example, cannot be a legal English syllable onset. Phonotactics can be represented by listing constraints on fillers of syllable positions, or by creating a finite-state model of possible phone sequences. It is also possible to create a probabilistic phonotactics, by training *N*-gram grammars on phone sequences.

Lexical Stress and Schwa

ACCENTED
PITCH ACCENT
BEAR

In a natural sentence of American English, certain syllables are more **prominent** than others. These are called **accented** syllables, and the linguistic marker associated with this prominence is called a **pitch accent**. Words or syllables which are prominent are said to **bear** (be associated with) a pitch accent. Pitch accent is also sometimes referred to as **sentence stress**, although sentence stress can instead refer to only the most prominent accent in a sentence.

LEXICAL STRESS

Accented syllables may be prominent by being louder, longer, by being associated with a pitch movement, or by any combination of the above. Since accent plays important roles in meaning, understanding exactly why a speaker chooses to accent a particular syllable is very complex, and we will return to this in detail in Sec. **??**. But one important factor in accent is often represented in pronunciation dictionaries. This factor is called **lexical stress**. The syllable that has lexical stress is the one that will be louder or longer if the word is accented. For example the word *parsley* is stressed in its first syllable, not its second. Thus if the word *parsley* receives a pitch accent in a sentence, it is the first syllable that will be stronger.

In IPA we write the symbol [ˈ] before a syllable to indicate that it has lexical stress (e.g. [ˈpɑr.sli]). This difference in lexical stress can affect the meaning of a word. For example the word *content* can be a noun or an adjective. When pronounced in isolation the two senses are pronounced differently since they have different stressed syllables (the noun is pronounced [ˈkɑn.tɛnt] and the adjective [kən.ˈtɛnt]).

REDUCED VOWELS
SCHWA

Vowels which are unstressed can be weakened even further to **reduced vowels**. The most common reduced vowel is **schwa** ([ax]). Reduced vowels in English don't have their full form; the articulatory gesture isn't as complete as for a full vowel. As a result the shape of the mouth is somewhat neutral; the tongue is neither particularly high nor particularly low. For example the second vowel in *parakeet* is a schwa: [p ae r ax k iy t].

While schwa is the most common reduced vowel, it is not the only one, at least not in some dialects. Bolinger (1981) proposed that American English had three reduced vowels: a reduced mid vowel [ə], a reduced front vowel [ɨ], and a reduced rounded vowel [ɵ]. The full ARPAbet includes two of these, the schwa [ax] and [ix] ([ɨ]), as well as [axr] which is an r-colored schwa (often called **schwar**), although [ix] is generally dropped in computational applications (Miller, 1998), and [ax] and [ix] are falling together in many dialects of English Wells (1982, p. 167–168).

Not all unstressed vowels are reduced; any vowel, and diphthongs in particular can retain their full quality even in unstressed position. For example the vowel [iy] can appear in stressed position as in the word *eat* [iy t] or in unstressed position in the word *carry* [k ae r iy].

SECONDARY STRESS

Some computational ARPAbet lexicons mark reduced vowels like schwa explicitly. But in general predicting reduction requires knowledge of things outside the lexicon (the prosodic context, rate of speech, etc, as we will see the next section). Thus other ARPAbet versions mark stress but don't mark how stress affects reduction. The CMU dictionary (CMU, 1993), for example, marks each vowel with the number 0 (unstressed) 1 (stressed), or 2 (secondary stress). Thus the word *counter* is listed as [K AW1 N T ER0], and the word *table* as [T EY1 B AH0 L]. **Secondary stress** is defined as a level of stress lower than primary stress, but higher than an unstressed vowel, as in the word *dictionary* [D IH1 K SH AH0 N EH2 R IY0]

PROMINENCE

We have mentioned a number of potential levels of **prominence**: accented, stressed, secondary stress, full vowel, and reduced vowel. It is still an open research question exactly how many levels are appropriate. Very few computational systems make use of all five of these levels, most using between one and three. We return to this discussion when we introduce prosody in more detail in Sec. **??**.

7.3 Phonological Categories and Pronunciation Variation

'Scuse me, while I kiss the sky
Jimi Hendrix, *Purple Haze*
'Scuse me, while I kiss this guy
Common mis-hearing of same lyrics

If each word was pronounced with a fixed string of phones, each of which was pronounced the same in all contexts and by all speakers, the speech recognition and speech synthesis tasks would be really easy. Alas, the realization of words and phones varies massively depending on many factors. Fig. 7.8 shows a sample of the wide variation in pronunciation in the words *because* and *about* from the hand-transcribed Switchboard corpus of American English telephone conversations (Greenberg et al., 1996).

How can we model and predict this extensive variation? One useful tool is the assumption that what is mentally represented in the speaker's mind are abstract cate-

because				about			
ARPAbet	**%**	**ARPAbet**	**%**	**ARPAbet**	**%**	**ARPAbet**	**%**
b iy k ah z	27%	k s	2%	ax b aw	32%	b ae	3%
b ix k ah z	14%	k ix z	2%	ax b aw t	16%	b aw t	3%
k ah z	7%	k ih z	2%	b aw	9%	ax b aw dx	3%
k ax z	5%	b iy k ah zh	2%	ix b aw	8%	ax b ae	3%
b ix k ax z	4%	b iy k ah s	2%	ix b aw t	5%	b aa	3%
b ih k ah z	3%	b iy k ah	2%	ix b ae	4%	b ae dx	3%
b ax k ah z	3%	b iy k aa z	2%	ax b ae dx	3%	ix b aw dx	2%
k uh z	2%	ax z	2%	b aw dx	3%	ix b aa t	2%

Figure 7.8 The 16 most common pronunciations of *because* and *about* from the hand-transcribed Switchboard corpus of American English conversational telephone speech (Godfrey et al., 1992; Greenberg et al., 1996).

gories rather than phones in all their gory phonetic detail. For example consider the different pronunciations of [t] in the words *tunafish* and *starfish*. The [t] of *tunafish* is **aspirated**. Aspiration is a period of voicelessness after a stop closure and before the onset of voicing of the following vowel. Since the vocal cords are not vibrating, aspiration sounds like a puff of air after the [t] and before the vowel. By contrast, a [t] following an initial [s] is **unaspirated**; thus the [t] in *starfish* ([s t aa r f ih sh]) has no period of voicelessness after the [t] closure. This variation in the realization of [t] is predictable: whenever a [t] begins a word or unreduced syllable in English, it is aspirated. The same variation occurs for [k]; the [k] of *sky* is often mis-heard as [g] in Jimi Hendrix's lyrics because [k] and [g] are both unaspirated.[2]

UNASPIRATED

There are other contextual variants of [t]. For example, when [t] occurs between two vowels, particularly when the first is stressed, it is often pronounced as a **tap**. Recall that a tap is a voiced sound in which the top of the tongue is curled up and back and struck quickly against the alveolar ridge. Thus the word *buttercup* is usually pronounced [b ah dx axr k uh p] rather than [b ah t axr k uh p]. Another variant of [t] occurs before the dental consonant [th]. Here the [t] becomes dentalized (IPA [t̪]). That is, instead of the tongue forming a closure against the alveolar ridge, the tongue touches the back of the teeth.

PHONEME

ALLOPHONES

In both linguistics and in speech processing, we use abstract classes to capture the similarity among all these [t]s. The simplest abstract class is called the **phoneme**, and its different surface realizations in different contexts are called **allophones**. We traditionally write phonemes inside slashes. So in the above examples, /t/ is a phoneme whose allophones include (in IPA) [t^{h}], [ɾ], and [t̪]. Fig. 7.9 summarizes a number of allophones of /t/. In speech synthesis and recognition, we use phonesets like the ARPAbet to approximate this idea of abstract phoneme units, and represent pronunciation lexicons using ARPAbet phones. For this reason, the allophones listed in Fig. 7.1 tend to be used for narrow transcriptions for analysis purposes, and less often used in speech recognition or synthesis systems.

[2] The ARPAbet does not have a way of marking aspiration; in the IPA aspiration is marked as [^{h}], so in IPA the word *tunafish* would be transcribed [t^{h}unəfɪʃ].

IPA	ARPABet	Description	Environment	Example
tʰ	[t]	aspirated	in initial position	*toucan*
t		unaspirated	after [s] or in reduced syllables	*starfish*
ʔ	[q]	glottal stop	word-finally or after vowel before [n]	*kitten*
ʔt	[qt]	glottal stop t	sometimes word-finally	*cat*
ɾ	[dx]	tap	between vowels	*butter*
t̚	[tcl]	unreleased t	before consonants or word-finally	*fruitcake*
t̪		dental t	before dental consonants ([θ])	*eighth*
		deleted t	sometimes word-finally	*past*

Figure 7.9 Some allophones of /t/ in General American English.

REDUCTION

HYPOARTICULATION

ASSIMILATION

PALATALIZATION

Variation is even more common than Fig. 7.9 suggests. One factor influencing variation is that the more natural and colloquial speech becomes, and the faster the speaker talks, the more the sounds are shortened, reduced and generally run together. This phenomena is known as **reduction** or **hypoarticulation**. For example **assimilation** is the change in a segment to make it more like a neighboring segment. The dentalization of [t] to ([t̪]) before the dental consonant [θ] is an example of assimilation. A common type of assimilation cross-linguistically is **palatalization**, when the constriction for a segment moves closer to the palate than it normally would, because the following segment is palatal or alveolo-palatal. In the most common cases, /s/ becomes [sh], /z/ becomes [zh], /t/ becomes [ch] and /d/ becomes [jh], We saw one case of palatalization in Fig. 7.8 in the pronunciation of *because* as [b iy k ah zh], because the following word was *you've*. The lemma *you* (*you*, *your*, *you've*, and *you'd*) is extremely likely to cause palatalization in the Switchboard corpus.

DELETION

Deletion is quite common in English speech. We saw examples of deletion of final /t/ above, in the words *about* and *it*. Deletion of final /t/ and /d/ has been extensively studied. /d/ is more likely to be deleted than /t/, and both are more likely to be deleted before a consonant (Labov, 1972). Fig. 7.10 shows examples of palatalization and final t/d deletion from the Switchboard corpus.

Palatalization			Final t/d Deletion		
Phrase	**Lexical**	**Reduced**	**Phrase**	**Lexical**	**Reduced**
set your	s eh t y ow r	s eh ch er	find him	f ay n d h ih m	f ay n ix m
not yet	n aa t y eh t	n aa ch eh t	and we	ae n d w iy	eh n w iy
did you	d ih d y uw	d ih jh y ah	draft the	d r ae f t dh iy	d r ae f dh iy

Figure 7.10 Examples of palatalization and final t/d/ deletion from the Switchboard corpus. Some of the t/d examples may have glottalization instead of being completely deleted.

7.3.1 Phonetic Features

The phoneme gives us only a very gross way to model contextual effects. Many of the phonetic processes like assimilation and deletion are best modeled by more fine-grained articulatory facts about the neighboring context. Fig. 7.10 showed that /t/ and /d/ were deleted before [h], [dh], and [w]; rather than list all the possible following phones which could influence deletion, we'd like to generalize that /t/ often deletes "before consonants". Similarly, flapping can be viewed as a kind of voicing assimilation, in which unvoiced /t/ becomes a voiced tap [dx] in between voiced vowels or glides. Rather than list every possible vowel or glide, we'd like to say that flapping happens 'near vowels or voiced segments'. Finally, vowels that precede nasal sounds [n], [m], and [ng], often acquire some of the nasal quality of the following vowel. In each of these cases, a phone is influenced by the articulation of the neighboring phones (nasal, consonantal, voiced). The reason these changes happen is that the movement of the speech articulators (tongue, lips, velum) during speech production is continuous and is subject to physical constraints like momentum. Thus an articulator may start moving during one phone to get into place in time for the next phone. When the realization of a phone is influenced by the articulatory movement of neighboring phones, we say it is influenced by **coarticulation**. **Coarticulation** is the movement of articulators to anticipate the next sound, or perseverating movement from the last sound.

COARTICULATION

We can capture generalizations about the different phones that cause coarticulation by using **distinctive features**. Features are (generally) binary variables which express some generalizations about groups of phonemes. For example the feature [voice] is true of the voiced sounds (vowels, [n], [v], [b], etc); we say they are [+voice] while unvoiced sounds are [-voice]. These articulatory features can draw on the articulatory ideas of **place** and **manner** that we described earlier. Common **place** features include [+labial] ([p, b, m]), [+coronal] ([ch d dh jh l n r s sh t th z zh]), and [+dorsal]. Manner features include [+consonantal] (or alternatively [+vocalic]), [+continuant], [+sonorant]. For vowels, features include [+high], [+low], [+back], [+round] and so on. Distinctive features are used to represent each phoneme as a matrix of feature values. Many different sets of distinctive features exist; probably any of these are perfectly adequate for most computational purposes. Fig. 7.11 shows the values for some phones from one partial set of features.

DISTINCTIVE FEATURES

	syl	son	cons	strident	nasal	high	back	round	tense	voice	labial	coronal	dorsal
b	-	-	+	-	-	-	-	+	+	+	+	-	-
p	-	-	+	-	-	-	-	-	+	-	+	-	-
iy	+	+	-	-	-	+	-	-	-	+	-	-	-

Figure 7.11 Some partial feature matrices for phones, values simplified from Chomsky and Halle (1968). *Syl* is short for syllabic; *son* for sonorant, and *cons* for consonantal.

One main use of these distinctive features is in capturing natural articulatory classes of phones. In both synthesis and recognition, as we will see, we often need to build models of how a phone behaves in a certain context. But we rarely have enough data to model the interaction of every possible left and right context phone on the behavior of a phone. For this reason we can use the relevant feature ([voice], [nasal], etc) as a useful model of the context; the feature functions as a kind of backoff model of the phone. Another use in speech recognition is to build articulatory feature detectors and use them to help in the task of phone detection; for example Kirchhoff et al. (2002) built neural-net detectors for the following set of multi-valued articulatory features and used them to improve the detection of phones in German speech recognition:

Feature	Values	Feature	Value
voicing	+voice, -voice, silence	**manner**	stop, vowel, lateral, nasal, fricative, silence
cplace	labial, coronal, palatal, velar	**vplace**	glottal, high, mid, low, silence
front-back	front, back, nil, silence	**rounding**	+round, -round, nil, silence

7.3.2 Predicting Phonetic Variation

For speech synthesis as well as recognition, we need to be able to represent the relation between the abstract category and its surface appearance, and predict the surface appearance from the abstract category and the context of the utterance. In early work in phonology, the relationship between a phoneme and its allophones was captured by writing a **phonological rule**. Here is the phonological rule for flapping in the traditional notation of Chomsky and Halle (1968):

$$\left/ \begin{Bmatrix} t \\ d \end{Bmatrix} \right/ \rightarrow \text{[dx]} \; / \; \acute{\text{V}} \;\underline{\quad}\; \text{V} \tag{7.1}$$

In this notation, the surface allophone appears to the right of the arrow, and the phonetic environment is indicated by the symbols surrounding the underbar (___). Simple rules like these are used in both speech recognition and synthesis when we want to generate many pronunciations for a word; in speech recognition this is often used as a first step toward picking the most likely single pronunciation for a word (see Sec. **??**).

In general, however, there are two reasons why these simple 'Chomsky-Halle'-type rules don't do well at telling us **when** a given surface variant is likely to be used. First, variation is a stochastic process; flapping sometimes occurs, and sometimes doesn't, even in the same environment. Second, many factors that are not related to the phonetic environment are important to this prediction task. Thus linguistic research and speech recognition/synthesis both rely on statistical tools to predict the surface form of a word by showing which factors cause, e.g., a particular /t/ to flap in a particular context.

7.3.3 Factors Influencing Phonetic Variation

RATE OF SPEECH

One important factor that influences phonetic variation is the **rate of speech**, generally measured in syllables per second. Rate of speech varies both across and within speakers. Many kinds of phonetic reduction processes are much more common in fast speech, including flapping, is vowel reduction, and final /t/ and /d/ deletion (Wolfram, 1969). Measuring syllables per second (or words per second) can be done with a transcription (by counting the number of words or syllables in the transcription of a region and dividing by the number of seconds), or by using signal-processing metrics (Morgan and Fosler-Lussier, 1989). Another factor affecting variation is word frequency or predictability. Final /t/ and /d/ deletion is particularly likely to happen in words which are very frequent like *and* and *just* (Labov, 1975; Neu, 1980). Deletion is also more likely when the two words surrounding the segment are a collocation (Bybee, 2000; Gregory et al., 1999; Zwicky, 1972). The phone [t] is more likely to be palatalized in frequent words and phrases. Words with higher conditional probability given the previous word are more likely to have reduced vowels, deleted consonants, and flapping (Bell et al., 2003; Gregory et al., 1999).

Other phonetic, phonological, and morphological factors affect variation as well. For example /t/ is much more likely to flap than /d/; and there are complicated interactions with syllable, foot, and word boundaries (Rhodes, 1992). As we will discuss in Ch. 8, speech is broken up into units called **intonation phrases** or **breath groups**. Words at the beginning or end of intonation phrases are longer and less likely to be reduced. As for morphology, it turns out that deletion is less likely if the word-final /t/ or /d/ is the English past tense ending (Guy, 1980). For example in Switchboard, deletion is more likely in the word *around* (73% /d/-deletion) than in the word *turned* (30% /d/-deletion) even though the two words have similar frequencies.

Variation is also affected by the speaker's state of mind. For example the word *the* can be pronounced with a full vowel [dh iy] or reduced vowel [dh ax]. It is more likely to be pronounced with the full vowel [iy] when the speaker is disfluent and having "planning problems"; in general speakers are more likely to use a full vowel than a reduced one if they don't know what they are going to say next (Fox Tree and Clark, 1997; Bell et al., 2003; Keating et al., 1994).

SOCIOLINGUISTIC
DIALECT

Sociolinguistic factors like gender, class, and **dialect** also affect pronunciation variation. North American English is often divided into eight dialect regions (Northern, Southern, New England, New York/Mid-Atlantic, North Midlands, South Midlands, Western, Canadian). Southern dialect speakers use a monophthong or near-monophthong [aa] or [ae] instead of a diphthong in some words with the vowel [ay]. In these dialects *rice* is pronounced [r aa s]. **African-American Vernacular English** (AAVE) shares many vowels with Southern American English, and also has individual words with specific pronunciations such as [b ih d n ih s] for *business* and [ae k s] for *ask*. For older speakers or those not from the American West or Midwest, the words *caught* and *cot* have different vowels ([k ao t] and [k aa t] respectively). Young American speakers or those from the West pronounce the two words *cot* and *caught* the same; the vowels [ao] and [aa] are usually not distinguished in these dialects except before [r]. For speakers of most non-American and some American dialects of English (for example Australian English), the words *Mary* ([m ey r iy]), *marry* ([m ae r iy]) and *merry* ([m eh r iy]) are all pronounced differently. Most American speakers pronounce all three of these words identically as ([m eh r iy]).

AFRICAN-AMERICAN VERNACULAR ENGLISH

REGISTER
STYLE

Other sociolinguistic differences are due to **register** or **style**; a speaker might pronounce the same word differently depending on who they were talking to or what the social situation is. One of the most well-studied examples of style-variation is the suffix *-ing* (as in *something*), which can be pronounced [ih ng] or [ih n] (this is often written *somethin'*). Most speakers use both forms; as Labov (1966) shows, they use [ih ng] when they are being more formal, and [ih n] when more casual. Wald and Shopen (1981) found that men are more likely to use the non-standard form [ih n] than women, that both men and women are more likely to use more of the standard form [ih ng] when the addressee is a women, and that men (but not women) tend to switch to [ih n] when they are talking with friends.

Many of these results on predicting variation rely on logistic regression on phonetically-transcribed corpora, a technique with a long history in the analysis of phonetic variation (Cedergren and Sankoff, 1974), particularly using the VARBRUL and GOLDVARB software (Rand and Sankoff, 1990).

Finally, the detailed acoustic realization of a particular phone is very strongly influenced by **coarticulation** with its neighboring phones. We will return to these fine-grained phonetic details in the following chapters (Sec. **??** and Sec. **??**) after we introduce acoustic phonetics.

7.4 Acoustic Phonetics and Signals

We will begin with a brief introduction to the acoustic waveform and how it is digitized, summarize the idea of frequency analysis and spectra. This will be an extremely brief overview; the interested reader is encouraged to consult the references at the end of the chapter.

7.4.1 Waves

Acoustic analysis is based on the sine and cosine functions. Fig. 7.12 shows a plot of a sine wave, in particular the function:

$$y = A * sin(2\pi f t) \tag{7.2}$$

where we have set the amplitude A to 1 and the frequency f to 10 cycles per second.

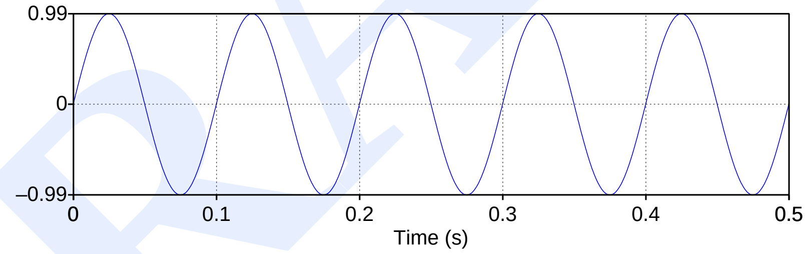


Figure 7.12 A sine wave with a frequency of 10 Hz and an amplitude of 1.

FREQUENCY
AMPLITUDE
CYCLES PER SECOND
HERTZ

Recall from basic mathematics that two important characteristics of a wave are its **frequency** and **amplitude**. The frequency is the number of times a second that a wave repeats itself, i.e. the number of **cycles**. We usually measure frequency in **cycles per second**. The signal in Fig. 7.12 repeats itself 5 times in .5 seconds, hence 10 cycles per second. Cycles per second are usually called **Hertz** (shortened to **Hz**), so the frequency in Fig. 7.12 would be described as 10 Hz. The **amplitude** A of a sine wave is the maximum value on the Y axis.

PERIOD

The **period** T of the wave is defined as the time it takes for one cycle to complete, defined as

$$T = \frac{1}{f} \tag{7.3}$$

In Fig. 7.12 we can see that each cycle lasts a tenth of a second, hence $T = .1$ seconds.

7.4.2 Speech Sound Waves

Let's turn from hypothetical waves to sound waves. The input to a speech recognizer, like the input to the human ear, is a complex series of changes in air pressure. These changes in air pressure obviously originate with the speaker, and are caused by the specific way that air passes through the glottis and out the oral or nasal cavities. We represent sound waves by plotting the change in air pressure over time. One metaphor which sometimes helps in understanding these graphs is to imagine a vertical plate which is blocking the air pressure waves (perhaps in a microphone in front of a speaker's mouth, or the eardrum in a hearer's ear). The graph measures the amount of **compression** or **rarefaction** (uncompression) of the air molecules at this plate. Fig. 7.13 shows a short segment of a waveform taken from the Switchboard corpus of telephone speech of the vowel [iy] from someone saying "she just had a baby".

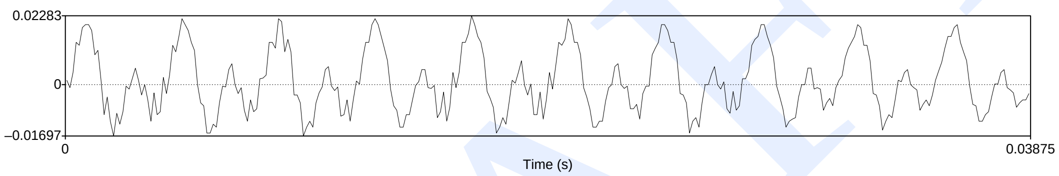


Figure 7.13 A waveform of the vowel [iy] from an utterance to be shown in Fig. 7.17. The *y*-axis shows the level of air pressure above and below normal atmospheric pressure. The *x*-axis shows time. Notice that the wave repeats regularly.

Let's explore how the digital representation of the sound wave shown in Fig. 7.13 would be constructed. The first step in processing speech is to convert the analog representations (first air pressure, and then analog electric signals in a microphone), into a digital signal. This process of **analog-to-digital conversion** has two steps: **sampling** and **quantization**. A signal is sampled by measuring its amplitude at a particular time; the **sampling rate** is the number of samples taken per second. In order to accurately measure a wave, it is necessary to have at least two samples in each cycle: one measuring the positive part of the wave and one measuring the negative part. More than two samples per cycle increases the amplitude accuracy, but less than two samples will cause the frequency of the wave to be completely missed. Thus the maximum frequency wave that can be measured is one whose frequency is half the sample rate (since every cycle needs two samples). This maximum frequency for a given sampling rate is called the **Nyquist frequency**. Most information in human speech is in frequencies below 10,000 Hz; thus a 20,000 Hz sampling rate would be necessary for complete accuracy. But telephone speech is filtered by the switching network, and only frequencies less than 4,000 Hz are transmitted by telephones. Thus an 8,000 Hz sampling rate is sufficient for **telephone-bandwidth** speech like the Switchboard corpus. A 16,000 Hz sampling rate (sometimes called **wideband**) is often used for microphone speech.

SAMPLING

SAMPLING RATE

NYQUIST FREQUENCY

TELEPHONE-BANDWIDTH

WIDEBAND

Even an 8,000 Hz sampling rate requires 8000 amplitude measurements for each second of speech, and so it is important to store the amplitude measurement efficiently. They are usually stored as integers, either 8-bit (values from -128–127) or 16 bit (values

QUANTIZATION

from -32768–32767). This process of representing real-valued numbers as integers is called **quantization** because there is a minimum granularity (the quantum size) and all values which are closer together than this quantum size are represented identically.

CHANNELS

PCM

Once data is quantized, it is stored in various formats. One parameter of these formats is the sample rate and sample size discussed above; telephone speech is often sampled at 8 kHz and stored as 8-bit samples, while microphone data is often sampled at 16 kHz and stored as 16-bit samples. Another parameter of these formats is the number of **channels**. For stereo data, or for two-party conversations, we can store both channels in the same file, or we can store them in separate files. A final parameter is whether each sample is stored linearly or whether it is compressed. One common compression format used for telephone speech is μ-law (often written u-law but still pronounced mu-law). The intuition of log compression algorithms like μ-law is that human hearing is more sensitive at small intensities than large ones; the log represents small values with more faithfulness at the expense of more error on large values. The linear (unlogged) values are generally referred to as **linear PCM** values (PCM stands for Pulse Code Modulation, but never mind that). Here's the equation for compressing a linear PCM sample value x to 8-bit μ-law, (where μ=255 for 8 bits):

$$F(x) = \frac{sgn(s)\log(1+\mu|s|)}{\log(1+\mu)} \tag{7.4}$$

There are a number of standard file formats for storing the resulting digitized wavefile, such as Microsoft's WAV, Apple AIFF and Sun AU, all of which have special headers; simple headerless 'raw' files are also used. For example the .wav format is a subset of Microsoft's RIFF format for multimedia files; RIFF is a general format that can represent a series of nested chunks of data and control information. Fig. 7.14 shows a simple .wav file with a single data chunk together with its format chunk:

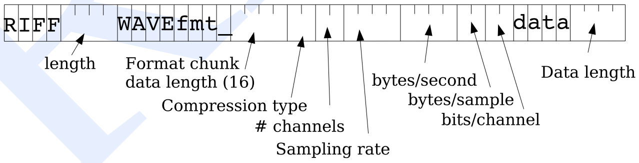


Figure 7.14 Microsoft wavefile header format, assuming simple file with one chunk. Following this 44-byte header would be the data chunk.

7.4.3 Frequency and Amplitude; Pitch and Loudness

Sound waves, like all waves, can be described in terms of frequency, amplitude and the other characteristics that we introduced earlier for pure sine waves. In sound waves these are not quite as simple to measure as they were for sine waves. Let's consider frequency. Note in Fig. 7.13 that although not exactly a sine, that the wave is nonetheless periodic, and that there are 10 repetitions of the wave in the 38.75 milliseconds (.03875 seconds) we have captured in the figure. Thus the frequency of this segment of the wave is 10/.03875 or 258 Hz.

Where does this periodic 258Hz wave come from? It comes from the speed of vibration of the vocal folds; since the waveform in Fig. 7.13 is from the vowel [iy], it is voiced. Recall that voicing is caused by regular openings and closing of the vocal folds. When the vocal folds are open, air is pushing up through the lungs, creating a region of high pressure. When the folds are closed, there is no pressure from the longs. Thus when the vocal folds are vibrating, we expect to see regular peaks in amplitude of the kind we see in Fig. 7.13, each major peak corresponding to an opening of the vocal folds. The frequency of the vocal fold vibration, or the frequency of the complex wave, is called the **fundamental frequency** of the waveform, often abbreviated $\mathbf{F_0}$. We can plot F_0 over time in a **pitch track**. Fig. 7.15 shows the pitch track of a short question, "Three o'clock?" represented below the waveform. Note the rise in F_0 at the end of the question.

FUNDAMENTAL FREQUENCY

F_0

PITCH TRACK

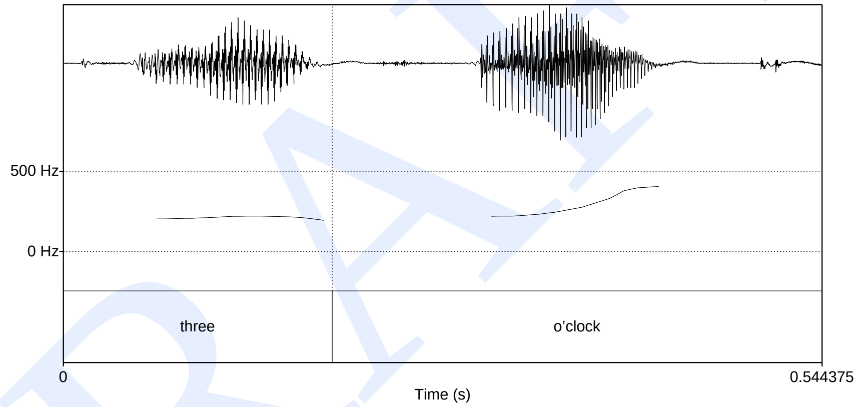


Figure 7.15 Pitch track of the question "Three o'clock?", shown below the wavefile. Note the rise in F0 at the end of the question. Note the lack of pitch trace during the very quiet part (the "o'" of "o'clock"; automatic pitch tracking is based on counting the pulses in the voiced regions, and doesn't work if there is no voicing (or insufficient sound at all).

The vertical axis in Fig. 7.13 measures the amount of air pressure variation; pressure is force per unit area, measured in Pascals (Pa). A high value on the vertical axis (a high amplitude) indicates that there is more air pressure at that point in time, a zero value means there is normal (atmospheric) air pressure, while a negative value means there is lower than normal air pressure (rarefaction).

In addition to this value of the amplitude at any point in time, we also often need to know the average amplitude over some time range, to give us some idea of how great the average displacement of air pressure is. But we can't just take the average of the amplitude values over a range; the positive and negative values would (mostly) cancel out, leaving us with a number close to zero. Instead, we generally use the RMS (root-mean-square) amplitude, which squares each number before averaging (making it positive), and then takes the square root at the end.

$$\text{RMS amplitude}_{i=1}^{N} = \sqrt{\sum_{i=1}^{N} \frac{x_i^2}{N}} \tag{7.5}$$

POWER

The **power** of the signal is related to the square of the amplitude. If the number of samples of a sound is N, the power is

$$\text{Power} = \frac{1}{N}\sum_{i=1}^{n} x[i]^2 \tag{7.6}$$

INTENSITY

Rather than power, we more often refer to the **intensity** of the sound, which normalizes the power to the human auditory threshold, and is measured in dB. If P_0 is the auditory threshold pressure = $2 \times 10^{-5} Pa$ then intensity is defined as follows:

$$\text{Intensity} = 10\log_{10} \frac{1}{NP_0}\sum_{i=1}^{n} x_i^2 \tag{7.7}$$

Fig. 7.16 shows an intensity plot for the sentence "Is it a long movie?" from the CallHome corpus, again shown below the waveform plot.

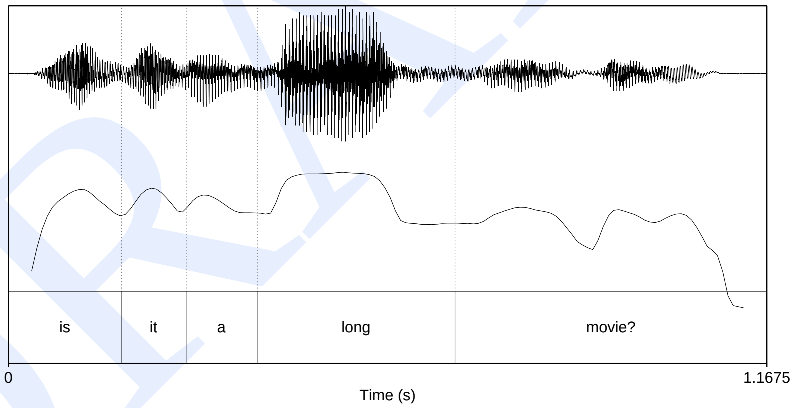


Figure 7.16 Intensity plot for the sentence "Is it a long movie?". Note the intensity peaks at each vowel, and the especially high peak for the word *long*.

PITCH

Two important perceptual properties, **pitch** and **loudness**, are related to frequency and intensity. The **pitch** of a sound is the mental sensation or perceptual correlate of fundamental frequency; in general if a sound has a higher fundamental frequency we perceive it as having a higher pitch. We say "in general" because the relationship is not linear, since human hearing has different acuities for different frequencies. Roughly speaking, human pitch perception is most accurate between 100Hz and 1000Hz, and in this range pitch correlates linearly with frequency. Human hearing represents frequencies above 1000 Hz less accurately and above this range pitch correlates logarithmically with frequency. Logarithmic representation means that the differences between high frequencies are compressed, and hence not as accurately perceived. There are various psychoacoustic models of pitch perception scales. One common model is the **mel** scale (Stevens et al., 1937; Stevens and Volkmann, 1940). A mel is a unit of pitch defined so that pairs of sounds which are perceptually equidistant in pitch are separated by an equal number of mels. The mel frequency m can be computed from the raw acoustic frequency as follows:

MEL

$$m = 1127\ln(1+\frac{f}{700}) \tag{7.8}$$

We will return to the mel scale in Ch. 9 when we introduce the MFCC representation of speech used in speech recognition.

The **loudness** of a sound is the perceptual correlate of the **power**. So sounds with higher amplitudes are perceived as louder, but again the relationship is not linear. First of all, as we mentioned above when we defined μ-law compression, humans have greater resolution in the low power range; the ear is more sensitive to small power differences. Second, it turns out that there is a complex relationship between power, frequency, and perceived loudness; sounds in certain frequency ranges are perceived as being louder than those in other frequency ranges.

Various algorithms exist for automatically extracting F_0. In a slight abuse of terminology these are called **pitch extraction** algorithms. The autocorrelation method of pitch extraction, for example, correlates the signal with itself, at various offsets. The offset that gives the highest correlation gives the period of the signal. Other methods for pitch extraction are based on the cepstral features we will return to in Ch. 9. There are various publicly available pitch extraction toolkits; for example an augmented autocorrelation pitch tracker is provided with `Praat` (Boersma and Weenink, 2005).

PITCH EXTRACTION

7.4.4 Interpreting Phones from a Waveform

Much can be learned from a visual inspection of a waveform. For example, vowels are pretty easy to spot. Recall that vowels are voiced; another property of vowels is that they tend to be long, and are relatively loud (as we can see in the intensity plot in Fig. 7.16). Length in time manifests itself directly on the x-axis, while loudness is related to (the square of) amplitude on the y-axis. We saw in the previous section that voicing is realized by regular peaks in amplitude of the kind we saw in Fig. 7.13, each major peak corresponding to an opening of the vocal folds. Fig. 7.17 shows the waveform of the short phrase ‘she just had a baby’. We have labeled this waveform with word and phone labels. Notice that each of the six vowels in Fig. 7.17, [iy], [ax], [ae], [ax], [ey], [iy], all have regular amplitude peaks indicating voicing.

For a stop consonant, which consists of a closure followed by a release, we can often see a period of silence or near silence followed by a slight burst of amplitude. We can see this for both of the [b]’s in *baby* in Fig. 7.17.

Another phone that is often quite recognizable in a waveform is a fricative. Recall that fricatives, especially very strident fricatives like [sh], are made when a narrow channel for airflow causes noisy, turbulent air. The resulting hissy sounds have a very noisy, irregular waveform. This can be seen somewhat in Fig. 7.17; it’s even clearer in Fig. 7.18, where we’ve magnified just the first word *she*.

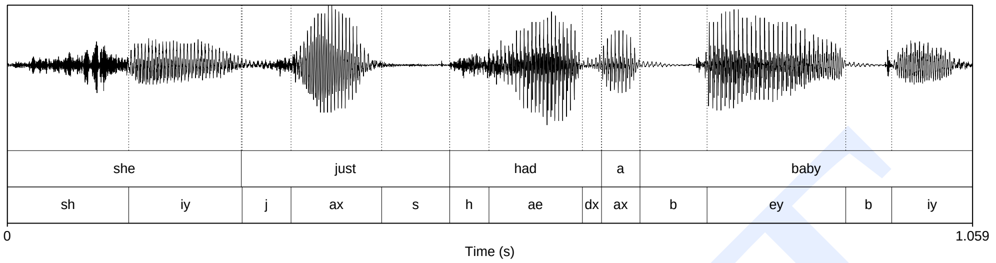


Figure 7.17 A waveform of the sentence "She just had a baby" from the Switchboard corpus (conversation 4325). The speaker is female, was 20 years old in 1991, which is approximately when the recording was made, and speaks the South Midlands dialect of American English.

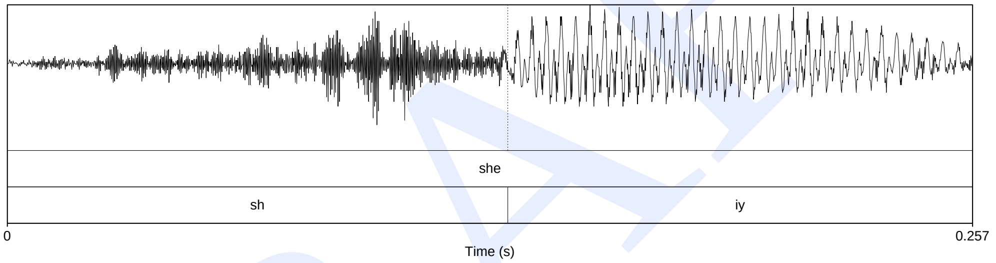


Figure 7.18 A more detailed view of the first word "she" extracted from the wavefile in Fig. 7.17. Notice the difference between the random noise of the fricative [sh] and the regular voicing of the vowel [iy].

7.4.5 Spectra and the Frequency Domain

While some broad phonetic features (such as energy, pitch, and the presence of voicing, stop closures, or fricatives) can be interpreted directly from the waveform, most computational applications such as speech recognition (as well as human auditory processing) are based on a different representation of the sound in terms of its component frequencies. The insight of **Fourier analysis** is that every complex wave can be represented as a sum of many sine waves of different frequencies. Consider the waveform in Fig. 7.19. This waveform was created (in `Praat`) by summing two sine waveforms, one of frequency 10 Hz and one of frequency 100 Hz.

SPECTRUM

We can represent these two component frequencies with a **spectrum**. The spectrum of a signal is a representation of each of its frequency components and their amplitudes. Fig. 7.20 shows the spectrum of Fig. 7.19. Frequency in Hz is on the x-axis and amplitude on the y-axis. Note that there are two spikes in the figure, one at 10 Hz and one at 100 Hz. Thus the spectrum is an alternative representation of the original waveform, and we use the spectrum as a tool to study the component frequencies of a soundwave at a particular time point.

Let's look now at the frequency components of a speech waveform. Fig. 7.21 shows

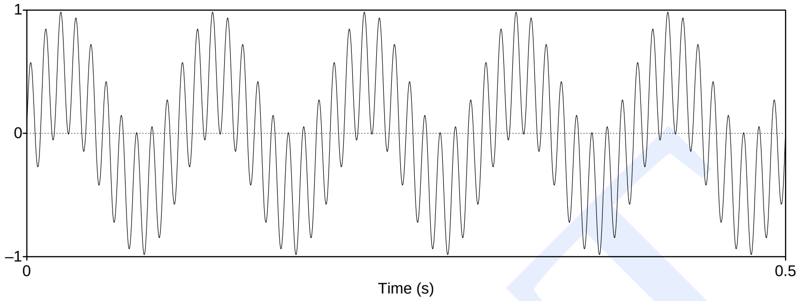


Figure 7.19 A waveform created by summing two sine waveforms, one of frequency 10 Hz (note the 5 repetitions in the half-second window) and one of frequency 100 Hz, both with amplitude 1.

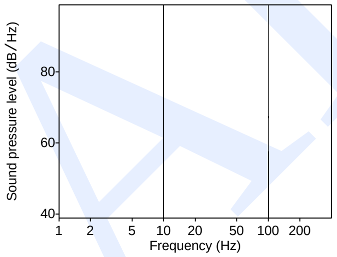


Figure 7.20 The spectrum of the waveform in Fig. 7.19.

part of the waveform for the vowel [ae] of the word *had*, cut out from the sentence shown in Fig. 7.17.

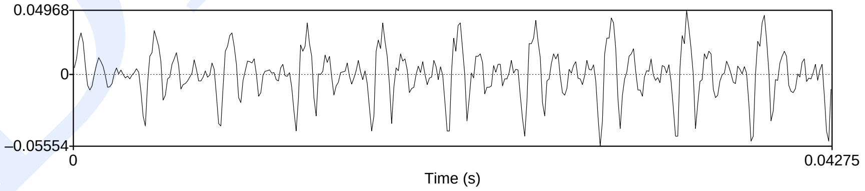


Figure 7.21 The waveform of part of the vowel [ae] from the word *had* cut out from the waveform shown in Fig. 7.17.

Note that there is a complex wave which repeats about ten times in the figure; but there is also a smaller repeated wave which repeats four times for every larger pattern (notice the four small peaks inside each repeated wave). The complex wave has a frequency of about 234 Hz (we can figure this out since it repeats roughly 10 times in .0427 seconds, and 10 cycles/.0427 seconds = 234 Hz).

The smaller wave then should have a frequency of roughly four times the frequency of the larger wave, or roughly 936 Hz. Then if you look carefully you can see two little waves on the peak of many of the 936 Hz waves. The frequency of this tiniest wave must be roughly twice that of the 936 Hz wave, hence 1872 Hz.

Fig. 7.22 shows a smoothed spectrum for the waveform in Fig. 7.21, computed via a Discrete Fourier Transform (DFT).

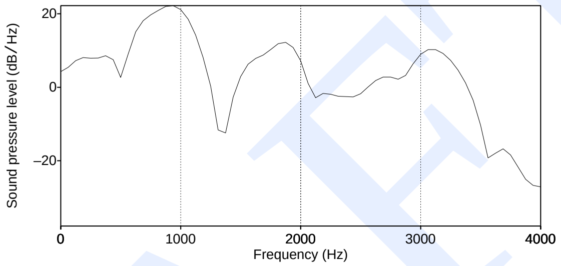


Figure 7.22 A spectrum for the vowel [ae] from the word *had* in the waveform of *She just had a baby* in Fig. 7.17.

The *x*-axis of a spectrum shows frequency while the *y*-axis shows some measure of the magnitude of each frequency component (in decibels (dB), a logarithmic measure of amplitude that we saw earlier). Thus Fig. 7.22 shows that there are significant frequency components at around 930 Hz, 1860 Hz, and 3020 Hz, along with many other lower-magnitude frequency components. These first two components are just what we noticed in the time domain by looking at the wave in Fig. 7.21!

COCHLEA

INNER EAR

Why is a spectrum useful? It turns out that these spectral peaks that are easily visible in a spectrum are very characteristic of different phones; phones have characteristic spectral "signatures". Just as chemical elements give off different wavelengths of light when they burn, allowing us to detect elements in stars looking at the spectrum of the light, we can detect the characteristic signature of the different phones by looking at the spectrum of a waveform. This use of spectral information is essential to both human and machine speech recognition. In human audition, the function of the **cochlea** or **inner ear** is to compute a spectrum of the incoming waveform. Similarly, the various kinds of acoustic features used in speech recognition as the HMM observation are all different representations of spectral information.

SPECTROGRAM

Let's look at the spectrum of different vowels. Since some vowels change over time, we'll use a different kind of plot called a **spectrogram**. While a spectrum shows the frequency components of a wave at one point in time, a **spectrogram** is a way of envisioning how the different frequencies that make up a waveform change over time. The *x*-axis shows time, as it did for the waveform, but the *y*-axis now shows frequencies in Hertz. The darkness of a point on a spectrogram corresponding to the amplitude of the frequency component. Very dark points have high amplitude, light points have low amplitude. Thus the spectrogram is a useful way of visualizing the three dimensions (time x frequency x amplitude).

Fig. 7.23 shows spectrograms of 3 American English vowels, [ih], [ae], and [ah]. Note that each vowel has a set of dark bars at various frequency bands, slightly different bands for each vowel. Each of these represents the same kind of spectral peak that we saw in Fig. 7.21.

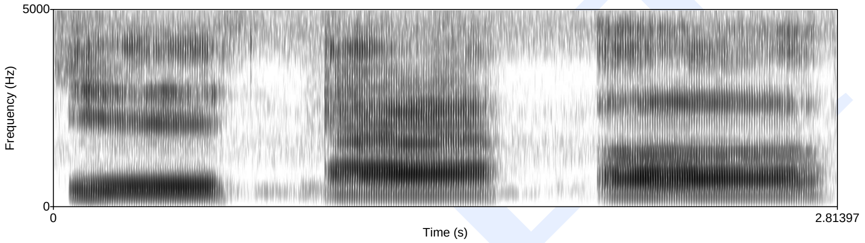


Figure 7.23 Spectrograms for three American English vowels, [ih], [ae], and [uh], spoken by the first author.

FORMANT

Each dark bar (or spectral peak) is called a **formant**. As we will discuss below, a formant is a frequency band that is particularly amplified by the vocal tract. Since different vowels are produced with the vocal tract in different positions, they will produce different kinds of amplifications or resonances. Let's look at the first two formants, called F1 and F2. Note that F1, the dark bar closest to the bottom is in different position for the 3 vowels; it's low for [ih] (centered at about 470Hz) and somewhat higher for [ae] and [ah] (somewhere around 800Hz) By contrast F2, the second dark bar from the bottom, is highest for [ih], in the middle for [ae], and lowest for [ah].

We can see the same formants in running speech, although the reduction and coarticulation processes make them somewhat harder to see. Fig. 7.24 shows the spectrogram of 'she just had a baby' whose waveform was shown in Fig. 7.17. F1 and F2 (and also F3) are pretty clear for the [ax] of *just*, the [ae] of *had*, and the [ey] of *baby*.

What specific clues can spectral representations give for phone identification? First, since different vowels have their formants at characteristic places, the spectrum can be used to distinguish vowels from each other. We've seen that [ae] in the sample waveform had formants at 930 Hz, 1860 Hz, and 3020 Hz. Consider the vowel [iy], at the beginning of the utterance in Fig. 7.17. The spectrum for this vowel is shown in Fig. 7.25. The first formant of [iy] is 540 Hz; much lower than the first formant for [ae], while the second formant (2581 Hz) is much higher than the second formant for [ae]. If you look carefully you can see these formants as dark bars in Fig. 7.24 just around 0.5 seconds.

The location of the first two formants (called F1 and F2) plays a large role in determining vowel identity, although the formants still differ from speaker to speaker. Higher formants tend to be caused more by general characteristic of the speakers vocal tract rather than by individual vowels. Formants also can be used to identify the nasal

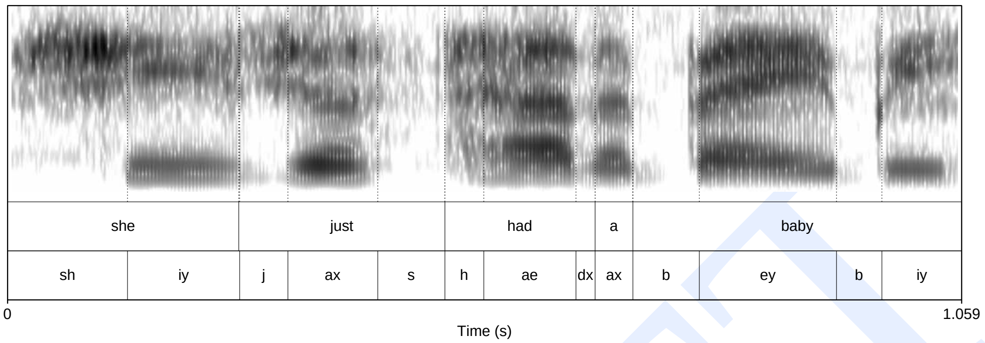


Figure 7.24 A spectrogram of the sentence “She just had a baby” whose waveform was shown in Fig. 7.17. We can think of a spectrogram is as a collection of spectra (time-slices) like Fig. 7.22 placed end to end. Note

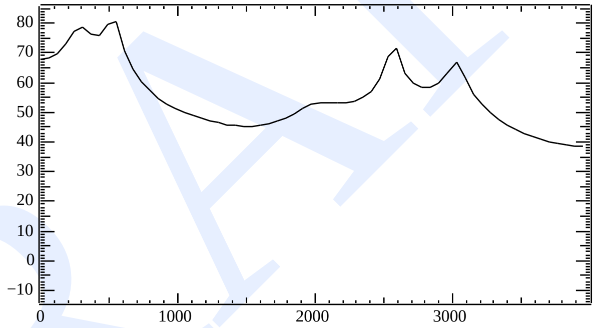


Figure 7.25 A smoothed (LPC) spectrum for the vowel [iy] at the start of *She just had a baby*. Note that the first formant (540 Hz) is much lower than the first formant for [ae] shown in Fig. 7.22, while the second formant (2581 Hz) is much higher than the second formant for [ae].

phones [n], [m], and [ng], and the liquids [l] and [r].

7.4.6 The Source-Filter Model

SOURCE-FILTER

Why do different vowels have different spectral signatures? As we briefly mentioned above, the formants are caused by the resonant cavities of the mouth. The **source-filter** model is a way of explaining the acoustics of a sound by modeling how the pulses produced by the glottis (the **source**) are shaped by the vocal tract (the **filter**).

HARMONICS

Let’s see how this works. Whenever we have a wave such as the vibration in air caused by the glottal pulse, the wave also has **harmonics**. A harmonic is another wave whose frequency is a multiple of the fundamental wave. Thus for example a 115 Hz glottal fold vibration leads to harmonics (other waves) of 230 Hz, 345 Hz, 460 Hz, and so on on. In general each of these waves will be weaker, i.e. have much less amplitude than the wave at the fundamental frequency.

It turns out, however, that the vocal tract acts as a kind of filter or amplifier; indeed any cavity such as a tube causes waves of certain frequencies to be amplified, and others to be damped. This amplification process is caused by the shape of the cavity; a given shape will cause sounds of a certain frequency to resonate and hence be amplified. Thus by changing the shape of the cavity we can cause different frequencies to be amplified.

Now when we produce particular vowels, we are essentially changing the shape of the vocal tract cavity by placing the tongue and the other articulators in particular positions. The result is that different vowels cause different harmonics to be amplified.

So a wave of the same fundamental frequency passed through different vocal tract positions will result in different harmonics being amplified.

We can see the result of this amplification by looking at the relationship between the shape of the oral tract and the corresponding spectrum. Fig. 7.26 shows the vocal tract position for three vowels and a typical resulting spectrum. The formants are places in the spectrum where the vocal tract happens to amplify particular harmonic frequencies.

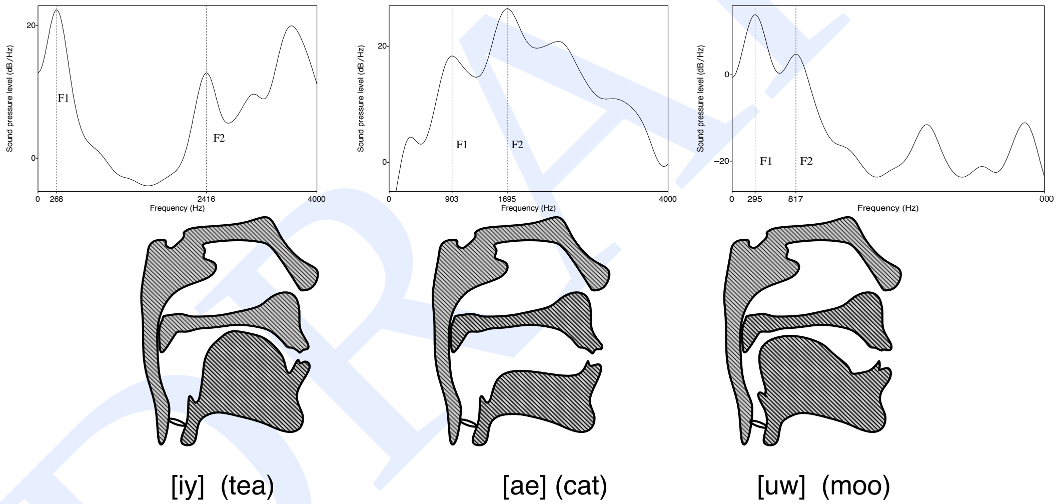


Figure 7.26 Visualizing the vocal tract position as a filter: the tongue positions for three English vowels and the resulting smoothed spectra showing F1 and F2. Tongue positions modeled after Ladefoged (1996).

7.5 PHONETIC RESOURCES

PRONUNCIATION DICTIONARIES

A wide variety of phonetic resources can be drawn on for computational work. One key set of resources are **pronunciation dictionaries**. Such on-line phonetic dictionaries give phonetic transcriptions for each word. Three commonly used on-line dictionaries for English are the CELEX, CMUdict, and PRONLEX lexicons; for other languages, the LDC has released pronunciation dictionaries for Egyptian Arabic, German, Japanese, Korean, Mandarin, and Spanish. All these dictionaries can be used for both speech recognition and synthesis work.

The CELEX dictionary (Baayen et al., 1995) is the most richly annotated of the dictionaries. It includes all the words in the Oxford Advanced Learner's Dictionary (1974) (41,000 lemmata) and the Longman Dictionary of Contemporary English (1978) (53,000 lemmata), in total it has pronunciations for 160,595 wordforms. Its (British rather than American) pronunciations are transcribed using an ASCII version of the IPA called SAM. In addition to basic phonetic information like phone strings, syllabification, and stress level for each syllable, each word is also annotated with morphological, part of speech, syntactic, and frequency information. CELEX (as well as CMU and PRONLEX) represent three levels of stress: primary stress, secondary stress, and no stress. For example, some of the CELEX information for the word *dictionary* includes multiple pronunciations ('dIk-S@n-rI and 'dIk-S@-n@-rI, corresponding to ARPABET [d ih k sh ax n r ih] and [d ih k sh ax n ax r ih] respectively), together with the CV-skelata for each one ([CVC][CVC][CV] and [CVC][CV][CV][CV]), the frequency of the word, the fact that it is a noun, and its morphological structure (diction+ary).

The free CMU Pronouncing Dictionary (CMU, 1993) has pronunciations for about 125,000 wordforms. It uses an 39-phone ARPAbet-derived phoneme set. Transcriptions are phonemic, and thus instead of marking any kind of surface reduction like flapping or reduced vowels, it marks each vowel with the number 0 (unstressed) 1 (stressed), or 2 (secondary stress). Thus the word *tiger* is listed as [T AY1 G ER0] the word *table* as [T EY1 B AH0 L], and the word *dictionary* as [D IH1 K SH AH0 N EH2 R IY0]. The dictionary is not syllabified, although the nucleus is implicitly marked by the (numbered) vowel.

The PRONLEX dictionary (LDC, 1995) was designed for speech recognition and contains pronunciations for 90,694 wordforms. It covers all the words used in many years of the Wall Street Journal, as well as the Switchboard Corpus. PRONLEX has the advantage that it includes many proper names (20,000, where CELEX only has about 1000). Names are important for practical applications, and they are both frequent and difficult; we return to a discussion of deriving name pronunciations in Ch. 8.

Another useful resource is a **phonetically annotated corpus**, in which a collection of waveforms is hand-labeled with the corresponding string of phones. Two important phonetic corpora in English are the TIMIT corpus and the Switchboard Transcription Project corpus.

TIME-ALIGNED TRANSCRIPTION

The TIMIT corpus (NIST, 1990) was collected as a joint project between Texas Instruments (TI), MIT, and SRI. It is a corpus of 6300 read sentences, where 10 sentences each from 630 speakers. The 6300 sentences were drawn from a set of 2342 predesigned sentences, some selected to have particular dialect shibboleths, others to maximize phonetic diphone coverage. Each sentence in the corpus was phonetically hand-labeled, the sequence of phones was automatically aligned with the sentence wavefile, and then the automatic phone boundaries were manually hand-corrected (Seneff and Zue, 1988). The result is a **time-aligned transcription**; a transcription in which each phone in the transcript is associated with a start and end time in the waveform; we showed a graphical example of a time-aligned transcription in Fig. 7.17.

The phoneset for TIMIT, and for the Switchboard Transcription Project corpus below, is a more detailed one than the minimal phonemic version of the ARPAbet. In particular, these phonetic transcriptions make use of the various reduced and rare phones mentioned in Fig. 7.1 and Fig. 7.2; the flap [dx], glottal stop [q], reduced vowels [ax], [ix], [axr], voiced allophone of [h] ([hv]), and separate phones for stop closure ([dcl], [tcl], etc) and release ([d], [t], etc). An example transcription is shown in Fig. 7.27.

she	had	your	dark	suit	in	greasy	wash	water	all	year
sh iy	hv ae dcl	jh axr	dcl d aa r kcl	s ux q	en	gcl g r iy s ix	w aa sh	q w aa dx axr q	aa l	y ix axr

Figure 7.27 Phonetic transcription from the TIMIT corpus. Note palatalization of [d] in *had*, unreleased final stop in *dark*, glottalization of final [t] in *suit* to [q], and flap of [t] in *water*. The TIMIT corpus also includes time-alignments for each phone (not shown).

Where TIMIT is based on read speech, the more recent Switchboard Transcription Project corpus is based on the Switchboard corpus of conversational speech. This phonetically-annotated portion consists of approximately 3.5 hours of sentences extracted from various conversations (Greenberg et al., 1996). As with TIMIT, each annotated utterance contains a time-aligned transcription. The Switchboard transcripts are time-aligned at the syllable level rather than at the phone level; thus a transcript consists of a sequence of syllables with the start and end time of each syllables in the corresponding wavefile. Fig. 7.28 shows an example from the Switchboard Transcription Project, for the phrase *they're kind of in between right now*:

0.470	0.640	0.720	0.900	0.953	1.279	1.410	1.630
dh er	k aa	n ax	v ih m	b ix	t w iy n	r ay	n aw

Figure 7.28 Phonetic transcription of the Switchboard phrase *they're kind of in between right now*. Note vowel reduction in *they're* and *of*, coda deletion in *kind* and *right*, and resyllabification (the [v] of *of* attaches as the onset of *in*). Time is given in number of seconds from beginning of sentence to start of each syllable.

Phonetically transcribed corpora are also available for other languages; the Kiel corpus of German is commonly used, as are various Mandarin corpora transcribed by the Chinese Academy of Social Sciences (Li et al., 2000).

In addition to resources like dictionaries and corpora, there are many useful phonetic software tools. One of the most versatile is the free Praat package (Boersma and Weenink, 2005), which includes spectrum and spectrogram analysis, pitch extraction and formant analysis, and an embedded scripting language for automation. It is available on Microsoft, Macintosh, and UNIX environments.

7.6 ADVANCED: ARTICULATORY AND GESTURAL PHONOLOGY

We saw in Sec. 7.3.1 that we could use **distinctive features** to capture generalizations across phone class. These generalizations were mainly articulatory (although some, like [strident] and the vowel height features, are primarily acoustic).

ARTICULATORY PHONOLOGY
ARTICULATORY GESTURE

This idea that articulation underlies phonetic production is used in a more sophisticated way in **articulatory phonology**, in which the **articulatory gesture** is the underlying phonological abstraction (Browman and Goldstein, 1986, 1992). Articulatory gestures are defined as parameterized **dynamical systems**. Since speech production requires the coordinated actions of tongue, lips, glottis, etc, articulatory phonology represents a speech utterances as a sequence of potentially overlapping articulatory gestures. Fig. 7.29 shows the sequent of gestures (or **gestural score**) required for the production of the word *pawn* [p aa n]. The lips first close, then the glottis opens, then the tongue body moves back toward the pharynx wall for the vowel [aa], the velum drops for the nasal sounds, and finally the tongue tip closes against the alveolar ridge. The lines in the diagram indicate gestures which are phased with respect to each other. With such a gestural representation, the nasality in the [aa] vowel is explained by the timing of the gestures; the velum drops before the tongue tip has quite closed.

GESTURAL SCORE

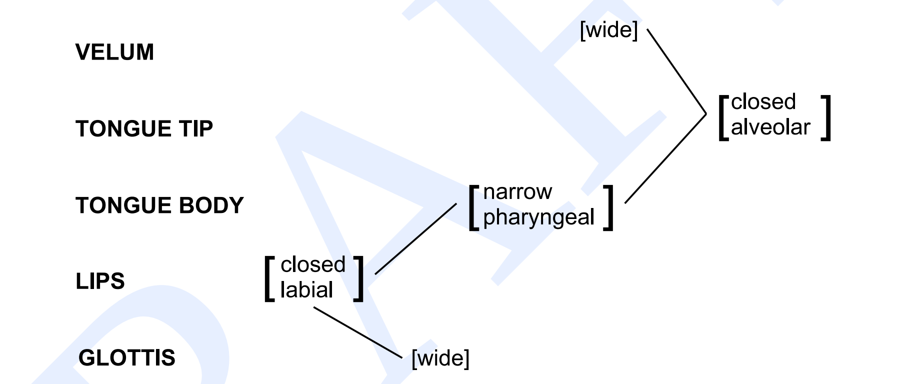


Figure 7.29 The gestural score for the word *pawn* as pronounced [p aa n], after Browman and Goldstein (1989) and Browman and Goldstein (1995).

The intuition behind articulatory phonology is that the gestural score is likely to be much better as a set of hidden states at capturing the continuous nature of speech than a discrete sequence of phones. In addition, using articulatory gestures as a basic unit can help in modeling the fine-grained effects of coarticulation of neighboring gestures that we will explore further when we introduce **diphones** (Sec. **??**) and **triphones** (Sec. **??**).

Computational implementations of articulatory phonology have recently appeared in speech recognition, using articulatory gestures rather than phones as the underlying representation or hidden variable. Since multiple articulators (tongue, lips, etc) can move simultaneously, using gestures as the hidden variable implies a multi-tier hidden representation. Fig. 7.30 shows the articulatory feature set used in the work of Livescu and Glass (2004) and Livescu (2005); Fig. 7.31 shows examples of how phones are mapped onto this feature set.

Feature	Description	value = meaning
LIP-LOC	position of lips	LAB = labial (neutral position); PRO = protruded (rounded); DEN = dental
LIP-OPEN	degree of opening of lips	CL = closed; CR = critical (labial/labio-dental fricative); NA = narrow (e.g., [w], [uw]); WI = wide (all other sounds)
TT-LOC	location of tongue tip	DEN = inter-dental ([th], [dh]); ALV = alveolar ([t], [n]); P-A = palato-alveolar ([sh]); RET = retroflex ([r])
TT-OPEN	degree of opening of tongue tip	CL = closed (stop); CR = critical (fricative); NA = narrow ([r], alveolar glide); M-N = medium-narrow;MID = medium;WI = wide
TB-LOC	location of tongue body	PAL = palatal (e.g. [sh], [y]); VEL = velar (e.g., [k], [ng]); UVU = uvular (neutral position); PHA = pharyngeal (e.g. [aa])
TB-OPEN	degree of opening of tongue body	CL = closed (stop); CR = critical (e.g. fricated [g] in ”legal”); NA = narrow (e.g. [y]); M-N = medium-narrow; MID = medium; WI = wide
VEL	state of the velum	CL = closed (non-nasal); OP = open (nasal)
GLOT	state of the glottis	CL = closed (glottal stop); CR = critical (voiced); OP = open (voiceless)

Figure 7.30 Articulatory-phonology-based feature set from Livescu (2005).

phone	LIP-LOC	LIP-OPEN	TT-LOC	TT-OPEN	TB-LOC	TB-OPEN	VEL	GLOT
aa	LAB	W	ALV	W	PHA	M-N	CL(.9),OP(.1)	CR
ae	LAB	W	ALV	W	VEL	W	CL(.9),OP(.1)	CR
b	LAB	CR	ALV	M	UVU	W	CL	CR
f	DEN	CR	ALV	M	VEL	M	CL	OP
n	LAB	W	ALV	CL	UVU	M	OP	CR
s	LAB	W	ALV	CR	UVU	M	CL	OP
uw	PRO	N	P-A	W	VEL	N	CL(.9),OP(.1)	CR

Figure 7.31 Livescu (2005): sample of mapping from phones to underyling target articulatory feature values. Note that some values are probabilistic.

7.7 Summary

This chapter has introduced many of the important concepts of phonetics and computational phonetics.

- We can represent the pronunciation of words in terms of units called **phones**. The standard system for representing phones is the **International Phonetic Alphabet** or **IPA**. The most common computational system for transcription of English is the **ARPAbet**, which conveniently uses ASCII symbols.
- Phones can be described by how they are produced **articulatorily** by the vocal organs; consonants are defined in terms of their **place** and **manner** of articulation and **voicing**, vowels by their **height**, **backness**, and **roundness**.
- A **phoneme** is a generalization or abstraction over different phonetic realizations. **Allophonic rules** express how a phoneme is realized in a given context.
- Speech sounds can also be described **acoustically**. Sound waves can be described in terms of **frequency**, **amplitude**, or their perceptual correlates, **pitch** and **loudness**.
- The **spectrum** of a sound describes its different frequency components. While some phonetic properties are recognizable from the waveform, both humans and machines rely on spectral analysis for phone detection.

- A **spectrogram** is a plot of a spectrum over time. Vowels are described by characteristic harmonics called **formants**.
- **Pronunciation dictionaries** are widely available, and used for both speech recognition and speech synthesis, including the CMU dictionary for English and CELEX dictionaries for English, German, and Dutch. Other dictionaries are available from the LDC.
- Phonetically transcribed corpora are a useful resource for building computational models of phone variation and reduction in natural speech.

Bibliographical and Historical Notes

The major insights of articulatory phonetics date to the linguists of 800–150 B.C. India. They invented the concepts of place and manner of articulation, worked out the glottal mechanism of voicing, and understood the concept of assimilation. European science did not catch up with the Indian phoneticians until over 2000 years later, in the late 19th century. The Greeks did have some rudimentary phonetic knowledge; by the time of Plato's *Theaetetus* and *Cratylus*, for example, they distinguished vowels from consonants, and stop consonants from continuants. The Stoics developed the idea of the syllable and were aware of phonotactic constraints on possible words. An unknown Icelandic scholar of the twelfth century exploited the concept of the phoneme, proposed a phonemic writing system for Icelandic, including diacritics for length and nasality. But his text remained unpublished until 1818 and even then was largely unknown outside Scandinavia (Robins, 1967). The modern era of phonetics is usually said to have begun with Sweet, who proposed what is essentially the phoneme in his *Handbook of Phonetics* (1877). He also devised an alphabet for transcription and distinguished between *broad* and *narrow* transcription, proposing many ideas that were eventually incorporated into the IPA. Sweet was considered the best practicing phonetician of his time; he made the first scientific recordings of languages for phonetic purposes, and advanced the state of the art of articulatory description. He was also infamously difficult to get along with, a trait that is well captured in Henry Higgins, the stage character that George Bernard Shaw modeled after him. The phoneme was first named by the Polish scholar Baudouin de Courtenay, who published his theories in 1894.

Students with further interest in transcription and articulatory phonetics should consult an introductory phonetics textbook such as Ladefoged (1993) or Clark and Yallop (1995). Pullum and Ladusaw (1996) is a comprehensive guide to each of the symbols and diacritics of the IPA. A good resource for details about reduction and other phonetic processes in spoken English is Shockey (2003). Wells (1982) is the definitive 3-volume source on dialects of English.

Many of the classic insights in acoustic phonetics had been developed by the late 1950's or early 1960's; just a few highlights include techniques like the sound spectrograph (Koenig et al., 1946), theoretical insights like the working out of the source-filter theory and other issues in the mapping between articulation and acoustics (Fant, 1970; Stevens et al., 1953; Stevens and House, 1955; Heinz and Stevens, 1961; Stevens and

House, 1961), the F1xF2 space of vowel formants Peterson and Barney (1952), the understanding of the phonetic nature of stress and and the use of duration and intensity as cues (Fry, 1955), and a basic understanding of issues in phone perception Miller and Nicely (1955), Liberman et al. (1952). Lehiste (1967) is a collection of classic papers on acoustic phonetics.

Excellent textbooks on acoustic phonetics include Johnson (2003) and (Ladefoged, 1996). (Coleman, 2005) includes an introduction to computational processing of acoustics as well as other speech processing issues, from a linguistic perspective. (Stevens, 1998) lays out an influential theory of speech sound production. There are a wide variety of books that address speech from a signal processing and electrical engineering perspective. The ones with the greatest coverage of computational phonetics issues include (Huang et al., 2001), (O'Shaughnessy, 2000), and (Gold and Morgan, 1999). An excellent textbook on digital signal processing is Lyons (2004).

There are a number of software packages for acoustic phonetic analysis. Probably the most widely-used one is Praat (Boersma and Weenink, 2005).

Many phonetics papers of computational interest are to be found in the *Journal of the Acoustical Society of America (JASA)*, *Computer Speech and Language*, and *Speech Communication*.

EXERCISES

7.1 Find the mistakes in the ARPAbet transcriptions of the following words:

a. "three" [dh r i]
b. "sing" [s ih n g]
c. "eyes" [ay s]
d. "study" [s t uh d i]
e. "though" [th ow]
f. "planning" [p pl aa n ih ng]
g. "slight" [s l iy t]

7.2 Translate the pronunciations of the following color words from the IPA into the ARPAbet (and make a note if you think you pronounce them differently than this!):

a. [rɛd]
b. [blu]
c. [grin]
d. [ˈjɛloʊ]
e. [blæk]
f. [waɪt]
g. [ˈɔrɪndʒ]
h. [ˈpɝpl̩]
i. [pjus]
j. [toʊp]

7.3 Ira Gershwin's lyric for *Let's Call the Whole Thing Off* talks about two pronunciations of the word "either" (in addition to the tomato and potato example given at the beginning of the chapter. Transcribe Ira Gershwin's two pronunciations of "either" in the ARPAbet.

7.4 Transcribe the following words in the ARPAbet:

a. dark
b. suit

c. greasy
d. wash
e. water

7.5 Take a wavefile of your choice. Some examples are on the textbook website. Download the PRAAT software, and use it to transcribe the wavefiles at the word level, and into ARPAbet phones, using Praat to help you play pieces of each wavfile, and to look at the wavefile and the spectrogram.

7.6 Record yourself saying five of the English vowels: [aa], [eh], [ae], [iy], [uw]. Find F1 and F2 for each of your vowels.

Baayen, R. H., Piepenbrock, R., and Gulikers, L. (1995). *The CELEX Lexical Database (Release 2) [CD-ROM]*. Linguistic Data Consortium, University of Pennsylvania [Distributor], Philadelphia, PA.

Bell, A., Jurafsky, D., Fosler-Lussier, E., Girand, C., Gregory, M. L., and Gildea, D. (2003). Effects of disfluencies, predictability, and utterance position on word form variation in English conversation. *Journal of the Acoustical Society of America*, *113*(2), 1001–1024.

Boersma, P. and Weenink, D. (2005). Praat: doing phonetics by computer (version 4.3.14). [Computer program]. Retrieved May 26, 2005, from http://www.praat.org/.

Bolinger, D. (1981). Two kinds of vowels, two kinds of rhythm. Indiana University Linguistics Club.

Browman, C. P. and Goldstein, L. (1986). Towards an articulatory phonology. *Phonology Yearbook*, *3*, 219–252.

Browman, C. P. and Goldstein, L. (1992). Articulatory phonology: An overview. *Phonetica*, *49*, 155–180.

Browman, C. P. and Goldstein, L. (1989). Articulatory gestures as phonological units. *Phonology*, *6*, 201–250.

Browman, C. P. and Goldstein, L. (1995). Dynamics and articulatory phonology. In Port, R. and v. Gelder, T. (Eds.), *Mind as Motion: Explorations in the Dynamics of Cognition*, pp. 175–193. MIT Press.

Bybee, J. L. (2000). The phonology of the lexicon: evidence from lexical diffusion. In Barlow, M. and Kemmer, S. (Eds.), *Usage-based Models of Language*, pp. 65–85. CSLI, Stanford.

Cedergren, H. J. and Sankoff, D. (1974). Variable rules: performance as a statistical reflection of competence. *Language*, *50*(2), 333–355.

Chomsky, N. and Halle, M. (1968). *The Sound Pattern of English*. Harper and Row.

Clark, J. and Yallop, C. (1995). *An Introduction to Phonetics and Phonology*. Blackwell, Oxford. 2nd ed.

CMU (1993). The Carnegie Mellon Pronouncing Dictionary v0.1. Carnegie Mellon University.

Coleman, J. (2005). *Introducing Speech and Language Processing*. Cambridge University Press.

Fant, G. M. (1960). *Acoustic Theory of Speech Production*. Mouton.

Fox Tree, J. E. and Clark, H. H. (1997). Pronouncing "the" as "thee" to signal problems in speaking. *Cognition*, *62*, 151–167.

Fry, D. B. (1955). Duration and intensity as physical correlates of linguistic stress. *Journal of the Acoustical Society of America*, *27*, 765–768.

Godfrey, J., Holliman, E., and McDaniel, J. (1992). SWITCHBOARD: Telephone speech corpus for research and development. In *IEEE ICASSP-92*, San Francisco, pp. 517–520. IEEE.

Gold, B. and Morgan, N. (1999). *Speech and Audio Signal Processing*. Wiley Press.

Greenberg, S., Ellis, D., and Hollenback, J. (1996). Insights into spoken language gleaned from phonetic transcription of the Switchboard corpus. In *ICSLP-96*, Philadelphia, PA, pp. S24–27.

Gregory, M. L., Raymond, W. D., Bell, A., Fosler-Lussier, E., and Jurafsky, D. (1999). The effects of collocational strength and contextual predictability in lexical production. In *CLS-99*, pp. 151–166. University of Chicago, Chicago.

Guy, G. R. (1980). Variation in the group and the individual: The case of final stop deletion. In Labov, W. (Ed.), *Locating Language in Time and Space*, pp. 1–36. Academic.

Heinz, J. M. and Stevens, K. N. (1961). On the properties of voiceless fricative consonants. *Journal of the Acoustical Society of America*, *33*, 589–596.

Huang, X., Acero, A., and Hon, H.-W. (2001). *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall, Upper Saddle River, NJ.

Johnson, K. (2003). *Acoustic and Auditory Phonetics*. Blackwell, Oxford. 2nd ed.

Keating, P. A., Byrd, D., Flemming, E., and Todaka, Y. (1994). Phonetic analysis of word and segment variation using the TIMIT corpus of American English. *Speech Communication*, *14*, 131–142.

Kirchhoff, K., Fink, G. A., and Sagerer, G. (2002). Combining acoustic and articulatory feature information for robust speech recognition. *Speech Communication*, *37*, 303–319.

Koenig, W., Dunn, H. K., Y., L., and Lacy (1946). The sound spectrograph. *Journal of the Acoustical Society of America*, *18*, 19–49.

Labov, W. (1966). *The Social Stratification of English in New York City*. Center for Applied Linguistics, Washington, D.C.

Labov, W. (1972). The internal evolution of linguistic rules. In Stockwell, R. P. and Macaulay, R. K. S. (Eds.), *Linguistic Change and Generative Theory*, pp. 101–171. Indiana University Press, Bloomington.

Labov, W. (1975). *The quantitative study of linguistic structure*. Pennsylvania Working Papers on Linguistic Change and Variation v.1 no. 3. U.S. Regional Survey, Philadelphia, PA.

Labov, W. (1994). *Principles of Linguistic Change: Internal Factors*. Blackwell, Oxford.

Ladefoged, P. (1993). *A Course in Phonetics*. Harcourt Brace Jovanovich. Third Edition.

Ladefoged, P. (1996). *Elements of Acoustic Phonetics*. University of Chicago, Chicago, IL. Second Edition.

LDC (1995). COMLEX English Pronunciation Dictionary Version 0.2 (COMLEX 0.2). Linguistic Data Consortium.

Lehiste, I. (Ed.). (1967). *Readings in Acoustic Phonetics*. MIT Press.

Li, A., Zheng, F., Byrne, W., Fung, P., Kamm, T., Yi, L., Song, Z., Ruhi, U., Venkataramani, V., and Chen, X. (2000). CASS: A phonetically transcribed corpus of mandarin spontaneous speech. In *ICSLP-00*, Beijing, China, pp. 485–488.

Liberman, A. M., Delattre, P. C., and Cooper, F. S. (1952). The role of selected stimulus variables in the perception of the unvoiced stop consonants. *American Journal of Psychology*, *65*, 497–516.

Livescu, K. (2005). *Feature-Based Pronuncaition Modeling for Automatic Speech Recognition*. Ph.D. thesis, Massachusetts Institute of Technology.

Livescu, K. and Glass, J. (2004). Feature-based pronunciation modeling with trainable asynchrony probabilities. In *ICSLP-04*, Jeju, South Korea.

Lyons, R. G. (2004). *Understanding Digital Signal Processing*. Prentice Hall, Upper Saddle River, NJ. 2nd. ed.

Miller, C. A. (1998). Pronunciation modeling in speech synthesis. Tech. rep. IRCS 98–09, University of Pennsylvania Institute for Research in Cognitive Science, Philadephia, PA.

Miller, G. A. and Nicely, P. E. (1955). An analysis of perceptual confusions among some English consonants. *Journal of the Acoustical Society of America*, *27*, 338–352.

Morgan, N. and Fosler-Lussier, E. (1989). Combining multiple estimators of speaking rate. In *IEEE ICASSP-89*.

Neu, H. (1980). Ranking of constraints on /t,d/ deletion in American English: A statistical analysis. In Labov, W. (Ed.), *Locating Language in Time and Space*, pp. 37–54. Academic Press.

NIST (1990). TIMIT Acoustic-Phonetic Continuous Speech Corpus. National Institute of Standards and Technology Speech Disc 1-1.1. NIST Order No. PB91-505065.

O'Shaughnessy, D. (2000). *Speech Communications: Human and Machine*. IEEE Press, New York. 2nd. ed.

Peterson, G. E. and Barney, H. L. (1952). Control methods used in a study of the vowels. *Journal of the Acoustical Society of America*, *24*, 175–184.

Pullum, G. K. and Ladusaw, W. A. (1996). *Phonetic Symbol Guide*. University of Chicago, Chicago, IL. Second Edition.

Rand, D. and Sankoff, D. (1990). Goldvarb: A variable rule application for the macintosh. Available at http://www.crm.umontreal.ca/ sankoff/GoldVarb_Eng.html.

Rhodes, R. A. (1992). Flapping in American English. In Dressler, W. U., Prinzhorn, M., and Rennison, J. (Eds.), *Proceedings of the 7th International Phonology Meeting*, pp. 217–232. Rosenberg and Sellier, Turin.

Robins, R. H. (1967). *A Short History of Linguistics*. Indiana University Press, Bloomington.

Seneff, S. and Zue, V. W. (1988). Transcription and alignment of the TIMIT database. In *Proceedings of the Second Symposium on Advanced Man-Machine Interface through Spoken Language*, Oahu, Hawaii.

Shockey, L. (2003). *Sound Patterns of Spoken English*. Blackwell, Oxford.

Shoup, J. E. (1980). Phonological aspects of speech recognition. In Lea, W. A. (Ed.), *Trends in Speech Recognition*, pp. 125–138. Prentice-Hall.

Stevens, K. N., Kasowski, S., and Fant, G. M. (1953). An electrical analog of the vocal tract. *Journal of the Acoustical Society of America*, *25*(4), 734–742.

Stevens, K. N. (1998). *Acoustic Phonetics*. MIT Press.

Stevens, K. N. and House, A. S. (1955). Development of a quantitative description of vowel articulation. *Journal of the Acoustical Society of America*, *27*, 484–493.

Stevens, K. N. and House, A. S. (1961). An acoustical theory of vowel production and some of its implications. *Journal of Speech and Hearing Research*, *4*, 303–320.

Stevens, S. S. and Volkmann, J. (1940). The relation of pitch to frequency: A revised scale. *The American Journal of Psychology*, *53*(3), 329–353.

Stevens, S. S., Volkmann, J., and Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *Journal of the Acoustical Society of America*, *8*, 185–190.

Sweet, H. (1877). *A Handbook of Phonetics*. Clarendon Press, Oxford.

Wald, B. and Shopen, T. (1981). A researcher's guide to the sociolinguistic variable (ING). In Shopen, T. and Williams, J. M. (Eds.), *Style and Variables in English*, pp. 219–249. Winthrop Publishers.

Wells, J. C. (1982). *Accents of English*. Cambridge University Press.

Wolfram, W. A. (1969). *A Sociolinguistic Description of Detroit Negro Speech*. Center for Applied Linguistics, Washington, D.C.

Zappa, F. and Zappa, M. U. (1982). Valley girl. From Frank Zappa album *Ship Arriving Too Late To Save A Drowning Witch*.

Zwicky, A. (1972). On Casual Speech. In *CLS-72*, pp. 607–615. University of Chicago.

8 SPEECH SYNTHESIS

And computers are getting smarter all the time: Scientists tell us that soon they will be able to talk to us. (By 'they' I mean 'computers': I doubt scientists will ever be able to talk to us.)

Dave Barry

In Vienna in 1769, Wolfgang von Kempelen built for the Empress Maria Theresa the famous Mechanical Turk, a chess-playing automaton consisting of a wooden box filled with gears, and a robot mannequin sitting behind the box who played chess by moving pieces with his mechanical arm. The Turk toured Europe and the Americas for decades, defeating Napolean Bonaparte and even playing Charles Babbage. The Mechanical Turk might have been one of the early successes of artificial intelligence if it were not for the fact that it was, alas, a hoax, powered by a human chessplayer hidden inside the box.

What is perhaps less well-known is that von Kempelen, an extraordinarily prolific inventor, also built between 1769 and 1790 what is definitely not a hoax: the first full-sentence speech synthesizer. His device consisted of a bellows to simulate the lungs, a rubber mouthpiece and a nose aperature, a reed to simulate the vocal folds, various whistles for each of the fricatives. and a small auxiliary bellows to provide the puff of air for plosives. By moving levers with both hands, opening and closing various openings, and adjusting the flexible leather 'vocal tract', different consonants and vowels could be produced.

More than two centuries later, we no longer build our speech synthesizers out of wood, leather, and rubber, nor do we need trained human operators. The modern task of **speech synthesis**, also called **text-to-speech** or **TTS**, is to produce speech (acoustic waveforms) from text input.

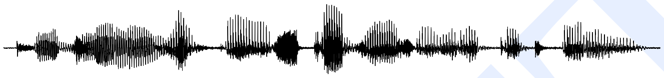
SPEECH SYNTHESIS
TEXT-TO-SPEECH
TTS

Modern speech synthesis has a wide variety of applications. Synthesizers are used, together with speech recognizers, in telephone-based conversational agents that conduct dialogues with people (see Ch. 23). Synthesizer are also important in non-conversational applications that speak **to** people, such as in devices that read out loud for the blind, or in video games or children's toys. Finally, speech synthesis can be used to speak **for** sufferers of neurological disorders, such as astrophysicist Steven Hawking who, having lost the use of his voice due to ALS, speaks by typing to a speech synthesizer and having the synthesizer speak out the words. State of the art systems in speech synthesis can achieve remarkably natural speech for a very wide variety of input situations, although even the best systems still tend to sound wooden and are limited in the voices they use.

The task of speech synthesis is to map a text like the following:

(8.1) `PG&E will file schedules on April 20.`

to a waveform like the following:



TEXT ANALYSIS

WAVEFORM SYNTHESIS

Speech synthesis systems perform this mapping in two steps, first converting the input text into a **phonemic internal representation** and then converting this internal representation into a waveform. We will call the first step **text analysis** and the second step **waveform synthesis** (although other names are also used for these steps).

A sample of the internal representation for this sentence is shown in Fig. 8.1. Note that the acronym `PG&E` is expanded into the words P G AND E, the number `20` is expanded into *twentieth*, a phone sequence is given for each of the words, and there is also prosodic and phrasing information (the *'s) which we will define later.

			*			*			* L-L%
P	G	AND	E	WILL	FILE	SCHEDULES	ON	APRIL	TWENTIETH
p iy	jh iy	ae n d	iy	w ih l	f ay l	s k eh jh ax l z	aa n	ey p r ih l	t w eh n t iy ax th

Figure 8.1 Intermediate output for a unit selection synthesizer for the sentence *PG&E will file schedules on April 20.*. The numbers and acronyms have been expanded, words have been converted into phones, and prosodic features have been assigned.

While text analysis algorithms are relatively standard, there are three widely different paradigms for waveform synthesis: **concatenative synthesis**, **formant synthesis**, and **articulatory synthesis**. The architecture of most modern commercial TTS systems is based on concatenative synthesis, in which samples of speech are chopped up, stored in a database, and combined and reconfigured to create new sentences. Thus we will focus on concatenative synthesis for most of this chapter, although we will briefly introduce formant and articulatory synthesis at the end of the chapter.

HOURGLASS METAPHOR

Fig. 8.2 shows the TTS architecture for concatenative unit selection synthesis, using the two-step **hourglass metaphor** of Taylor (2008). In the following sections, we'll examine each of the components in this architecture.

8.1 Text Normalization

NORMALIZED

In order to generate a phonemic internal representation, raw text first needs to be preprocessed or **normalized** in a variety of ways. We'll need to break the input text into

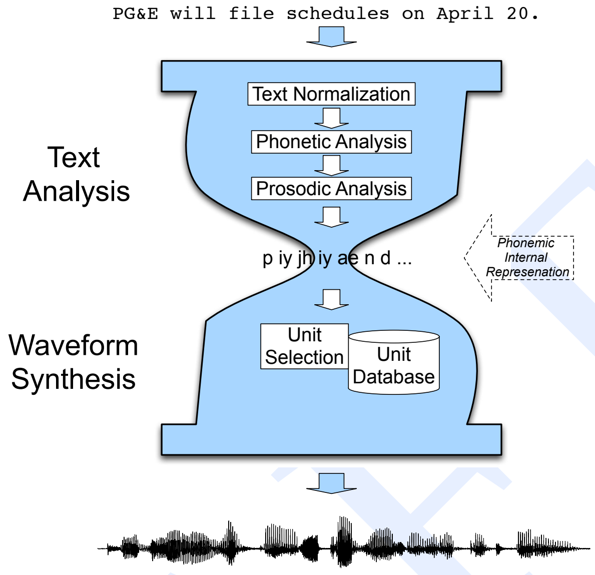


Figure 8.2 Architecture for the unit selection (concatenative) architecture for speech synthesis.

sentences, and deal with the idiosyncracies of abbreviations, numbers, and so on. Consider the difficulties in the following text drawn from the Enron corpus (Klimt and Yang, 2004):

> He said the increase in credit limits helped B.C. Hydro achieve record net income of about $1 billion during the year ending March 31. This figure does not include any write-downs that may occur if Powerex determines that any of its customer accounts are not collectible. Cousins, however, was insistent that all debts will be collected: “We continue to pursue monies owing and we expect to be paid for electricity we have sold.”

SENTENCE TOKENIZATION

The first task in text normalization is **sentence tokenization**. In order to segment this paragraph into separate utterances for synthesis, we need to know that the first sentence ends at the period after *March 31*, not at the period of *B.C.*. We also need to know that there is a sentence ending at the word *collected*, despite the punctuation being a colon rather than a period. The second normalization task is dealing with **non-standard words**. Non-standard words include number, acronyms, abbreviations, and so on. For example, *March 31* needs to be pronounced *March thirty-first*, not *March three one*; *$1 billion* needs to be pronounced *one billion dollars*, with the word *dollars* appearing after the word *billion*.

8.1.1 Sentence Tokenization

We saw two examples above where sentence tokenization is difficult because sentence boundaries are not always indicated by periods, and can sometimes be indicated by punctuation like colons. An additional problem occurs when an abbreviation ends a sentence, in which case the abbreviation-final period is playing a dual role:

(8.2) He said the increase in credit limits helped B.C. Hydro achieve record net income of about $1 billion during the year ending March 31.

(8.3) Cousins, however, was insistent that all debts will be collected: “We continue to pursue monies owing and we expect to be paid for electricity we have sold.”

(8.4) The group included Dr. J. M. Freeman and T. Boone Pickens Jr.

A key part of sentence tokenization is thus period disambiguation; we've seen a simple perl script for period disambiguation in Ch. 3. Most sentence tokenization algorithms are slightly more complex than this deterministic algorithm, and in particular are trained by machine learning methods rather than being hand-built. We do this by hand-labeling a training set with sentence boundaries, and then using any supervised machine learning method (decision trees, logistic regression, SVM, etc) to train a classifier to mark the sentence boundary decisions.

More specifically, we could start by tokenizing the input text into tokens separated by whitespace, and then select any token containing one of the three characters `!`, `.` or `?` (or possibly also `:`). After hand-labeling a corpus of such tokens, then we train a classifier to make a binary decision (EOS (end-of-sentence) versus not-EOS) on these potential sentence boundary characters inside these tokens.

The success of such a classifier depends on the features that are extracted for the classification. Let's consider some feature templates we might use to disambiguate these **candidate** sentence boundary characters, assuming we have a small amount of training data, labeled for sentence boundaries:

- the prefix (the portion of the candidate token preceding the candidate)
- the suffix (the portion of the candidate token following the candidate)
- whether the prefix or suffix is an abbreviation (from a list)
- the word preceding the candidate
- the word following the candidate
- whether the word preceding the candidate is an abbreviation
- whether the word following the candidate is an abbreviation

Consider the following example:

(8.5) ANLP Corp. chairman Dr. Smith resigned.

Given these feature templates, the feature values for the period `.` in the word `Corp.` in (8.5) would be:

```
PreviousWord = ANLP             NextWord = chairman
Prefix = Corp                   Suffix = NULL
PreviousWordAbbreviation = 1 NextWordAbbreviation = 0
```

If our training set is large enough, we can also look for lexical cues about sentence boundaries. For example, certain words may tend to occur sentence-initially, or sentence-finally. We can thus add the following features:

- Probability[candidate occurs at end of sentence]
- Probability[word following candidate occurs at beginning of sentence]

Finally, while most of the above features are relatively language-independent, we can use language-specific features. For example, in English, sentences usually begin with capital letters, suggesting features like the following:

- case of candidate: Upper, Lower, AllCap, Numbers
- case of word following candidate: Upper, Lower, AllCap, Numbers

Similary, we can have specific subclasses of abbreviations, such as honorifics or titles (e.g., Dr., Mr., Gen.), corporate designators (e.g., Corp., Inc.), or month-names (e.g., Jan., Feb.).

Any machine learning method can be applied to train EOS classifiers. Logistic regression and decision trees are two very common methods; logistic regression may have somewhat higher accuracy, although we have instead shown an example of a decision tree in Fig. 8.3 because it is easier for the reader to see how the features are used.

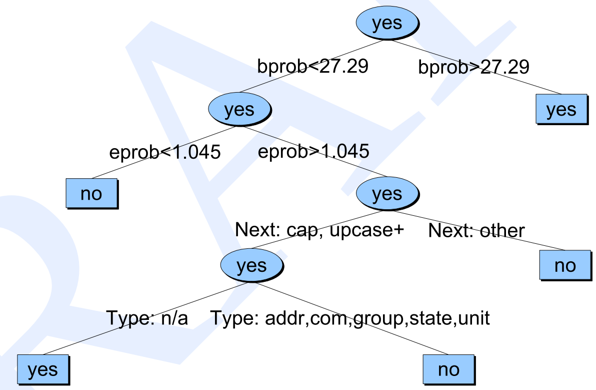


Figure 8.3 A decision tree for predicting whether a period '.' is an end of sentence (YES) or not an end-of-sentence (NO), using features like the log likelihood of the current word being the beginning of a sentence (`bprob`), the previous word being an end of sentence (`eprob`), the capitalization of the next word, and the abbreviation subclass (company, state, unit of measurement). After slides by Richard Sproat.

8.1.2 Non-Standard Words

NON-STANDARD WORDS

The second step in text normalization is normalizing **non-standard words**. Non-standard words are tokens like numbers or abbreviations, which need to be expanded into sequences of English words before they can be pronounced.

What is difficult about these non-standard words is that they are often very ambiguous. For example, the number *1750* can be spoken in at least three different ways, depending on the context:

`seventeen fifty:` (in *'The European economy in 1750'*)
`one seven five zero:` (in *'The password is 1750'*)
`seventeen hundred and fifty:` (in *'1750 dollars'*)
`one thousand, seven hundred, and fifty:` (in *'1750 dollars'*)

Similar ambiguities occur for Roman numerals like *IV*, (which can be pronounced `four`, `fourth`, or as the letters `I V` (meaning 'intravenous')), or *2/3*, which can be `two thirds` or `February third` or `two slash three`.

In addition to numbers, various non-standard words are composed of letters. Three types non-standard words include **abbreviations**, **letter sequences**, and **acronyms**. Abbreviations are generally pronounced by **expanding** them; thus *Jan 1* is pronounced `January first`, and *Wed* is pronounced `Wednesday`. **Letter sequences** like *UN*, *DVD*, *PC*, and *IBM* are pronounced by pronouncing each letter in a sequence (IBM is thus pronounced *ay b iy eh m*). **Acronyms** like *IKEA*, *MoMA*, *NASA*, and *UNICEF* are pronounced as if they were words; *MoMA* is pronounced *m ow m ax*. Ambiguity occurs here as well; should *Jan* be read as a word (the name `Jan`) or expanded as the month `January`?

These different types of numeric and alphabetic non-standard words can be summarized in Fig. 8.4. Each of the types has a particular realization (or realizations). For example, a year NYER is generally read in the **paired** method, in which each pair of digits is pronounced as an integer (e.g., `seventeen fifty` for 1750), while a U.S. zip code NZIP is generally read in the **serial** method, as a sequence of single digits (e.g., `nine four one one zero` for *94110*). The type BMONEY deals with the idiosyncracies of expressions like *$3.2 billion*, which must be read out with the word `dollars` at the end, as `three point two billion dollars`.

PAIRED

SERIAL

For the alphabetic NSWs, we have the class EXPN for abbreviations like *N.Y.* which are expanded, LSEQ for acronyms pronounced as letter sequences, and ASWD for acronyms pronounced as if they were words.

Dealing with non-standard words requires at least three steps: **tokenization** to separate out and identify potential non-standard words, **classification** to label them with a type from Fig. 8.4, and **expansion** to convert each type into a string of standard words.

In the tokenization step, we can tokenize the input by whitespace, and then assume that any word which is not in the pronunciation dictionary is a non-standard word. More sophisticated tokenization algorithms would also deal with the fact that some dictionaries already contain some abbreviations. The CMU dictionary, for example, contains abbreviated (and hence incorrect) pronunciations for *st*, *mr*, *mrs*, as well as day and month abbreviations like *mon*, *tues*, *nov*, *dec*, etc. Thus in addition to unseen words, we also need to label any of these acronyms and also single-character token as potential non-standard words. Tokenization algorithms also need to split words which are combinations of two tokens, like *2-car* or *RVing*. Words can be split by simple heuristics, such as splitting at dashes, or at changes from lower-case to upper-case.

The next step is assigning a NSW type; many types can be detected with simple regular expressions. For example, NYER could be detected by the following regular expression:

/(1[89][0-9][0-9])|(20[0-9][0-9]/

Other classes might be harder to write rules for, and so a more powerful option is

ALPHA	EXPN	abbreviation	*adv, N.Y., mph, gov't*
	LSEQ	letter sequence	*DVD, D.C., PC, UN, IBM,*
	ASWD	read as word	*IKEA*, unknown words/names
NUMBERS	NUM	number (cardinal)	*12, 45, 1/2, 0.6*
	NORD	number (ordinal)	*May 7, 3rd, Bill Gates III*
	NTEL	telephone (or part of)	*212-555-4523*
	NDIG	number as digits	*Room 101*
	NIDE	identifier	*747, 386, I5, pc110, 3A*
	NADDR	number as street address	*747, 386, I5, pc110, 3A*
	NZIP	zip code or PO Box	91020
	NTIME	a (compound) time	*3.20, 11:45*
	NDATE	a (compound) date	*2/28/05, 28/02/05*
	NYER	year(s)	*1998, 80s, 1900s, 2008*
	MONEY	money (US or other)	*$3.45, HK$300, Y20,200, $200K*
	BMONEY	money tr/m/billions	*$3.45 billion*
	PRCT	percentage	*75% 3.4%*

Figure 8.4 Some types of non-standard words in text normalization, selected from Table 1 of Sproat et al. (2001); not listed are types for URLs, emails, and some complex uses of punctuation.

to use a machine learning classifier with many features.

To distinguish between the alphabetic ASWD, LSEQ and EXPN classes, for example we might want features over the component letters. Thus short, all-capital words (*IBM*, *US*) might be LSEQ, longer all-lowercase words with a single-quote (*gov't*, *cap'n*) might be EXPN, and all-capital words with multiple vowels (*NASA*, *IKEA*) might be more likely to be ASWD.

Another very useful features is the identity of neighboring words. Consider ambiguous strings like *3/4*, which can be an NDATE march third or a *num* three-fourths. NDATE might be preceded by the word *on*, followed by the word *of*, or have the word *Monday* somewhere in the surrounding words. By contrast, NUM examples might be preceded by another number, or followed by words like *mile* and *inch*. Similarly, Roman numerals like *VII* tend to be NORD (*seven*) when preceded by *Chapter*, *part*, or *Act*, but NUM (*seventh*) when the words *king* or *Pope* occur in the neighborhood. These context words can be chosen as features by hand, or can be learned by machine learning techniques like the **decision list** algorithm of Ch. 8.

We can achieve the most power by building a single machine learning classifier which combines all of the above ideas. For example, the NSW classifier of (Sproat et al., 2001) uses 136 features, including letter-based features like *'all-upper-case'*, *'has-two-vowels'*, *'contains-slash'*, and *'token-length'*, as well as binary features for the presence of certain words like *Chapter*, *on*, or *king* in the surrounding context. Sproat et al. (2001) also included a rough-draft rule-based classifier, which used handwritten regular expression to classify many of the number NSWs. The output of this rough-draft classifier was used as just another feature in the main classifier.

In order to build such a main classifier, we need a hand-labeled training set, in which each token has been labeled with its NSW category; one such hand-labeled data-base was produced by Sproat et al. (2001). Given such a labeled training set, we can use any supervised machine learning algorithm to build the classifier.

Formally, we can model this task as the goal of producing the tag sequence T which is most probable given the observation sequence:

$$T^* = \operatorname*{argmax}_T P(T|O) \tag{8.6}$$

One way to estimate this probability is via decision trees. For example, for each observed token o_i, and for each possible NSW tag t_j, the decision tree produces the posterior probability $P(t_j|o_i)$. If we make the incorrect but simplifying assumption that each tagging decision is independent of its neighbors, we can predict the best tag sequence $\hat{T} = argmax_T P(T|O)$ using the tree:

$$\begin{aligned} \hat{T} &= \operatorname*{argmax}_T P(T|O) \\ &\approx \prod_{i=1}^{m} \operatorname*{argmax}_t P(t|o_i) \end{aligned} \tag{8.7}$$

The third step in dealing with NSWs is expansion into ordinary words. One NSW type, EXPN, is quite difficult to expand. These are the abbreviations and acronyms like **NY**. Generally these must be expanded by using an abbreviation dictionary, with any ambiguities dealt with by the homonym disambiguation algorithms discussed in the next section.

Expansion of the other NSW types is generally deterministic. Many expansions are trivial; for example, LSEQ expands to a sequence of words, one for each letter, ASWD expands to itself, NUM expands to a sequence of words representing the cardinal number, NORD expands to a sequence of words representing the ordinal number, and NDIG and NZIP both expand to a sequence of words, one for each digit.

HUNDREDS

TRAILING UNIT

Other types are slightly more complex; NYER expands to two pairs of digits, unless the year ends in *00*, in which case the four years are pronounced as a cardinal number (*2000* as `two thousand`) or in the **hundreds** method (e.g., 1800 as `eighteen hundred`). NTEL can be expanded just as a sequence of digits; alternatively, the last four digits can be read as **paired digits**, in which each pair is read as an integer. It is also possible to read them in a form known as **trailing unit**, in which the digits are read serially until the last nonzero digit, which is pronounced followed by the appropriate unit (e.g., *876-5000* as `eight seven six five thousand`). The expansion of NDATE, MONEY, and NTIME is left as exercises (8.1)-(8.4) for the reader.

Of course many of these expansions are dialect-specific. In Australian English, the sequence *33* in a telephone number is generally read `double three`. Other languages also present additional difficulties in non-standard word normalization. In French or German, for example, in addition to the above issues, normalization may depend on morphological properties. In French, the phrase *1 fille* ('one girl') is normalized to `une fille`, but *1 garçon* ('one boy') is normalized to `un garcçon`. Similarly, in German *Heinrich IV* ('Henry IV') can be normalized to `Heinrich der Vierte`, `Heinrich des Vierten`, `Heinrich dem Vierten`, or `Heinrich den Vierten` depending on the grammatical case of the noun (Demberg, 2006).

8.1.3 Homograph Disambiguation

HOMOGRAPHS

The goal of our NSW algorithms in the previous section was to determine which sequence of standard words to pronounce for each NSW. But sometimes determining how to pronounce even standard words is difficult. This is particularly true for **homographs**, which are words with the same spelling but different pronunciations. Here are some examples of the English homographs *use*, *live*, and *bass*:

(8.8) It's no use (/y uw s/) to ask to use (/y uw z/) the telephone.

(8.9) Do you live (/l ih v/) near a zoo with live (/l ay v/) animals?

(8.10) I prefer bass (/b ae s/) fishing to playing the bass (/b ey s/) guitar.

French homographs include *fils* (which has two pronunciations [fis] 'son' versus [fil] 'thread]), or the multiple pronunciations for *fier* ('proud' or 'to trust'), and *est* ('is' or 'East') (Divay and Vitale, 1997).

Luckily for the task of homograph disambiguation, the two forms of homographs in English (as well as in similar languages like French and German) tend to have different parts of speech.For example, the two forms of *use* above are (respectively) a noun and a verb, while the two forms of *live* are (respectively) a verb and a noun. Fig. 8.5 shows some interesting systematic relations between the pronunciation of some noun-verb and adj-verb homographs.

Final voicing			**Stress shift**			**-ate final vowel**		
	N (/s/)	**V** (/z/)		**N** (init. stress)	**V** (fin. stress)		**N/A** (final /ax/)	**V** (final /ey/)
use	y uw s	y uw z	record	r eh1 k axr0 d	r ix0 k ao1 r d	estimate	eh s t ih m ax t	eh s t ih m ey t
close	k l ow s	k l ow z	insult	ih1 n s ax0 l t	ix0 n s ah1 l t	separate	s eh p ax r ax t	s eh p ax r ey t
house	h aw s	h aw z	object	aa1 b j eh0 k t	ax0 b j eh1 k t	moderate	m aa d ax r ax t	m aa d ax r ey t

Figure 8.5 Some systematic relationships between homographs: final consonant (noun /s/ versus verb /z/), stress shift (noun initial versus verb final stress), and final vowel weakening in *-ate* noun/adjs.

Indeed, Liberman and Church (1992) showed that many of the most frequent homographs in 44 million words of AP newswire are disambiguatable just by using part-of-speech (the most frequent 15 homographs in order are: *use*, *increase*, *close*, *record*, *house*, *contract*, *lead*, *live*, *lives*, *protest*, *survey*, *project*, *separate*, *present*, *read*).

Thus because knowledge of part-of-speech is sufficient to disambiguate many homographs, in practice we perform homograph disambiguation by storing distinct pronunciations for these homographs labeled by part-of-speech, and then running a part-of-speech tagger to choose the pronunciation for a given homograph in context.

There are a number of homographs, however, where both pronunciations have the same part-of-speech. We saw two pronunciations for *bass* (fish versus instrument) above. Other examples of these include *lead* (because there are two noun pronunciations, /l iy d/ (a leash or restraint) and /l eh d/ (a metal)). We can also think of the task of disambiguating certain abbreviations (mentioned early as NSW disambiguation) as homograph disambiguation. For example, *Dr.* is ambiguous between `doctor` and `drive`, and *St.* between `Saint` or `street`. Finally, there are some words that differ in capitalizations like *polish*/*Polish*, which are homographs only in situations like sentence beginnings or all-capitalized text.

In practice, these latter classes of homographs that cannot be resolved using part-of-speech are often ignored in TTS systems. Alternatively, we can attempt to resolve them using the word sense disambiguation algorithms that we will introduce in Ch. 20, like the **decision-list** algorithm of Yarowsky (1997).

8.2 PHONETIC ANALYSIS

The next stage in synthesis is to take the normalized word strings from text analysis and produce a pronunciation for each word. The most important component here is a large pronunciation dictionary. Dictionaries alone turn out to be insufficient, because running text always contains words that don't appear in the dictionary. For example Black et al. (1998) used a British English dictionary, the OALD lexicon on the first section of the Penn Wall Street Journal Treebank. Of the 39923 words (tokens) in this section, 1775 word tokens (4.6%) were not in the dictionary, of which 943 are unique (i.e. 943 types). The distributions of these unseen word tokens was as follows:

names	unknown	typos and other
1360	351	64
76.6%	19.8%	3.6%

Thus the two main areas where dictionaries need to be augmented is in dealing with names and with other unknown words. We'll discuss dictionaries in the next section, followed by names, and then turn to grapheme-to-phoneme rules for dealing with other unknown words.

8.2.1 Dictionary Lookup

Phonetic dictionaries were introduced in Sec. **??** of Ch. 8. One of the most widely-used for TTS is the freely available CMU Pronouncing Dictionary (CMU, 1993), which has pronunciations for about 120,000 words. The pronunciations are roughly phonemic, from a 39-phone ARPAbet-derived phoneme set. Phonemic transcriptions means that instead of marking surface reductions like the reduced vowels [ax] or [ix], CMUdict marks each vowel with a stress tag, 0 (unstressed), 1 (stressed), or 2 (secondary stress). Thus (non-diphthong) vowels with 0 stress generally correspond to [ax] or [ix]. Most words have only a single pronunciation, but about 8,000 of the words have two or even three pronunciations, and so some kinds of phonetic reductions are marked in these pronunciations. The dictionary is not syllabified, although the nucleus is implicitly marked by the (numbered) vowel. Fig. 8.6 shows some sample pronunciations.

The CMU dictionary was designed for speech recognition rather than synthesis uses; thus it does not specify which of the multiple pronunciations to use for synthesis, does not mark syllable boundaries, and because it capitalizes the dictionary headwords, does not distinguish between e.g., *US* and *us* (the form *US* has the two pronunciations [AH1 S] and [Y UW1 EH1 S].

ANTECEDENTS	AE2 N T IH0 S IY1 D AH0 N T S	*PAKISTANI*	P AE2 K IH0 S T AE1 N IY0
CHANG	CH AE1 NG	*TABLE*	T EY1 B AH0 L
DICTIONARY	D IH1 K SH AH0 N EH2 R IY0	*TROTSKY*	T R AA1 T S K IY2
DINNER	D IH1 N ER0	*WALTER*	W AO1 L T ER0
LUNCH	L AH1 N CH	*WALTZING*	W AO1 L T S IH0 NG
MCFARLAND	M AH0 K F AA1 R L AH0 N D	*WALTZING(2)*	W AO1 L S IH0 NG

Figure 8.6 Some sample pronunciations from the CMU Pronouncing Dictionary.

The 110,000 word UNISYN dictionary, freely available for research purposes, resolves many of these issues as it was designed specifically for synthesis (Fitt, 2002). UNISYN gives syllabifications, stress, and some morphological boundaries. Furthermore, pronunciations in UNISYN can also be read off in any of dozens of dialects of English, including General American, RP British, Australia, and so on. The UNISYN uses a slightly different phone set; here are some examples:

going: `{ g * ou }.> i ng >`
antecedents: `{ * a n . tˆ i . s ˜ ii . d n! t }> s >`
dictionary: `{ d * i k . sh @ . n ˜ e . r ii }`

8.2.2 Names

As the error analysis above indicated, names are an important issue in speech synthesis. The many types can be categorized into personal names (first names and surnames), geographical names (city, street, and other place names), and commercial names (company and product names). For personal names alone, Spiegel (2003) gives an estimate from Donnelly and other household lists of about two million different surnames and 100,000 first names just for the United States. Two million is a very large number; an order of magnitude more than the entire size of the CMU dictionary. For this reason, most large-scale TTS systems include a large name pronunciation dictionary. As we saw in Fig. 8.6 the CMU dictionary itself contains a wide variety of names; in particular it includes the pronunciations of the most frequent 50,000 surnames from an old Bell Lab estimate of US personal name frequency, as well as 6,000 first names.

How many names are sufficient? Liberman and Church (1992) found that a dictionary of 50,000 names covered 70% of the name tokens in 44 million words of AP newswire. Interestingly, many of the remaining names (up to 97.43% of the tokens in their corpus) could be accounted for by simple modifications of these 50,000 names. For example, some name pronunciations can be created by adding simple stress-neutral suffixes like `s` or `ville` to names in the 50,000, producing new names as follows:

```
walters = walter+s   lucasville = lucas+ville   abelson = abel+son
```

Other pronunciations might be created by rhyme analogy. If we have the pronunciation for the name *Trotsky*, but not the name *Plotsky*, we can replace the initial /tr/ from *Trotsky* with initial /pl/ to derive a pronunciation for *Plotsky*.

Techniques such as this, including morphological decomposition, analogical formation, and mapping unseen names to spelling variants already in the dictionary (Fackrell and Skut, 2004), have achieved some success in name pronunciation. In general, however, name pronunciation is still difficult. Many modern systems deal with unknown names via the grapheme-to-phoneme methods described in the next section, often by building two predictive systems, one for names and one for non-names. Spiegel (2003, 2002) summarizes many more issues in proper name pronunciation.

8.2.3 Grapheme-to-Phoneme

GRAPHEME-TO-PHONEME

Once we have expanded non-standard words and looked them all up in a pronunciation dictionary, we need to pronounce the remaining, unknown words. The process of converting a sequence of letters into a sequence of phones is called **grapheme-to-phoneme** conversion, sometimes shortened **g2p**. The job of a grapheme-to-phoneme algorithm is thus to convert a letter string like *cake* into a phone string like `[K EY K]`.

LETTER-TO-SOUND

The earliest algorithms for grapheme-to-phoneme conversion were rules written by hand using the Chomsky-Halle phonological rewrite rule format of Eq. **??** in Ch. 7. These are often called **letter-to-sound** or LTS rules, and they are still used in some systems. LTS rules are applied in order, with later (default) rules only applying if the context for earlier rules are not applicable. A simple pair of rules for pronouncing the letter *c* might be as follows:

$$c \rightarrow \text{[k]} \;/\; \_\_ \;\{\text{a,o}\}\text{V} \qquad \text{; context-dependent} \tag{8.11}$$

$$c \rightarrow \text{[s]} \qquad \text{; context-independent} \tag{8.12}$$

Actual rules must be much more complicated (for example *c* can also be pronounced [ch] in *cello* or *concerto*). Even more complex are rules for assigning stress, which are famously difficult for English. Consider just one of the many stress rules from Allen et al. (1987), where the symbol *X* represents all possible syllable onsets:

$$V \rightarrow \text{[+stress]} \;/\; X \,\_\_\text{C*} \;\{\text{V}_{\text{short}}\ \text{C C?}|\text{V}\}\ \{\text{V}_{\text{short}}\ \text{C*}|\text{V}\} \tag{8.13}$$

This rule represents the following two situations:

1. Assign 1-stress to the vowel in a syllable preceding a weak syllable followed by a morpheme-final syllable containing a short vowel and 0 or more consonants (e.g. *difficult*)
2. Assign 1-stress to the vowel in a syllable preceding a weak syllable followed by a morpheme-final vowel (e.g. *oregano*)

While some modern systems still use such complex hand-written rules, most systems achieve higher accuracy by relying instead on automatic or semi-automatic methods based on machine learning. This modern probabilistic grapheme-to-phoneme problem was first formalized by Lucassen and Mercer (1984). Given a letter sequence *L*, we are searching for the most probable phone sequence *P*:

$$\hat{P} = \underset{P}{\operatorname{argmax}} P(P|L) \tag{8.14}$$

The probabilistic method assumes a training set and a test set; both sets are lists of words from a dictionary, with a spelling and a pronunciation for each word. The next subsections show how the popular **decision tree** model for estimating this probability $P(P|L)$ can be trained and applied to produce the pronunciation for an unseen word.

Finding a letter-to-phone alignment for the training set

Most letter-to-phone algorithms assume that we have an **alignment**, which tells us which phones align with each letter. We'll need this alignment for each word in the training set. Some letters might align to multiple phones (e.g., *x* often aligns to `k s`), while other letters might align with no phones at all, like the final letter of *cake* in the following alignment:

L:	c	a	k	e
	|	|	|	|
P:	K	EY	K	ε

One method for finding such a letter-to-phone alignment is the semi-automatic method of (Black et al., 1998). Their algorithm is semi-automatic because it relies on a hand-written list of the **allowable** phones that can realize each letter. Here are allowables lists for the letters *c* and *e*:

c: `k ch s sh t-s` ε
e: `ih iy er ax ah eh ey uw ay ow y-uw oy aa` ε

In order to produce an alignment for each word in the training set, we take this allowables list for all the letters, and for each word in the training set, we find all alignments between the pronunciation and the spelling that conform to the allowables list. From this large list of alignments, we compute, by summing over all alignments for all words, the total count for each letter being aligned to each phone (or multi-phone or ε). From these counts we can normalize to get for each phone p_i and letter l_j a probability $P(p_i|l_j)$:

$$P(p_i|l_j) = \frac{\text{count}(p_i, l_j)}{\text{count}(l_j)} \tag{8.15}$$

We can now take these probabilities and realign the letters to the phones, using the Viterbi algorithm to produce the best (Viterbi) alignment for each word, where the probability of each alignment is just the product of all the individual phone/letter alignments.

In this way we can produce a single good alignment A for each particular pair (P,L) in our training set.

Choosing the best phone string for the test set

Given a new word *w*, we now need to map its letters into a phone string. To do this, we'll first train a machine learning classifier, like a decision tree, on the aligned training set. The job of the classifier will be to look at a letter of the word and generate the most probable phone.

What features should we use in this decision tree besides the aligned letter l_i itself? Obviously we can do a better job of predicting the phone if we look at a window of surrounding letters; for example consider the letter *a*. In the word *cat*, the *a* is pronounce `AE`. But in our word *cake*, *a* is pronounced `EY`, because *cake* has a final *e*; thus knowing whether there is a final *e* is a useful feature. Typically we look at the *k* previous letters and the *k* following letters.

Another useful feature would be the correct identity of the previous phone. Knowing this would allow us to get some phonotactic information into our probability model.

Of course, we can't know the true identity of the previous phone, but we can approximate this by looking at the previous phone that was predicted by our model. In order to do this, we'll need to run our decision tree left to right, generating phones one by one.

In summary, in the most common decision tree model, the probability of each phone p_i is estimated from a window of k previous and k following letters, as well as the most recent k phones that were previously produced.

Fig. 8.7 shows a sketch of this left-to-right process, indicating the features that a decision tree would use to decide the letter corresponding to the letter *s* in the word *Jurafsky*. As this figure indicates, we can integrate stress prediction into phone prediction by augmenting our set of phones with stress information. We can do this by having two copies of each vowel (e.g., `AE` and `AE1`), or possibly even the three levels of stress `AE0`, `AE1`, and `AE2`, that we saw in the CMU lexicon. We'll also want to add other features into the decision tree, including the part-of-speech tag of the word (most part-of-speech taggers provide an estimate of the part-of-speech tag even for unknown words) and facts such as whether the previous vowel was stressed.

In addition, grapheme-to-phoneme decision trees can also include other more sophisticated features. For example, we can use classes of letters (corresponding roughly to consonants, vowels, liquids, and so on). In addition, for some languages, we need to know features about the following word. For example French has a phenomenon called **liaison**, in which the realization of the final phone of some words depends on whether there is a next word, and whether it starts with a consonant or a vowel. For example the French word *six* can be pronounced [sis] (in *j'en veux six* 'I want six'), [siz] (*six enfants* 'six children'), [si] (*six filles* 'six girls').

LIAISON

Finally, most synthesis systems build two separate grapheme-to-phoneme decision trees, one for unknown personal names and one for other unknown words. For pronouncing personal names it turns out to be helpful to use additional features that indicate which foreign language the names originally come from. Such features could be the output of a foreign-language classifier based on letter sequences (different languages have characteristic letter N-gram sequences).

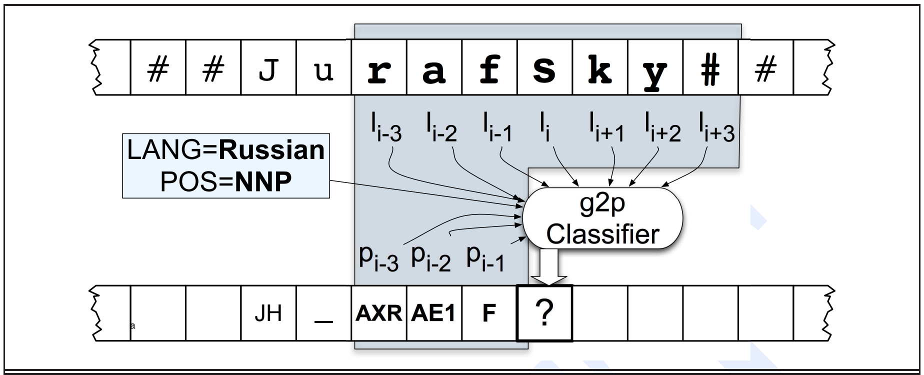
The decision tree is a conditional classifier, computing the phoneme string that has the highest conditional probability given the grapheme sequence. More recent grapheme-to-phoneme conversion makes use of a joint classifier, in which the hidden state is a combination of phone and grapheme called a **graphone**; see the end of the chapter for references.

8.3 Prosodic Analysis

PROSODY

The final stage of linguistic analysis is prosodic analysis. In poetry, the word **prosody** refers to the study of the metrical structure of verse. In linguistics and language processing, however, we use the term **prosody** to mean the study of the intonational and rhythmic aspects of language. More technically, prosody has been defined by Ladd (1996) as the 'use of suprasegmental features to convey sentence-level pragmatic meanings'. The term **suprasegmental** means above and beyond the level of the segment or phone, and refers especially to the uses of acoustic features like **F0 duration**, and

SUPRASEGMENTAL



# J u r a f s k y #
l_{i-3} l_{i-2} l_{i-1} l_i l_{i+1} l_{i+2} l_{i+3}
LANG=Russian
POS=NNP
g2p Classifier
p_{i-3} p_{i-2} p_{i-1}
JH _ AXR AE1 F ?

Figure 8.7 The process of converting graphemes to phonemes, showing the left-to-right process making a decision for the letter *s*. The features used by the decision tree are shown in blue. We have shown the context window $k = 3$; in real TTS systems the window size is likely to be 5 or even larger.

energy independently of the phone string.

By **sentence-level pragmatic meaning**, Ladd is referring to a number of kinds of meaning that have to do with the relation between a sentence and its discourse or external context. For example, prosody can be used to mark **discourse structure or function**, like the difference between statements and questions, or the way that a conversation is structured into segments or subdialogs. Prosody is also used to mark **saliency**, such as indicating that a particular word or phrase is important or salient. Finally, prosody is heavily used for affective and emotional meaning, such as expressing happiness, surprise, or anger.

In the next sections we will introduce the three aspects of prosody, each of which is important for speech synthesis: **prosodic prominence**, **prosodic structure** and **tune**. Prosodic analysis generally proceeds in two parts. First, we compute an abstract representation of the prosodic prominence, structure and tune of the text. For unit selection synthesis, this is all we need to do in the text analysis component. For diphone and HMM synthesis, we have one further step, which is to predict **duration** and **F0** values from these prosodic structures.

8.3.1 Prosodic Structure

PROSODIC PHRASING

INTONATION PHRASES

INTERMEDIATE PHRASE

Spoken sentences have prosodic structure in the sense that some words seem to group naturally together and some words seem to have a noticeable break or disjuncture between them. Often prosodic structure is described in terms of **prosodic phrasing**, meaning that an utterance has a prosodic phrase structure in a similar way to it having a syntactic phrase structure. For example, in the sentence *I wanted to go to London, but could only get tickets for France* there seems to be two main **intonation phrases**, their boundary occurring at the comma. Furthermore, in the first phrase, there seems to be another set of lesser prosodic phrase boundaries (often called **intermediate phrase**s) that split up the words as follows *I wanted* | *to go* | *to London*.

Prosodic phrasing has many implications for speech synthesis; the final vowel of a phrase is longer than usual, we often insert a pause after an intonation phrases, and, as we will discuss in Sec. 8.3.6, there is often a slight drop in F0 from the beginning of an intonation phrase to its end, which resets at the beginning of a new intonation phrase.

Practical phrase boundary prediction is generally treated as a binary classification task, where we are given a word and we have to decide whether or not to put a prosodic boundary after it. A simple model for boundary prediction can be based on deterministic rules. A very high-precision rule is the one we saw for sentence segmentation: insert a boundary after punctuation. Another commonly used rule inserts a phrase boundary before a function word following a content word.

More sophisticated models are based on machine learning classifiers. To create a training set for classifiers, we first choose a corpus, and then mark every prosodic boundaries in the corpus. One way to do this prosodic boundary labeling is to use an intonational model like ToBI or Tilt (see Sec. 8.3.4), have human labelers listen to speech and label the transcript with the boundary events defined by the theory. Because prosodic labeling is extremely time-consuming, however, a text-only alternative is often used. In this method, a human labeler looks only at the text of the training corpus, ignoring the speech. The labeler marks any juncture between words where they feel a prosodic boundary might legitimately occur if the utterance were spoken.

Given a labeled training corpus, we can train a decision tree or other classifier to make a binary (boundary vs. no boundary) decision at every juncture between words (Wang and Hirschberg, 1992; Ostendorf and Veilleux, 1994; Taylor and Black, 1998).

Features that are commonly used in classification include:

- **Length features**: phrases tend to be of roughly equal length, and so we can use various feature that hint at phrase length (Bachenko and Fitzpatrick, 1990; Grosjean et al., 1979; Gee and Grosjean, 1983).
  - The total number of words and syllables in utterance
  - The distance of the juncture from the beginning and end of the sentence (in words or syllables)
  - The distance in words from the last punctuation mark
- **Neighboring part-of-speech and punctuation**:
  - The part-of-speech tags for a window of words around the juncture. Generally the two words before and after the juncture are used.
  - The type of following punctuation

There is also a correlation between prosodic structure and the **syntactic structure** that will be introduced in Ch. 12, Ch. 13, and Ch. 14 (Price et al., 1991). Thus robust parsers like Collins (1997) can be used to label the sentence with rough syntactic information, from which we can extract syntactic features such as the size of the biggest syntactic phrase that ends with this word (Ostendorf and Veilleux, 1994; Koehn et al., 2000).

8.3.2 Prosodic prominence

PROMINENT

In any spoken utterance, some words sound more **prominent** than others. Prominent words are perceptually more salient to the listener; speakers make a word more salient in English by saying it louder, saying it slower (so it has a longer duration), or by varying F0 during the word, making it higher or more variable.

PITCH ACCENT

BEAR

We generally capture the core notion of prominence by associating a linguistic marker with prominent words, a marker called **pitch accent**. Words which are prominent are said to **bear** (be associated with) a pitch accent. Pitch accent is thus part of the phonological description of a word in context in a spoken utterance.

Pitch accent is related to **stress**, which we discussed in Ch. 7. The stressed syllable of a word is where pitch accent is realized. In other words, if a speaker decides to highlight a word by giving it a pitch accent, the accent will appear on the stressed syllable of the word.

The following example shows accented words in capital letters, with the stressed syllable bearing the accent (the louder, longer, syllable) in boldface:

(8.16) I'm a little SUR**PRISED** to hear it **CHAR**ACTERIZED as UP**BEAT**.

Note that the function words tend not to bear pitch accent, while most of the content words are accented. This is a special case of the more general fact that very informative words (content words, and especially those that are new or unexpected) tend to bear accent (Ladd, 1996; Bolinger, 1972).

NUCLEAR ACCENT

We've talked so far as if we only need to make a binary distinction between accented and unaccented words. In fact we generally need to make more fine-grained distinctions. For example the last accent in a phrase generally is perceived as being more prominent than the other accents. This prominent last accent is called the **nuclear accent**. Emphatic accents like nuclear accent are generally used for semantic purposes, for example to indicate that a word is the **semantic focus** of the sentence (see Ch. 21) or that a word is contrastive or otherwise important in some way. Such emphatic words are the kind that are often written IN CAPITAL LETTERS or with **STARS** around them in SMS or email or *Alice in Wonderland*; here's an example from the latter:

(8.17) 'I know SOMETHING interesting is sure to happen,' she said to herself,

Another way that accent can be more complex than just binary is that some words can be **less** prominent than usual. We introduced in Ch. 7 the idea that function words are often phonetically very **reduced**.

A final complication is that accents can differ according to the **tune** associated with them; for example accents with particularly high pitch have different functions than those with particularly low pitch; we'll see how this is modeled in the ToBI model in Sec. 8.3.4.

Ignoring tune for the moment, we can summarize by saying that speech synthesis systems can use as many as four levels of prominence: **emphatic accent**, **pitch accent**, **unaccented**, and **reduced**. In practice, however, many implemented systems make do with a subset of only two or three of these levels.

Let's see how a 2-level system would work. With two-levels, pitch accent prediction is a binary classification task, where we are given a word and we have to decide whether it is accented or not.

Since content words are very often accented, and function words are very rarely accented, the simplest accent prediction system is just to accent all content words and no function words. In most cases better models are necessary.

In principle accent prediction requires sophisticated semantic knowledge, for example to understand if a word is new or old in the discourse, whether it is being used contrastively, and how much new information a word contains. Early models made use of sophisticated linguistic models of all of this information (Hirschberg, 1993). But Hirschberg and others showed better prediction by using simple, robust features that correlate with these sophisticated semantics.

For example, the fact that new or unpredictable information tends to be accented can be modeled by using robust features like N-grams or TF*IDF (Pan and Hirschberg, 2000; Pan and McKeown, 1999). The unigram probability of a word $P(w_i)$ and its bigram probability $P(w_i|w_{i-1})$, both correlate with accent; the more probable a word, the less likely it is to be accented. Similarly, an information-retrieval measure known as

TF*IDF

TF*IDF (Term-Frequency/Inverse-Document Frequency; see Ch. 23) is a useful accent predictor. TF*IDF captures the semantic importance of a word in a particular document d, by downgrading words that tend to appear in lots of different documents in some large background corpus with N documents. There are various versions of TF*IDF; one version can be expressed formally as follows, assuming Nw is the frequency of w in the document d, and k is the total number of documents in the corpus that contain w:

$$\text{TF*IDF}(w) = Nw \times \log(\frac{N}{k}) \tag{8.18}$$

ACCENT RATIO

For words which have been seen enough times in a training set, the **accent ratio** feature can be used, which models a word's individual probability of being accented. The accent ratio of a word is equal to the estimated probability of the word being accented if this probability is significantly different from 0.5, and equal to 0.5 otherwise. More formally,

$$AccentRatio(w) = \begin{cases} \frac{k}{N} & \texttt{if } B(k,N,0.5) \leq 0.05 \\ 0.5 & \texttt{otherwise} \end{cases}$$

where N is the total number of times the word w occurred in the training set, k is the number of times it was accented, and $B(k,n,0.5)$ is the probability (under a binomial distribution) that there are k successes in n trials if the probability of success and failure is equal (Nenkova et al., 2007; Yuan et al., 2005).

Features like part-of-speech, N-grams, TF*IDF, and accent ratio can then be combined in a decision tree to predict accents. While these robust features work relatively well, a number of problems in accent prediction still remain the subject of research.

For example, it is difficult to predict which of the two words should be accented in adjective-noun or noun-noun compounds. Some regularities do exist; for example adjective-noun combinations like *new truck* are likely to have accent on the right word (*new TRUCK*), while noun-noun compounds like *TREE surgeon* are likely to have accent on the left. But the many exceptions to these rules make accent prediction in noun compounds quite complex. For example the noun-noun compound *APPLE cake* has the accent on the first word while the noun-noun compound *apple PIE* or *city HALL* both have the accent on the second word (Liberman and Sproat, 1992; Sproat, 1994, 1998a).

CLASH
LAPSE

Another complication has to do with rhythm; in general speakers avoid putting accents too close together (a phenomenon known as **clash**) or too far apart (**lapse**).

Thus *city HALL* and *PARKING lot* combine as *CITY hall PARKING lot* with the accent on *HALL* shifting forward to *CITY* to avoid the clash with the accent on *PARKING* (Liberman and Prince, 1977),

Some of these rhythmic constraints can be modeled by using machine learning techniques that are more appropriate for sequence modeling. This can be done by running a decision tree classifier left to right through a sentence, and using the output of the previous word as a feature, or by using more sophisticated machine learning models like Conditional Random Fields (CRFs) (Gregory and Altun, 2004).

8.3.3 Tune

TUNE

QUESTION RISE

FINAL FALL

Two utterances with the same prominence and phrasing patterns can still differ prosodically by having different **tunes**. The **tune** of an utterance is the rise and fall of its F0 over time. A very obvious example of tune is the difference between statements and yes-no questions in English. The same sentence can be said with a final rise in F0 to indicate a yes-no-question, or a final fall in F0 to indicate a declarative intonation. Fig. 8.8 shows the F0 track of the same words spoken as a question or a statement. Note that the question rises at the end; this is often called a **question rise**. The falling intonation of the statement is called a **final fall**.

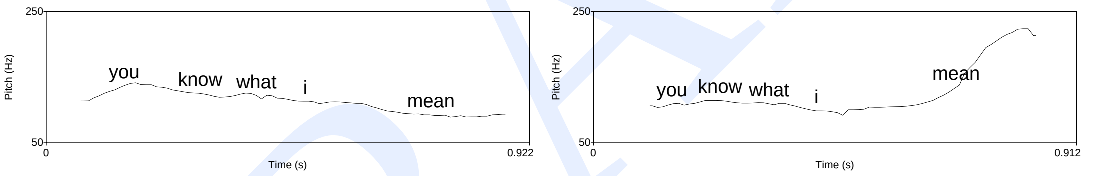


Figure 8.8 The same text read as the statement *You know what I mean.* (on the left) and as a question *You know what I mean?* (on the right). Notice that yes-no-question intonation in English has a sharp final rise in F0.

CONTINUATION RISE

It turns out that English makes very wide use of tune to express meaning. Besides this well known rise for yes-no questions, an English phrase containing a list of nouns separated by commas often has a short rise called a **continuation rise** after each noun. English also has characteristic contours to express contradiction, to express surprise, and many more.

The mapping between meaning and tune in English is extremely complex, and linguistic theories of intonation like ToBI have only begun to develop sophisticated models of this mapping.

In practice, therefore, most synthesis systems just distinguish two or three tunes, such as the **continuation rise** (at commas), the **question rise** (at question mark if the question is a yes-no question), and a **final fall** otherwise.

8.3.4 More sophisticated models: ToBI

While current synthesis systems generally use simple models of prosody like the ones discussed above, recent research focuses on the development of much more sophisticated models. We’ll very briefly discuss the **ToBI**, and **Tilt** models here.

ToBI

TOBI

One of the most widely used linguistic models of prosody is the **ToBI** (Tone and Break Indices) model (Silverman et al., 1992; Beckman and Hirschberg, 1994; Pierrehumbert, 1980; Pitrelli et al., 1994). ToBI is a phonological theory of intonation which models prominence, tune, and boundaries. ToBI’s model of prominence and tunes is based on the 5 **pitch accents** and 4 **boundary tones** shown in Fig. 8.9.

Pitch Accents		Boundary Tones	
H*	peak accent	**L-L%**	“final fall”: “declarative contour” of American English”
L*	low accent	**L-H%**	continuation rise
L*+H	scooped accent	**H-H%**	“question rise”: cantonical yes-no question contour
L+H*	rising peak accent	**H-L%**	final level plateau (plateau because H- causes “upstep” of following)
H+!H*	step down		

Figure 8.9 The accent and boundary tones labels from the ToBI transcription system for American English intonation (Beckman and Ayers, 1997; Beckman and Hirschberg, 1994).

BOUNDARY TONES

An utterance in ToBI consists of a sequence of intonational phrases, each of which ends in one of the four **boundary tones**. The boundary tones are used to represent the utterance final aspects of tune discussed in Sec. 8.3.3. Each word in the utterances can optionally be associated with one of the five types of pitch accents.

Each intonational phrase consists of one or more **intermediate phrase**. These phrases can also be marked with kinds of boundary tone, including the **%H** high initial boundary tone, which is used to mark a phrase which is particularly high in the speakers’ pitch range, as well as final phrase accents **H-** and **L-**.

BREAK INDEX

In addition to accents and boundary tones, ToBI distinguishes four levels of phrasing, which are labeled on a separate **break index** tier. The largest levels of phrasing are the intonational phrase (break index **4**) and the intermediate phrase (break index **3**), and were discussed above. Break index **2** is used to mark a disjuncture or pause between words that is smaller than an intermediate phrase, while **1** is used for normal phrase-medial word boundaries.

TIERS

Fig. 8.10 shows the tone, orthographic, and phrasing **tiers** of a ToBI transcription, using the `praat` program. We see the same sentence read with two different intonation patterns. In (a), the word *Marianna* is spoken with a high H* accent, and the sentence has the declarative boundary tone L-L%. In (b), the word *Marianna* is spoken with a low L* accent and the yes-no question boundary tone H-H%. One goal of ToBI is to express different meanings to the different type of accents. Thus, for example, the L* accent adds a meaning of *surprise* to the sentence (i.e., with a connotation like ‘Are you really saying it was Marianna?’). (Hirschberg and Pierrehumbert, 1986; Steedman, 2003).

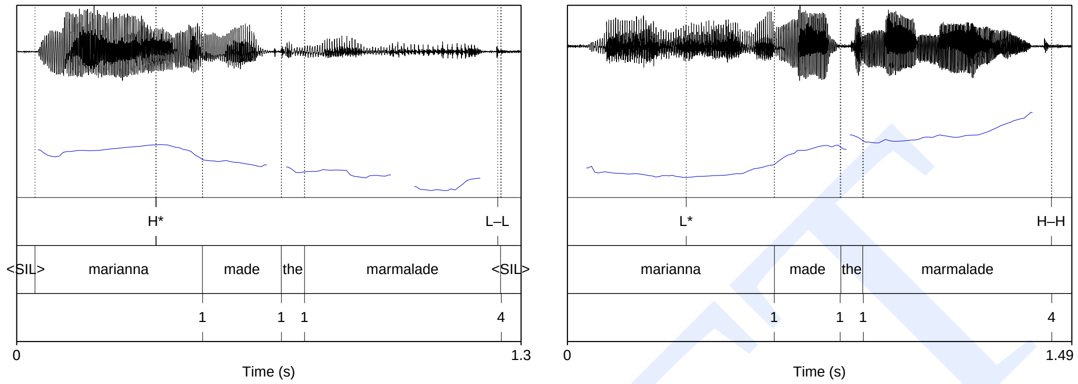


Figure 8.10 The same sentence read by Mary Beckman with two different intonation patterns and transcribed in ToBI. (a) shows an H* accent and the typical American English declarative final fall L-L%. (b) shows the L* accent, with the typical American English yes-no question rise H-H%.

ToBI models have been proposed for many languages, such as the J_TOBI system for Japanese (Venditti, 2005); see Jun (2005).

Other Intonation models

TILT The **Tilt** model (Taylor, 2000) resembles ToBI in using sequences of intonational events like accents and boundary tones. But Tilt does not use ToBI-style discrete phonemic classes for accents. Instead, each event is modeled by continuous parameters that represent the F0 shape of the accent.

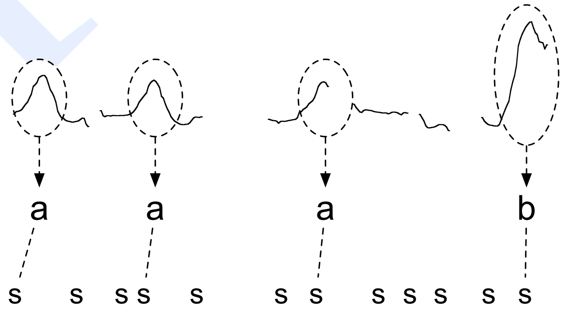


Figure 8.11 Schematic view of events in the Tilt model (Taylor, 2000). Each pitch accent (*a*) and boundary tone (*b*) is aligned with a syllable nucleus *s*.

Instead of giving each event a category label, as in ToBI, each Tilt prosodic event is characterized by a set of three acoustic parameters: the duration, the amplitude, and the

TILT **tilt** parameter. These acoustic parameters are trained on a corpus which has been hand-labeled for pitch accents (**a**) and boundary tones (**b**). The human labeling specifies the syllable which bears the accent or tone; the acoustic parameters are then trained automatically from the wavefile. Fig. 8.11 shows a sample of a Tilt representation. Each accent in Tilt is viewed as having a (possibly zero) **rise component** up to peak, followed by a (possible zero) **fall component**. An automatic accent detector finds the start, peak, and end point of each accent in the wavefile, which determines the duration and amplitude of the rise and fall components. The tilt parameter is an abstract description of the F0 slope of an event, calculated by comparing the relative sizes of the rise and fall for an event. A tilt value of 1.0 indicates a rise, tilt of -1.0 a fall, 0 equal rise and fall, -0.5 is an accent with a rise and a larger fall, and so on:

$$\begin{aligned}\text{tilt} &= \frac{\text{tilt}_{\text{amp}}+\text{tilt}_{\text{dur}}}{2}\\ &= \frac{|A_{\text{rise}}|-|A_{\text{fall}}|}{|A_{\text{rise}}|+|A_{\text{fall}}|}+\frac{D_{\text{rise}}-D_{\text{fall}}}{D_{\text{rise}}+D_{\text{fall}}}\end{aligned} \tag{8.19}$$

See the end of the chapter for pointers to a wide variety of other intonational models.

8.3.5 Computing duration from prosodic labels

The results of the text analysis processes described so far is a string of phonemes, annotated with words, with pitch accent marked on relevant words, and appropriate boundary tones marked. For the **unit selection** synthesis approaches that we will describe in Sec. 8.5, this is a sufficient output from the text analysis component.

For **diphone** synthesis, as well as other approaches like formant synthesis, we also need to specify the **duration** and the **F0** values of each segment.

Phones vary quite a bit in duration. Some of the duration is inherent to the identity of the phone itself. Vowels, for example, are generally much longer than consonants; in the Switchboard corpus of telephone speech, the phone [aa] averages 118 milliseconds, while [d] averages 68 milliseconds. But phone duration is also affected by a wide variety of contextual factors, which can be modeled by rule-based or statistical methods.

The most well-known of the rule-based methods is the method of Klatt (1979), which uses rules to model how the the average or 'context-neutral' duration of a phone $\bar{d}$ is lengthened or shortened by context, while staying above a minimum duration d_{min}. Each rule is associated with a duration multiplicative factor; some examples:

Prepasual Lengthening: The vowel or syllabic consonant in the syllable before a pause is lengthened by 1.4.

Non-phrase-final Shortening: Segments which are not phrase-final are shortened by 0.6. Phrase-final postvocalic liquids and nasals are lengthened by 1.4.

Unstressed Shortening: Unstressed segments are more compressible, so their minimum duration d_{min} is halved, and are shortened by .7 for most phone types.

Lengthening for Accent: A vowel which bears accent is lengthened by 1.4

Shortening in Clusters: A consonant followed by a consonant is shortened by 0.5.

Pre-voiceless shortening: Vowels are shortened before a voiceless plosive by 0.7

Given the set of N factor weights f, the Klatt formula for the duration of a phone is:

$$d = d_{\text{min}} + \prod_{i=1}^{N} f_i \times (\bar{d} - d_{\text{min}}) \tag{8.20}$$

More recent machine-learning systems use the Klatt hand-written rules as the basis for defining features, for example using features such as the following:

- identity of the left and right context phone
- lexical stress and accent values of current phone
- position in syllable, word, phrase
- following pause

SUM-OF-PRODUCTS

We can then train machine learning classifiers like decision trees or the **sum-of-products** model (van Santen, 1994, 1997, 1998), to combine the features to predict the final duration of the segment.

8.3.6 Computing F0 from prosodic labels

TARGET POINTS

For diphone, articulatory, HMM, and formant synthesis we also need to specify the F0 values of each segment. For the tone sequence models like ToBI or Tilt, this F0 generation can be done by specifying F0 **target points** for each pitch accent and boundary tone; the F0 contour for the whole sentence can be created by interpolating among these targets (Anderson et al., 1984).

PITCH RANGE

BASELINE FREQUENCY

TOPLINE

REFERENCE LINE

In order to specify a target point we need to describe what it is (the F0 value) and when it occurs (the exact time at which this peak or trough occurs in the syllable). The F0 values of the target points are generally not specified in absolute terms of Hertz. Instead, they are defined relative to **pitch range**. A speaker's **pitch range** is the range between the lowest frequency they use in a particular utterance (the **baseline frequency**) and the the highest frequency in the utterance (the **topline**). In some models, target points are specified relative to a line in between called the **reference line**.

For example, we might write a rule specifying that the very beginning of an utterance have a target point of 50% (halfway between the baseline and topline). In the rule-based system of Jilka et al. (1999) the target point for an H* accent is at 100% (the topline) and for an L* accent at 0% (at the baseline). L+H* accents have two target points, at 20% and 100%. Final boundary tones H-H% and L-L% are extra-high and extra-low at 120% and -20% respectively.

ALIGNMENT

Second, we must also specify exactly where in the accented syllable the targets apply; this is known as accent **alignment**. In the rule-based system of Jilka et al. (1999), again, H* accents are aligned 60% of the way through the voiced part of the accent syllable (although IP-initial accents are aligned somewhat later in the syllable, while IP-final accents are aligned somewhat earlier).

Instead of writing these rules by hand, the mapping from pitch accent sequence to F0 value may be learned automatically. For example Black and Hunt (1996) used linear regression to assign target values to each syllable. For each syllable with a pitch accent or boundary tone, they predicted three target values, at the beginning, middle, and end of the syllable. They trained three separate linear regression models, one for each of the three positions in the syllable. Features included:

- accent type on the current syllable as well as two previous and two following syllables
- lexical stress of this syllable and surrounding syllables
- number of syllables to start of phrase and to end of phrase
- number of accented syllables to end of phrase

Such machine learning models require a training set that is labeled for accent; a number of such prosodically-labeled corpora exist, although it is not clear how well these models generalize to unseen corpora.

DECLINATION

Finally, F0 computation models must model the fact that pitch tends to decline through a sentence; this subtle drop in pitch across an utterance is called **declination**; an example is shown in Fig. 8.12.

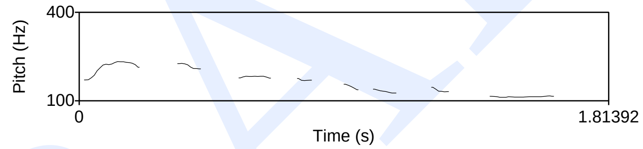


Figure 8.12 F0 declination in the sentence ‘I was pretty goofy for about twenty-four hours afterwards’.

The exact nature of declination is a subject of much research; in some models, it is treated by allowing the baseline (or both baseline and top-line) to decrease slowly over the utterance. In ToBI-like models, this downdrift in F0 is modeled by two separate components; in addition to declination, certain high tones are marked as carrying **downstep**. Each downstepped high accent causes the pitch range to be compressed, resulting in a lowered topline for each such accent.

DOWNSTEP

8.3.7 Final result of text analysis: Internal Representation

The final output of text analysis is what we called the **internal representation** of the input text sentence. For unit selection synthesis, the internal representation can be as simple as a phone string together with indications of prosodic boundaries and prominent syllables, as shown in Fig. 8.1. For diphone synthesis as well as non-concatenative synthesis algorithms the internal representation must also include a duration and an F0 value for each phone.

Fig. 8.13 shows some sample TTS output from the FESTIVAL (Black et al., 1999) diphone speech synthesis system for the sentence *Do you really want to see all of it?*. This output, together with the F0 values shown in Fig. 8.14 would be the input to the **waveform synthesis** component described in Sec. 8.4. The durations here are computed by a CART-style decision tree (Riley, 1992).

				H*												L*		L- H%			
do		you		really				want				to		see		all		of		it	
d	uw	y	uw	r	ih	l	iy	w	aa	n	t	t	ax	s	iy	ao	l	ah	v	ih	t
110	110	50	50	75	64	57	82	57	50	72	41	43	47	54	130	76	90	44	62	46	220

Figure 8.13 Output of the FESTIVAL (Black et al., 1999) generator for the sentence *Do you really want to see all of it?*, together with the F0 contour shown in Fig. 8.14. Figure thanks to Paul Taylor.

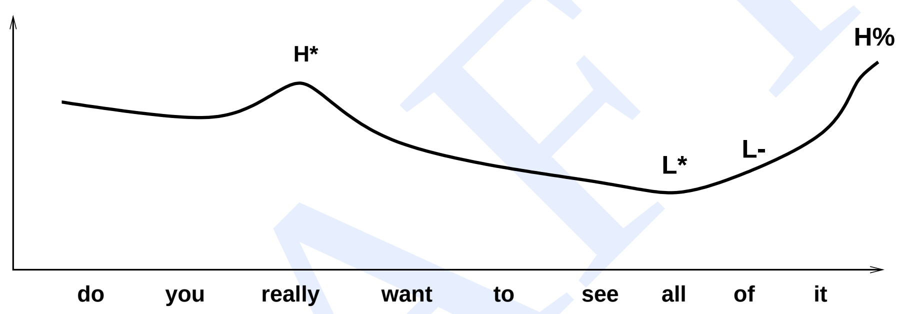


Figure 8.14 The F0 contour for the sample sentence generated by the FESTIVAL synthesis system in Fig. 8.13, thanks to Paul Taylor.

As was suggested above, determining the proper prosodic pattern for a sentence is difficult, as real-world knowledge and semantic information is needed to know which syllables to accent, and which tune to apply. This sort of information is difficult to extract from the text and hence prosody modules often aim to produce a "neutral declarative" version of the input text, which assume the sentence should be spoken in a default way with no reference to discourse history or real-world events. This is one of the main reasons why intonation in TTS often sounds "wooden".

8.4 Diphone Waveform synthesis

We are now ready to see how the internal representation can be turned into a waveform. We will present two kinds of **concatentative** synthesis: **diphone synthesis** in this section, and **unit selection synthesis** in the next section.

Recall that for diphone synthesis, our internal representation is as shown in Fig. 8.13 and Fig. 8.14, consisting of a list of phones, each phone associated with a duration and a set of F0 targets.

The diphone concatenative synthesis model generates a waveform from a sequence of phones by selecting and concatenating units from a prerecorded database of **diphones**. A diphone is a phone-like unit going from roughly the middle of one phone to

DIPHONES

the middle of the following phone. Diphone concatenative synthesis can be characterized by the following steps:

Training:

1. Record a single speaker saying an example of each diphone.
2. Cut each diphone out from the speech and store all diphones in a diphone database.

Synthesis:

1. Take from the database a sequence of diphones that corresponds to the desired phone sequence.
2. Concatenate the diphones, doing some slight signal processing at the boundaries
3. Use signal processing to change the prosody (f0, duration) of the diphone sequence to the desired prosody.

COARTICULATION

We tend to use diphones rather than phones for concatenative synthesis because of the phenomenon of **coarticulation**. In Ch. 7 we defined **coarticulation** as the movement of articulators to anticipate the next sound, or perseverating movement from the last sound. Because of coarticulation, each phone differs slightly depending on the previous and following phone. This if we just concatenated phones together, we would have very large discontinuities at the boundaries.

In a diphone, we model this coarticulation by including the transition to the next phone inside the unit. The diphone [w-eh], for example, includes the transition from the [w] phone to the [eh] phone. Because a diphone is defined from the middle of one phone to the middle of the next, when we concatenate the diphones, we are concatenating the middle of phones, and the middle of phones tend to be less influenced by the context. Fig. **??** shows the intuition that the beginning and end of the vowel [eh] have much more movement than the center.

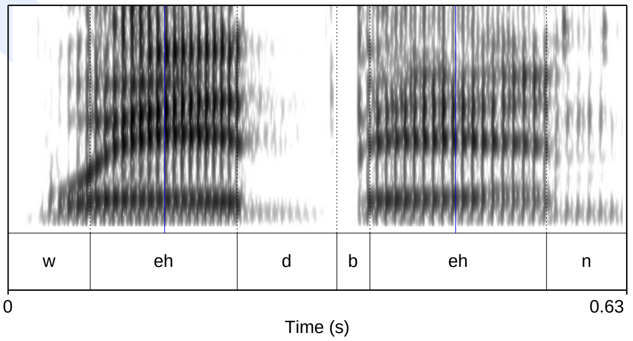


Figure 8.15 The vowel [eh] in different surrounding contexts, in the words *wed* and *Ben*. Notice the differences in the second formants (F2) at the beginning and end of the [eh], but the relatively steady state portion in the middle at the blue line.

8.4.1 Building a diphone database

There are six steps in building a diphone database:

1. Create a **diphone inventory**
2. Recruit a speaker
3. Create a text for the speaker to read for each diphone
4. Record the speaker reading each diphone
5. Segment, label, and pitch-mark the diphones
6. Excise the diphones

What is the inventory of diphones that we need for a system? If we have 43 phones (like the AT&T diphone system of Olive et al. (1998)), there are $43^2 = 1849$ hypothetically possible diphone combinations. Not all of these diphones can actually occur. For example, the rules of English **phonotactics** rules out some combinations; phones like [h], [y], and [w] can only occur before vowels. In addition, some diphone systems don't bother storing diphones if there is no possible coarticulation between the phones, such as across the silence between successive voiceless stops. The 43-phone system of Olive et al. (1998) thus has only 1162 diphones rather than the 1849 hypothetically possible set.

VOICE TALENT

VOICE

Next we recruit our speaker, often called a **voice talent**. The database of diphones for this speaker is called a **voice**; commercial systems often have multiple voices, such as one male and one female voice.

CARRIER PHRASE

We'll now create a text for the voice talent to say, and record each diphone. The most important thing in recording diphones is to keep them as consistent as possible; if possible, they should have constant pitch, energy, and duration, so they are easy to paste together without noticeable breaks. We do this by enclosing each diphone to be recorded in a **carrier phrase**. By putting the diphone in the middle of other phones, we keep utterance-final lengthening or initial phone effects from making any diphone louder or quieter than the others. We'll need different carrier phrases for consonant-vowel, vowel-consonant, phone-silence, and silence-phone sequences. For example, a consonant vowel sequence like [b aa] or [b ae] could be embedded between the syllables [t aa] and [m aa]:

pause t aa b aa m aa pause
pause t aa b ae m aa pause
pause t aa b eh m aa pause
...

If we have an earlier synthesizer voice lying around, we usually use that voice to read the prompts out loud, and have our voice talent repeat after the prompts. This is another way to keep the pronunciation of each diphone consistent. It is also very important to use a a high quality microphone and a quiet room or, better, a studio sound booth.

Once we have recorded the speech, we need to label and segment the two phones that make up each diphone. This is usually done by running a speech recognizer in **forced alignment mode**. In forced alignment mode, a speech recognition is told exactly what the phone sequence is; its job is just to find the exact phone boundaries in the waveform. Speech recognizers are not completely accurate at finding phone boundaries, and so usually the automatic phone segmentation is hand-corrected.

We now have the two phones (for example [b aa]) with hand-corrected boundaries. There are two ways we can create the /b-aa/ diphone for the database. One method is to use rules to decide how far into the phone to place the diphone boundary. For example, for stops, we put place the diphone boundary 30% of the way into the phone. For most other phones, we place the diphone boundary 50% into the phone.

OPTIMAL COUPLING

A more sophisticated way to find diphone boundaries is to store the entire two phones, and wait to excise the diphones until we are know what phone we are about to concatenate with. In this method, known as **optimal coupling**, we take the two (complete, uncut) diphones we need to concatenate, and we check every possible cutting point for each diphones, choosing the two cutting points that would make the final frame of the first diphone acoustically most similar to the end frame of the next diphone (Taylor and Isard, 1991; Conkie and Isard, 1996). Acoustical similar can be measured by using **cepstral similarity**, to be defined in Sec. **??**.

8.4.2 Diphone concatenation and TD-PSOLA for prosodic adjustment

We are now ready to see the remaining steps for synthesizing an individual utterance. Assume that we have completed text analysis for the utterance, and hence arrived at a sequence of diphones and prosodic targets, and that we have also grabbed the appropriate sequence of diphones from the diphone database. Next we need to concatenate the diphones together and then adjust the prosody (pitch, energy, and duration) of the diphone sequence to match the prosodic requirements from the intermediate representation.

CLICK

PITCH-SYNCHRONOUSLY

Given two diphones, what do we need to do to concatenate them successfully? If the waveforms of the two diphones edges across the juncture are very different, a perceptible **click** will result. Thus we need to apply a windowing function to the edge of both diphones so that the samples at the juncture have low or zero amplitude. Furthermore, if both diphones are voiced, we need to insure that the two diphones are joined **pitch-synchronously**. This means that the pitch periods at the end of the first diphone must line up with the pitch periods at the beginning of the second diphone; otherwise the resulting single irregular pitch period at the juncture is perceptible as well.

TD-PSOLA

Now given our sequence of concatenated diphones, how do we modify the pitch and duration to meet our prosodic requirements? It turns out there is a very simple algorithm for doing this called **TD-PSOLA** (**Time-Domain Pitch-Synchronous OverLap-and-Add**).

PITCH MARKING

PITCH TRACKING

As we just said, a **pitch-synchronous** algorithm is one in which we do something at each pitch period or **epoch**. For such algorithms it is important to have very accurate pitch markings: measurements of exactly where each pitch pulse or **epoch** occurs. An epoch can be defined by the instant of maximum glottal pressure, or alternatively by the instant of glottal closure. Note the distinction between **pitch marking** or **epoch detection** and **pitch tracking**. Pitch tracking gives the value of F0 (the average cycles per second of the glottis) at each particular point in time, averaged over a neighborhood.

Pitch marking finds the exact point in time at each vibratory cycle at which the vocal folds reach some specific point (epoch).

ELECTROGLOTTOGRAPH
EGG (ELECTROGLOTTOGRAPH)
LARYNGOGRAPH
LX (LARYNGOGRAPH)

Epoch-labeling can be done in two ways. The traditional way, and still the most accurate, is to use an **electroglottograph** or **EGG (electroglottograph)** (often also called a **laryngograph** or **Lx (laryngograph)**). An EGG is a device which straps onto the (outside of the) speaker's neck near the larynx and sends a small current through the Adam's apple. A transducer detects whether the glottis is open or closed by measuring the impedance across the vocal folds. Some modern synthesis databases are still recorded with an EGG. The problem with using an EGG is that it must be attached to the speaker while they are recording the database. Although an EGG isn't particularly invasive, this is still annoying, and the EGG must be used during recording; it can't be used to pitch-mark speech that has already been collected. Modern epoch detectors are now approaching a level of accuracy that EGGs are no longer used in most commercial TTS engines. Algorithms for epoch detection include Brookes and Loke (1999), Veldhuis (2000).

OVERLAP-AND-ADD
OLA

Given an epoch-labeled corpus, the intuition of TD-PSOLA is that we can modify the pitch and duration of a waveform by extracting a frame for each pitch period (windowed so that the frame doesn't have sharp edges) and then recombining these frames in various ways by simply overlapping and adding the windowed pitch period frames (we will introduce the idea of windows in Sec. **??**). The idea that we modify a signal by extracting frames, manipulating them in some way and then recombining them by adding up the overlapped signals is called the **overlap-and-add** or **OLA** algorithm; TD-PSOLA is a special case of overlap-and-add in which the frames are pitch-synchronous, and the whole process takes place in the time domain.

For example, in order to assign a specific duration to a diphone, we might want to lengthen the recorded master diphone. To lengthen a signal with TD-PSOLA, we simply insert extra copies of some of the pitch-synchronous frames, essentially duplicating a piece of the signal. Fig. 8.16 shows the intuition.

TD-PSOLA can also be used to change the F0 value of a recorded diphone to give a higher or lower value. To increase the F0, we extract each pitch-synchronous frame from the original recorded diphone signal, place the frames closer together (overlapping them), with the amount of overlap determined by the desired period and hence frequency, and then add up the overlapping signals to produce the final signal. But note that by moving all the frames closer together, we make the signal shorter in time! Thus in order to change the pitch while holding the duration constant, we need to add duplicate frames.

Fig. 8.17 shows the intuition; in this figure we have explicitly shown the extracted pitch-synchronous frames which are overlapped and added; note that the frames moved closer together (increasing the pitch) while extra frames have been added to hold the duration constant.

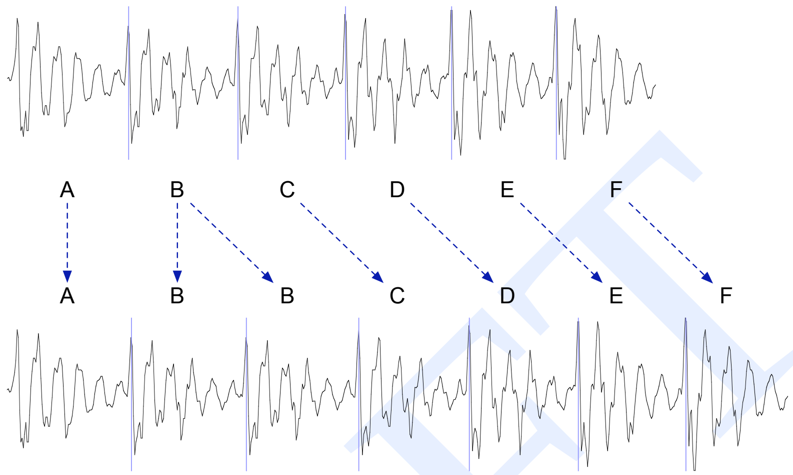


Figure 8.16 TD-PSOLA for duration modification. Individual pitch-synchronous frames can be duplicated to lengthen the signal (as shown here), or deleted to shorten the signal.

8.5 Unit Selection (Waveform) Synthesis

Diphone waveform synthesis suffers from two main problems. First, the stored diphone database must be modified by signal process methods like PSOLA to produce the desired prosody. Any kind of signal processing of the stored speech leaves artifacts in the speech which can make the speech sound unnatural. Second, diphone synthesis only captures the coarticulation due to a single neighboring phone. But there are many more global effects on phonetic realization, including more distant phones, syllable structure, the stress patterns of nearby phones, and even word-level effects.

UNIT SELECTION SYNTHESIS

For this reason, modern commercial synthesizers are based on a generalization of diphone synthesis called **unit selection synthesis**. Like diphone synthesis, unit selection synthesis is a kind of concatenative synthesis algorithm. It differs from classic diphone synthesis in two ways:

1. In diphone synthesis the database stores exactly one copy of each diphone, while in unit selection, the unit database is many hours long, containing many copies of each diphone.
2. In diphone synthesis, the prosody of the concatenated units is modified by PSOLA or similar algorithms, while in unit selection no (or minimal) signal processing is applied to the concatenated units.

The strengths of unit selection are due to the large unit database. In a sufficiently large database, entire words or phrases of the utterance we want to synthesize may be already present in the database, resulting in an extremely natural waveform for these

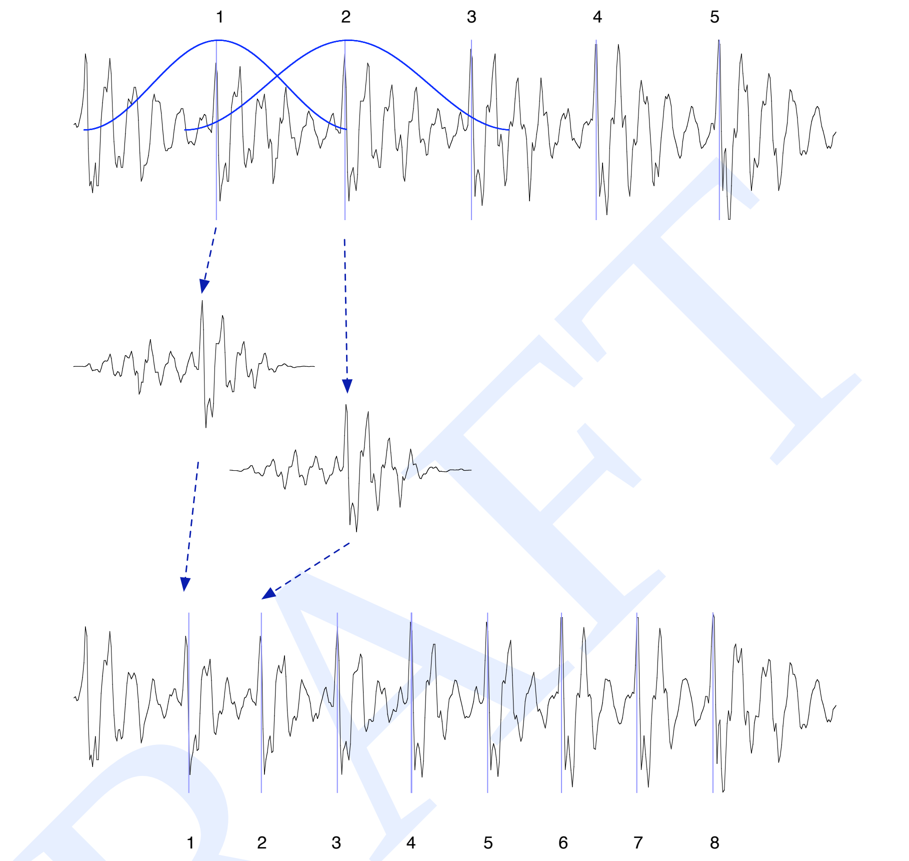


Figure 8.17 TD-PSOLA for pitch (F0) modification. In order to increase the pitch, the individual pitch-synchronous frames are extracted, Hanning windowed, moved closer together and then added up. To decrease the pitch, we move the frames further apart. Increasing the pitch will result in a shorter signal (since the frames are closer together), so we also need to duplicate frames if we want to change the pitch while holding the duration constant.

words or phrases. In addition, in cases where we can’t find a large chunk and have to back off to individual diphones, the fact that there are so many copies of each diphone makes it more likely that we will find one that will fit in very naturally.

The architecture of unit selection can be summarized as follows. We are given a large database of units; let’s assume these are diphones (although it’s also possible to do unit selection with other kinds of units such half-phones, syllables, or half-syllables). We are also given a characterization of the target ‘internal representation’, i.e. a phone string together with features such as stress values, word identity, F0 information, as described in Fig. 8.1.

The goal of the synthesizer is to select from the database the best sequence of diphone units that corresponds to the target representation. What do we mean by the 'best' sequence? Intuitively, the best sequence would be one in which:

- each diphone unit we select exactly meets the specifications of the target diphone (in terms of F0, stress level, phonetic neighbors, etc)
- each diphone unit concatenates smoothly with its neighboring units, with no perceptible break.

Of course, in practice, we can't guarantee that there wil be a unit which exactly meets our specifications, and we are unlikely to find a sequence of units in which every single join is imperceptible. Thus in practice unit selection algorithms implement a gradient version of these constraints, and attempt to find the sequence of unit which at least minimizes these two costs:

TARGET COST **target cost** $T(u_t, s_t)$: how well the target specification s_t matches the potential unit u_t

JOIN COST **join cost** $J(u_t, u_{t+1})$: how well (perceptually) the potential unit u_t joins with its potential neighbor u_{t+1}

The T and J values are expressed as **costs** meaning that high values indicate bad matches and bad joins (Hunt and Black, 1996a).

Formally, then, the task of unit selection synthesis, given a sequence S of T target specifications, is to find the sequence $\hat{U}$ of T units from the database which minimizes the sum of these costs:

$$\hat{U} = \underset{U}{\operatorname{argmin}} \sum_{t=1}^{T} T(s_t, u_t) + \sum_{t=1}^{T-1} J(u_t, u_{t+1}) \tag{8.21}$$

Let's now define the target cost and the join cost in more detail before we turn to the decoding and training tasks.

The target cost measures how well the unit matches the target diphone specification. We can think of the specification for each diphone target as a feature vector; here are three sample vectors for three target diphone specifications, using dimensions (features) like *should the syllable be stressed*, and *where in the intonational phrase should the diphone come from*:

```
/ih-t/, +stress, phrase internal, high F0, content word
/n-t/, -stress, phrase final, high F0, function word
/dh-ax/, -stress, phrase initial, low F0, word 'the'
```

We'd like the distance between the target specification s and the unit to be some function of the how different the unit is on each of these dimensions from the specification. Let's assume that for each dimension p, we can come up with some **subcost** $T_p(s_t[p], u_j[p])$. The subcost for a binary feature like *stress* might be 1 or 0. The subcost for a continuous feature like F0 might be the difference (or log difference) between the specification F0 and unit F0. Since some dimensions are more important to speech perceptions than others, we'll also want to weight each dimension. The simplest way to combine all these subcosts is just to assume that they are independent and additive. Using this model, the total target cost for a given target/unit pair is the weighted sum over all these subcosts for each feature/dimension:

$$T(s_t,u_j) = \sum_{p=1}^{P} w_p T_p(s_t[p],u_j[p]) \tag{8.22}$$

The target cost is a function of the desired diphone specification and a unit from the database. The **join cost**, by contrast, is a function of two units from the database. The goal of the join cost is to be low (0) when the join is completely natural, and high when the join would be perceptible or jarring. We do this by measuring the acoustic similarity of the edges of the two units that we will be joining. If the two units have very similar energy, F0, and spectral features, they will probably join well. Thus as with the target cost, we compute a join cost by summing weighted subcosts:

$$J(u_t,u_{t+1}) = \sum_{p=1}^{P} w_p J_p(u_t[p],u_{t+1}[p]) \tag{8.23}$$

The three subcosts used in the classic Hunt and Black (1996b) algorithm are the **cepstral distance** at the point of concatenation, and the absolute differences in log power and F0. We will introduce the cepstrum in Sec. **??**.

In addition, if the two units u_t and u_{t+1} to be concatenated were consecutive diphones in the unit database (i.e. they followed each other in the original utterance), then we set the join cost to 0: $J(u_t,u_{t+1}) = 0$. This is an important feature of unit selection synthesis, since it encourages large natural sequences of units to be selected from the database.

How do we find the best sequence of units which minimizes the sum of the target and join costs as expressed in Eq. 8.21? The standard method is to think of the unit selection problem as a Hidden Markov Model. The target units are the observed outputs, and the units in the database are the hidden states. Our job is to find the best hidden state sequence. We will use the Viterbi algorithm to solve this problem, just as we saw it in Ch. 5 and Ch. 6, and will see it again in Ch. 9. Fig. 8.18 shows a sketch of the search space as well as the best (Viterbi) path that determines the best unit sequence.

The weights for join and target costs are often set by hand, since the number of weights is small (on the order of 20) and machine learning algorithms don't always achieve human performance. The system designer listens to entire sentences produced by the system, and chooses values for weights that result in reasonable sounding utterances. Various automatic weight-setting algorithms do exist, however. Many of these assume we have some sort of distance function between the acoustics of two sentences, perhaps based on cepstral distance. The method of Hunt and Black (1996b), for example, holds out a test set of sentences from the unit selection database. For each of these test sentences, we take the word sequence and synthesize a sentence waveform (using units from the other sentences in the training database). Now we compare the acoustics of the synthesized sentence with the acoustics of the true human sentence. Now we have a sequence of synthesized sentences, each one associated with a distance function to its human counterpart. Now we use linear regression based on these distances to set the target cost weights so as to minimize the distance.

There are also more advanced methods of assigning both target and join costs. For example, above we computed target costs between two units by looking at the features

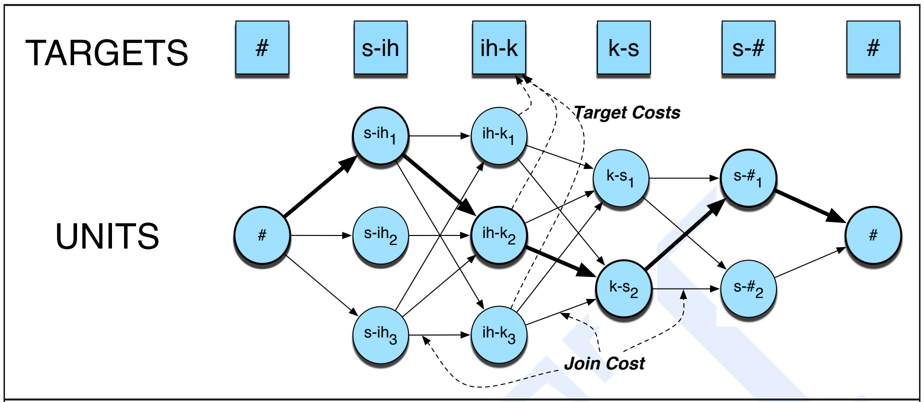


Figure 8.18 The process of decoding in unit selection. The figure shows the sequence of target (specification) diphones for the word *six*, and the set of possible database diphone units that we must search through. The best (Viterbi) path that minimizes the sum of the target and join costs is shown in bold.

of the two units, doing a weighted sum of feature costs, and choosing the lowest-cost unit. An alternative approach (which the new reader might need to come back to after learning the speech recognition techniques introduced in the next chapters) is to map the target unit into some acoustic space, and then find a unit which is near the target in that acoustic space. In the method of Donovan and Eide (1998), Donovan and Woodland (1995), for example, all the training units are clustered using the decision tree algorithm of speech recognition described in Sec. **??**. The decision tree is based on the same features described above, but here for each set of features, we follow a path down the decision tree to a leaf node which contains a cluster of units that have those features. This cluster of units can be parameterized by a Gaussian model, just as for speech recognition, so that we can map a set of features into a probability distribution over cepstral values, and hence easily compute a distance between the target and a unit in the database. As for join costs, more sophisticated metrics make use of how perceivable a particular join might be (Wouters and Macon, 1998; Syrdal and Conkie, 2004; Bulyko and Ostendorf, 2001).

8.6 Evaluation

Speech synthesis systems are evaluated by human listeners. The development of a good automatic metric for synthesis evaluation, that would eliminate the need for expensive and time-consuming human listening experiments, remains an open and exiting research topic.

INTELLIGIBILITY

The minimal evaluation metric for speech synthesis systems is **intelligibility**: the ability of a human listener to correctly interpret the words and meaning of the synthesized utterance. A further metric is **quality**; an abstract measure of the naturalness, fluency, or clarity of the speech.

QUALITY

DIAGNOSTIC RHYME TEST
DRT
MODIFIED RHYME TEST
MRT

The most local measures of intelligibility test the ability of a listener to discriminate between two phones. The **Diagnostic Rhyme Test** (**DRT**) (Voiers et al., 1975) tests the intelligibility of initial consonants. It is based on 96 pairs of confusable rhyming words which differ only in a single phonetic feature, such as (*dense/tense*) or *bond/pond* (differing in voicing) or *mean/beat* or *neck/deck* (differing in nasality), and so on. For each pair, listeners hear one member of the pair, and indicate which they think it is. The percentage of right answers is then used as an intelligibility metric. The **Modified Rhyme Test** (**MRT**) (House et al., 1965) is a similar test based on a different set of 300 words, consisting of 50 sets of 6 words. Each 6-word set differs in either initial or final consonants (e.g., *went*, *sent*, *bent*, *dent*, *tent*, *rent* or *bat*, *bad*, *back*, *bass*, *ban*, *bath*). Listeners are again given a single word and must identify from a closed list of six words; the percentage of correct identifications is again used as an intelligibility metric.

CARRIER PHRASES

Since context effects are very important, both DRT and MRT words are embedded in **carrier phrases** like the following:

```
Now we will say <word> again.
```

SUS

In order to test larger units than single phones, we can use **semantically unpredictable sentences** (**SUS**) (Benoît et al., 1996). These are sentences constructed by taking a simple POS template like DET ADJ NOUN VERB DET NOUN and inserting random English words in the slots, to produce sentences like

```
The unsure steaks closed the fish.
```

Measures of intelligibility like DRT/MRT and SUS are designed to factor out the role of context in measuring intelligibility. While this allows us to get a carefully controlled measure of a system's intelligibility, such acontextual or semantically unpredictable sentences aren't a good fit to how TTS is used in most commercial applications. Thus in commercial applications instead of DRT or SUS, we generally test intelligibility using situations that mimic the desired applications; reading addresses out loud, reading lines of news text, and so on.

MOS

To further evaluate the **quality** of the synthesized utterances, we can play a sentence for a listener and ask them to give a **mean opinion score** (**MOS**), a rating of how good the synthesized utterances are, usually on a scale from 1-5. We can then compare systems by comparing their MOS scores on the same sentences (using, e.g., t-tests to test for significant differences).

AB TESTS

If we are comparing exactly two systems (perhaps to see if a particular change actually improved the system), we can use **AB tests** In AB tests, we play the same sentence synthesized by two different systems (an A and a B system). The human listener chooses which of the two utterances they like better. We can do this for 50 sentences and compare the number of sentences preferred for each systems. In order to avoid ordering preferences, for each sentence we must present the two synthesized waveforms in random order.

Bibliographical and Historical Notes

As we noted at the beginning of the chapter, speech synthesis is one of the earliest fields of speech and language processing. The 18th century saw a number of physical models of the articulation process, including the von Kempelen model mentioned above, as well as the 1773 vowel model of Kratzenstein in Copenhagen using organ pipes.

But the modern era of speech synthesis can clearly be said to have arrived by the early 1950's, when all three of the major paradigms of waveform synthesis had been proposed (formant synthesis, articulatory synthesis, and concatenative synthesis).

Concatenative synthesis seems to have been first proposed by Harris (1953) at Bell Laboratories, who literally spliced together pieces of magnetic tape corresponding to phones. Harris's proposal was actually more like unit selection synthesis than diphone synthesis, in that he proposed storing multiple copies of each phone, and proposed the use of a join cost (choosing the unit with the smoothest formant transitions with the neighboring unit). Harris's model was based on the phone, rather than diphone, resulting in problems due to coarticulation. Peterson et al. (1958) added many of the basic ideas of unit selection synthesis, including the use of diphones, a database with multiple copies of each diphone with differing prosody, and each unit labeled with intonational features including F0, stress, and duration, and the use of join costs based on F0 and formant distant between neighboring units. They also proposed microconcatenation techniques like windowing the waveforms. The Peterson et al. (1958) model was purely theoretical, however, and concatenative synthesis was not implemented until the 1960's and 1970's, when diphone synthesis was first implemented (Dixon and Maxey, 1968; Olive, 1977). Later diphone systems included larger units such as consonant clusters (Olive and Liberman, 1979). Modern unit selection, including the idea of large units of non-uniform length, and the use of a target cost, was invented by Sagisaka (1988), Sagisaka et al. (1992). Hunt and Black (1996b) formalized the model, and put it in the form in which we have presented it in this chapter in the context of the ATR CHATR system (Black and Taylor, 1994). The idea of automatically generating synthesis units by clustering was first invented by Nakajima and Hamada (1988), but was developed mainly by (Donovan, 1996) by incorporating decision tree clustering algorithms from speech recognition. Many unit selection innovations took place as part of the ATT NextGen synthesizer (Syrdal et al., 2000; Syrdal and Conkie, 2004).

We have focused in this chapter on concatenative synthesis, but there are two other paradigms for synthesis: **formant synthesis**, in which we attempt to build rules which generate artificial spectra, including especially formants, and **articulatory synthesis**, in which we attempt to directly model the physics of the vocal tract and articulatory process.

Formant synthesizers originally were inspired by attempts to mimic human speech by generating artificial spectrograms. The Haskins Laboratories Pattern Playback Machine generated a sound wave by painting spectrogram patterns on a moving transparent belt, and using reflectance to filter the harmonics of a waveform (Cooper et al., 1951); other very early formant synthesizers include Lawrence (1953) and Fant (3951). Perhaps the most well-known of the formant synthesizers were the **Klatt formant syn-**

thesizer and its successor systems, including the MITalk system (Allen et al., 1987), and the Klattalk software used in Digital Equipment Corporation's DECtalk (Klatt, 1982). See Klatt (1975) for details.

Articulatory synthesizers attempt to synthesize speech by modeling the physics of the vocal tract as an open tube. Representative models, both early and somewhat more recent include Stevens et al. (1953), Flanagan et al. (1975), Fant (1986) See Klatt (1975) and Flanagan (1972) for more details.

Development of the text analysis components of TTS came somewhat later, as techniques were borrowed from other areas of natural language processing. The input to early synthesis systems was not text, but rather phonemes (typed in on punched cards). The first text-to-speech system to take text as input seems to have been the system of Umeda and Teranishi (Umeda et al., 1968; Teranishi and Umeda, 1968; Umeda, 1976). The system included a lexicalized parser which was used to assign prosodic boundaries, as well as accent and stress; the extensions in Coker et al. (1973) added additional rules, for example for deaccenting light verbs and explored articulatory models as well. These early TTS systems used a pronunciation dictionary for word pronunciations. In order to expand to larger vocabularies, early formant-based TTS systems such as MITlak (Allen et al., 1987) used letter-to-sound rules instead of a dictionary, since computer memory was far too expensive to store large dictionaries.

GRAPHONE

Modern grapheme-to-phoneme models derive from the influential early probabilistic grapheme-to-phoneme model of Lucassen and Mercer (1984), which was originally proposed in the context of speech recognition. The widespread use of such machine learning models was delayed, however, because early anecdotal evidence suggested that hand-written rules worked better than e.g., the neural networks of Sejnowski and Rosenberg (1987). The careful comparisons of Damper et al. (1999) showed that machine learning methods were in generally superior. A number of such models make use of pronunciation by analogy (Byrd and Chodorow, 1985; ?; Daelemans and van den Bosch, 1997; Marchand and Damper, 2000) or latent analogy (Bellegarda, 2005); HMMs (Taylor, 2005) have also been proposed. The most recent work makes use of joint **graphone** models, in which the hidden variables are phoneme-grapheme pairs and the probabilistic model is based on joint rather than conditional likelihood (Deligne et al., 1995; Luk and Damper, 1996; Galescu and Allen, 2001; Bisani and Ney, 2002; Chen, 2003).

FUJISAKI

INTONATION UNITS

TONE UNITS

There is a vast literature on prosody. Besides the ToBI and TILT models described above, other important computational models include the **Fujisaki** model (Fujisaki and Ohno, 1997). IViE (Grabe, 2001) is an extension of ToBI that focuses on labelling different varieties of English (Grabe et al., 2000). There is also much debate on the units of intonational structure (**intonational phrases** (Beckman and Pierrehumbert, 1986), **intonation units** (Du Bois et al., 1983) or **tone units** (Crystal, 1969)), and their relation to clauses and other syntactic units (Chomsky and Halle, 1968; Langendoen, 1975; Streeter, 1978; Hirschberg and Pierrehumbert, 1986; Selkirk, 1986; Nespor and Vogel, 1986; Croft, 1995; Ladd, 1996; Ford and Thompson, 1996; Ford et al., 1996).

HMM SYNTHESIS

One of the most exciting new paradigms for speech synthesis is **HMM synthesis**, first proposed by Tokuda et al. (1995b) and elaborated in Tokuda et al. (1995a), Tokuda et al. (2000), and Tokuda et al. (2003). See also the textbook summary of HMM synthesis in Taylor (2008).

More details on TTS evaluation can be found in Huang et al. (2001) and Gibbon et al. (2000). Other descriptions of evaluation can be found in the annual speech synthesis competition called the **Blizzard Challenge** (Black and Tokuda, 2005; Bennett, 2005).

BLIZZARD CHALLENGE

Much recent work on speech synthesis has focused on generating emotional speech (Cahn, 1990; Bulut1 et al., 2002; Hamza et al., 2004; Eide et al., 2004; Lee et al., 2006; Schroder, 2006, inter alia)

Two classic text-to-speech synthesis systems are described in Allen et al. (1987) (the *MITalk* system) and Sproat (1998b) (the Bell Labs system). Recent textbooks include Dutoit (1997), Huang et al. (2001), Taylor (2008), and Alan Black's online lecture notes at `http://festvox.org/festtut/notes/festtut_toc.html`. Influential collections of papers include van Santen et al. (1997), Sagisaka et al. (1997), Narayanan and Alwan (2004). Conference publications appear in the main speech engineering conferences (INTERSPEECH, *IEEE ICASSP*), and the *Speech Synthesis Workshops*. Journals include *Speech Communication*, *Computer Speech and Language*, the *IEEE Transactions on Audio, Speech, and Language Processing*, and the *ACM Transactions on Speech and Language Processing*.

Exercises

8.1 Implement the text normalization routine that deals with MONEY, i.e. mapping strings of dollar amounts like *$45*, *$320*, and *$4100* to words (either writing code directly or designing an FST). If there are multiple ways to pronounce a number you may pick your favorite way.

8.2 Implement the text normalization routine that deals with NTEL, i.e. seven-digit phone numbers like *555-1212*, *555-1300*, and so on. You should use a combination of the **paired** and **trailing unit** methods of pronunciation for the last four digits. (Again you may either write code or design an FST).

8.3 Implement the text normalization routine that deals with type DATE in Fig. 8.4

8.4 Implement the text normalization routine that deals with type NTIME in Fig. 8.4.

8.5 (Suggested by Alan Black). Download the free Festival speech synthesizer. Augment the lexicon to correctly pronounce the names of everyone in your class.

8.6 Download the Festival synthesizer. Record and train a diphone synthesizer using your own voice.

Allen, J., Hunnicut, M. S., and Klatt, D. H. (1987). *From Text to Speech: The MITalk system*. Cambridge University Press.

Anderson, M. J., Pierrehumbert, J. B., and Liberman, M. Y. (1984). Improving intonational phrasing with syntactic information. In *IEEE ICASSP-84*, pp. 2.8.1–2.8.4.

Bachenko, J. and Fitzpatrick, E. (1990). A computational grammar of discourse-neutral prosodic phrasing in English. *Computational Linguistics*, *16*(3), 155–170.

Beckman, M. E. and Ayers, G. M. (1997). Guidelines for ToBI labelling..

Beckman, M. E. and Hirschberg, J. (1994). The tobi annotation conventions. Manuscript, Ohio State University.

Beckman, M. E. and Pierrehumbert, J. B. (1986). Intonational structure in English and Japanese. *Phonology Yearbook*, *3*, 255–310.

Bellegarda, J. R. (2005). Unsupervised, language-independent grapheme-to-phoneme conversion by latent analogy. *Speech Communication*, *46*(2), 140–152.

Bennett, C. (2005). Large scale evaluation of corpus-based synthesizers: Results and lessons from the blizzard challenge 2005. In *EUROSPEECH-05*.

Benoît, C., Grice, M., and Hazan, V. (1996). The SUS test: A method for the assessment of text-to-speech synthesis intelligibility using Semantically Unpredictable Sentences. *Speech Communication*, *18*(4), 381–392.

Bisani, M. and Ney, H. (2002). Investigations on joint-multigram models for grapheme-to-phoneme conversion. In *ICSLP-02*, Vol. 1, pp. 105–108.

Black, A. W. and Taylor, P. (1994). CHATR: a generic speech synthesis system. In *COLING-94*, Kyoto, Vol. II, pp. 983–986.

Black, A. W. and Hunt, A. J. (1996). Generating F0 contours from ToBI labels using linear regression. In *ICSLP-96*, Vol. 3, pp. 1385–1388.

Black, A. W., Lenzo, K., and Pagel, V. (1998). Issues in building general letter to sound rules. In *3rd ESCA Workshop on Speech Synthesis, Jenolan Caves, Australia*.

Black, A. W., Taylor, P., and Caley, R. (1996-1999). The Festival Speech Synthesis System system. Manual and source code available at `www.cstr.ed.ac.uk/projects/festival.html`.

Black, A. W. and Tokuda, K. (2005). The Blizzard Challenge–2005: Evaluating corpus-based speech synthesis on common datasets. In *EUROSPEECH-05*.

Bolinger, D. (1972). Accent is predictable (if you're a mind-reader). *Language*, *48*(3), 633–644.

Brookes, D. M. and Loke, H. P. (1999). Modelling energy flow in the vocal tract with applications to glottal closure and opening detection. In *IEEE ICASSP-99*, pp. 213–216.

Bulut1, M., Narayanan, S. S., and Syrdal, A. K. (2002). Expressive speech synthesis using a concatenative synthesizer. In *ICSLP-02*.

Bulyko, I. and Ostendorf, M. (2001). Unit selection for speech synthesis using splicing costs with weighted finite state transducers. In *EUROSPEECH-01*, Vol. 2, pp. 987–990.

Byrd, R. J. and Chodorow, M. S. (1985). Using an On-Line dictionary to find rhyming words and pronunciations for unknown words. In *ACL-85*, pp. 277–283.

Cahn, J. E. (1990). The generation of affect in synthesized speech. In *Journal of the American Voice I/O Society*, Vol. 8, pp. 1–19.

Chen, S. F. (2003). Conditional and joint models for grapheme-to-phoneme conversion. In *EUROSPEECH-03*.

Chomsky, N. and Halle, M. (1968). *The Sound Pattern of English*. Harper and Row.

CMU (1993). The Carnegie Mellon Pronouncing Dictionary v0.1. Carnegie Mellon University.

Coker, C., Umeda, N., and Browman, C. (1973). Automatic synthesis from ordinary english test. *IEEE Transactions on Audio and Electroacoustics*, *21*(3), 293–298.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *ACL/EACL-97*, Madrid, Spain, pp. 16–23.

Conkie, A. and Isard, S. (1996). Optimal coupling of diphones. In van Santen, J. P. H., Sproat, R., Olive, J. P., and Hirschberg, J. (Eds.), *Progress in Speech Synthesis*. Springer.

Cooper, F. S., Liberman, A. M., and Borst, J. M. (1951). The Interconversion of Audible and Visible Patterns as a Basis for Research in the Perception of Speech. *Proceedings of the National Academy of Sciences*, *37*(5), 318–325.

Croft, W. (1995). Intonation units and grammatical structure. *Linguistics*, *33*, 839–882.

Crystal, D. (1969). *Prosodic systems and intonation in English*. Cambridge University Press.

Daelemans, W. and van den Bosch, A. (1997). Language-independent data-oriented grapheme-to-phoneme conversion. In van Santen, J. P. H., Sproat, R., Olive, J. P., and Hirschberg, J. (Eds.), *Progress in Speech Synthesis*, pp. 77–89. Springer.

Damper, R. I., Marchand, Y., Adamson, M. J., and Gustafson, K. (1999). Evaluating the pronunciation component of text-to-speech systems for english: A performance comparison of different approaches. *Computer Speech and Language*, *13*(2), 155–176.

Deligne, S., Yvon, F., and Bimbot, F. (1995). Variable-length sequence matching for phonetic transcription using joint multigrams. In *EUROSPEECH-95*, Madrid.

Demberg, V. (2006). Letter-to-phoneme conversion for a german text-to-speech system. Diplomarbeit Nr. 47, Universiẗ Stuttgart.

Divay, M. and Vitale, A. J. (1997). Algorithms for grapheme-phoneme translation for English and French: Applications for database searches and speech synthesis. *Computational Linguistics*, *23*(4), 495–523.

Dixon, N. and Maxey, H. (1968). Terminal analog synthesis of continuous speech using the diphone method of segment assembly. *IEEE Transactions on Audio and Electroacoustics*, *16*(1), 40–50.

Donovan, R. E. (1996). *Trainable Speech Synthesis*. Ph.D. thesis, Cambridge University Engineering Department.

Donovan, R. E. and Eide, E. M. (1998). The IBM trainable speech synthesis system. In *ICSLP-98*, Sydney.

Donovan, R. E. and Woodland, P. C. (1995). Improvements in an HMM-based speech synthesiser. In *EUROSPEECH-95*, Madrid, Vol. 1, pp. 573–576.

Du Bois, J. W., Schuetze-Coburn, S., Cumming, S., and Paolino, D. (1983). Outline of discourse transcription. In Edwards, J. A. and Lampert, M. D. (Eds.), *Talking Data: Transcription and Coding in Discourse Research*, pp. 45–89. Lawrence Erlbaum.

Dutoit, T. (1997). *An Introduction to Text to Speech Synthesis*. Kluwer.

Eide, E. M., Bakis, R., Hamza, W., and Pitrelli, J. F. (2004). Towards synthesizing expressive speech. In Narayanan, S. S. and Alwan, A. (Eds.), *Text to Speech Synthesis: New paradigms and Advances*. Prentice Hall.

Fackrell, J. and Skut, W. (2004). Improving pronunciation dictionary coverage of names by modelling spelling variation. In *Proceedings of the 5th Speech Synthesis Workshop*.

Fant, C. G. M. (3951). Speech communication research. *Ing. Vetenskaps Akad. Stockholm, Sweden*, *24*, 331–337.

Fant, G. M. (1986). Glottal flow: Models and interaction. *Journal of Phonetics*, *14*, 393–399.

Fitt, S. (2002). Unisyn lexicon. `http://www.cstr.ed.ac.uk/projects/unisyn/`.

Flanagan, J. L. (1972). *Speech Analysis, Synthesis, and Perception*. Springer.

Flanagan, J. L., Ishizaka, K., and Shipley, K. L. (1975). Synthesis of speech from a dynamic model of the vocal cords and vocal tract. *The Bell System Technical Journal*, *54*(3), 485–506.

Ford, C., Fox, B., and Thompson, S. A. (1996). Practices in the construction of turns. *Pragmatics*, *6*, 427–454.

Ford, C. and Thompson, S. A. (1996). Interactional units in conversation: syntactic, intonational, and pragmatic resources for the management of turns. In Ochs, E., Schegloff, E. A., and Thompson, S. A. (Eds.), *Interaction and Grammar*, pp. 134–184. Cambridge University Press.

Fujisaki, H. and Ohno, S. (1997). Comparison and assessment of models in the study of fundamental frequency contours of speech. In *ESCA workshop on Intonation: Theory Models and Applications*.

Galescu, L. and Allen, J. (2001). Bi-directional conversion between graphemes and phonemes using a joint N-gram model. In *Proceedings of the 4th ISCA Tutorial and Research Workshop on Speech Synthesis*.

Gee, J. P. and Grosjean, F. (1983). Performance structures: A psycholinguistic and linguistic appraisal. *Cognitive Psychology*, *15*, 411–458.

Gibbon, D., Mertins, I., and Moore, R. (2000). *Handbook of Multimodal and Spoken Dialogue Systems: Resources, Terminology and Product Evaluation*. Kluwer, Dordrecht.

Grabe, E., Post, B., Nolan, F., and Farrar, K. (2000). Pitch accent realisation in four varieties of British English. *Journal of Phonetics*, *28*, 161–186.

Grabe, E. (2001). The ivie labelling guide..

Gregory, M. and Altun, Y. (2004). Using conditional random fields to predict pitch accents in conversational speech. In *ACL-04*.

Grosjean, F., Grosjean, L., and Lane, H. (1979). The patterns of silence: Performance structures in sentence production. *Cognitive Psychology*, *11*, 58–81.

Hamza, W., Bakis, R., Eide, E. M., Picheny, M. A., and Pitrelli, J. F. (2004). The IBM expressive speech synthesis system. In *ICSLP-04*, Jeju, Korea.

Harris, C. M. (1953). A study of the building blocks in speech. *Journal of the Acoustical Society of America*, *25*(5), 962–969.

Hirschberg, J. (1993). Pitch Accent in Context: Predicting Intonational Prominence from Text. *Artificial Intelligence*, *63*(1-2), 305–340.

Hirschberg, J. and Pierrehumbert, J. B. (1986). The intonational structuring of discourse. In *ACL-86*, New York, pp. 136–144.

House, A. S., Williams, C. E., Hecker, M. H. L., and Kryter, K. D. (1965). Articulation-Testing Methods: Consonantal Differentiation with a Closed-Response Set. *Journal of the Acoustical Society of America*, *37*, 158–166.

Huang, X., Acero, A., and Hon, H.-W. (2001). *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall, Upper Saddle River, NJ.

Hunt, A. J. and Black, A. W. (1996a). Unit selection in a concatenative speech synthesis system using a large speech database. In *IEEE ICASSP-96*, Atlanta, GA, Vol. 1, pp. 373–376. IEEE.

Hunt, A. J. and Black, A. W. (1996b). Unit selection in a concatenative speech synthesis system using a large speech database. In *IEEE ICASSP-06*, Vol. 1, pp. 373–376.

Jilka, M., Mohler, G., and Dogil, G. (1999). Rules for the generation of ToBI-based American English intonation. *Speech Communication*, *28*(2), 83–108.

Jun, S.-A. (Ed.). (2005). *Prosodic Typology and Transcription: A Unified Approach*. Oxford University Press.

Klatt, D. H. (1975). Voice onset time, friction, and aspiration in word-initial consonant clusters. *Journal of Speech and Hearing Research*, *18*, 686–706.

Klatt, D. H. (1982). The Klattalk text-to-speech conversion system. In *IEEE ICASSP-82*, pp. 1589–1592.

Klatt, D. H. (1979). Synthesis by rule of segmental durations in English sentences. In Lindblom, B. E. F. and Öhman, S. (Eds.), *Frontiers of Speech Communication Research*, pp. 287–299. Academic.

Klimt, B. and Yang, Y. (2004). The Enron corpus: A new dataset for email classification research. In *Proceedings of the European Conference on Machine Learning*, pp. 217–226. Springer.

Koehn, P., Abney, S. P., Hirschberg, J., and Collins, M. (2000). Improving intonational phrasing with syntactic information. In *IEEE ICASSP-00*.

Ladd, D. R. (1996). *Intonational Phonology*. Cambridge Studies in Linguistics. Cambridge University Press.

Langendoen, D. T. (1975). Finite-state parsing of phrase-structure languages and the status of readjustment rules in the grammar. *Linguistic Inquiry*, *6*(4), 533–554.

Lawrence, W. (1953). The synthesis of speech from signals which have a low information rate.. In Jackson, W. (Ed.), *Communication Theory*, pp. 460–469. Butterworth.

Lee, S., Bresch, E., Adams, J., Kazemzadeh, A., and Narayanan, S. S. (2006). A study of emotional speech articulation using a fast magnetic resonance imaging technique. In *ICSLP-06*.

Liberman, M. Y. and Church, K. W. (1992). Text analysis and word pronunciation in text-to-speech synthesis. In Furui, S. and Sondhi, M. M. (Eds.), *Advances in Speech Signal Processing*, pp. 791–832. Marcel Dekker.

Liberman, M. Y. and Prince, A. (1977). On stress and linguistic rhythm. *Linguistic Inquiry*, *8*, 249–336.

Liberman, M. Y. and Sproat, R. (1992). The stress and structure of modified noun phrases in English. In Sag, I. A. and Szabolcsi, A. (Eds.), *Lexical Matters*, pp. 131–181. CSLI, Stanford University.

Lucassen, J. and Mercer, R. L. (1984). An information theoretic approach to the automatic determination of phonemic baseforms. In *IEEE ICASSP-84*, Vol. 9, pp. 304–307.

Luk, R. W. P. and Damper, R. I. (1996). Stochastic phonographic transduction for english. *Computer Speech and Language*, *10*(2), 133–153.

Marchand, Y. and Damper, R. I. (2000). A multi-strategy approach to improving pronunciation by analogy. *Computational Linguistics*, *26*(2), 195–219.

Nakajima, S. and Hamada, H. (1988). Automatic generation of synthesis units based on context oriented clustering. In *IEEE ICASSP-88*, pp. 659–662.

Narayanan, S. S. and Alwan, A. (Eds.). (2004). *Text to Speech Synthesis: New paradigms and advances*. Prentice Hall.

Nenkova, A., Brenier, J., Kothari, A., Calhoun, S., Whitton, L., Beaver, D., and Jurafsky, D. (2007). To memorize or to predict: Prominence labeling in conversational speech. In *NAACL-HLT 07*.

Nespor, M. and Vogel, I. (1986). *Prosodic phonology*. Foris, Dordrecht.

Olive, J. and Liberman, M. (1979). A set of concatenative units for speech synthesis. *Journal of the Acoustical Society of America*, *65*, S130.

Olive, J. P. (1977). Rule synthesis of speech from dyadic units. In *ICASSP77*, pp. 568–570.

Olive, J. P., van Santen, J. P. H., Möbius, B., and Shih, C. (1998). Synthesis. In Sproat, R. (Ed.), *Multilingual Text-To-Speech Synthesis: The Bell Labs Approach*, pp. 191–228. Kluwer, Dordrecht.

Ostendorf, M. and Veilleux, N. (1994). A hierarchical stochastic model for automatic prediction of prosodic boundary location. *Computational Linguistics*, *20*(1).

Pan, S. and Hirschberg, J. (2000). Modeling local context for pitch accent prediction. In *ACL-00*, Hong Kong, pp. 233–240.

Pan, S. and McKeown, K. R. (1999). Word informativeness and automatic pitch accent modeling. In *EMNLP/VLC-99*.

Peterson, G. E., Wang, W. W.-Y., and Sivertsen, E. (1958). Segmentation techniques in speech synthesis. *Journal of the Acoustical Society of America*, *30*(8), 739–742.

Pierrehumbert, J. B. (1980). *The Phonology and Phonetics of English Intonation*. Ph.D. thesis, MIT.

Pitrelli, J. F., Beckman, M. E., and Hirschberg, J. (1994). Evaluation of prosodic transcription labeling reliability in the ToBI framework. In *ICSLP-94*, Vol. 1, pp. 123–126.

Price, P. J., Ostendorf, M., Shattuck-Hufnagel, S., and Fong, C. (1991). The use of prosody in syntactic disambiguation. *Journal of the Acoustical Society of America*, *90*(6).

Riley, M. D. (1992). Tree-based modelling for speech synthesis. In Bailly, G. and Beniot, C. (Eds.), *Talking Machines: Theories, Models and Designs*. North Holland, Amsterdam.

Sagisaka, Y. (1988). Speech synthesis by rule using an optimal selection of non-uniform synthesis units. In *IEEE ICASSP-88*, pp. 679–682.

Sagisaka, Y., Kaiki, N., Iwahashi, N., and Mimura, K. (1992). Atr – ν-talk speech synthesis system. In *ICSLP-92*, Banff, Canada, pp. 483–486.

Sagisaka, Y., Campbell, N., and Higuchi, N. (Eds.). (1997). *Computing Prosody: Computational Models for Processing Spontaneous Speech*. Springer.

Schroder, M. (2006). Expressing degree of activation in synthetic speech. *IEEE Transactions on Audio, Speech, and Language Processing*, *14*(4), 1128–1136.

Sejnowski, T. J. and Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, *1*(1), 145–168.

Selkirk, E. (1986). On derived domains in sentence phonology. *Phonology Yearbook*, *3*, 371–405.

Silverman, K., Beckman, M. E., Pitrelli, J. F., Ostendorf, M., Wightman, C., Price, P. J., Pierrehumbert, J. B., and Hirschberg, J. (1992). ToBI: a standard for labelling English prosody. In *ICSLP-92*, Vol. 2, pp. 867–870.

Spiegel, M. F. (2002). Proper name pronunciations for speech technology applications. In *Proceedings of IEEE Workshop on Speech Synthesis*, pp. 175–178.

Spiegel, M. F. (2003). Proper name pronunciations for speech technology applications. *International Journal of Speech Technology*, *6*(4), 419–427.

Sproat, R. (1994). English noun-phrase prediction for text-to-speech. *Computer Speech and Language*, *8*, 79–94.

Sproat, R. (1998a). Further issues in text analysis. In Sproat, R. (Ed.), *Multilingual Text-To-Speech Synthesis: The Bell Labs Approach*, pp. 89–114. Kluwer, Dordrecht.

Sproat, R. (Ed.). (1998b). *Multilingual Text-To-Speech Synthesis: The Bell Labs Approach*. Kluwer, Dordrecht.

Sproat, R., Black, A. W., Chen, S. F., Kumar, S., Ostendorf, M., and Richards, C. (2001). Normalization of non-standard words. *Computer Speech & Language*, *15*(3), 287–333.

Steedman, M. (2003). Information-structural semantics for English intonation..

Stevens, K. N., Kasowski, S., and Fant, G. M. (1953). An electrical analog of the vocal tract. *Journal of the Acoustical Society of America*, *25*(4), 734–742.

Streeter, L. (1978). Acoustic determinants of phrase boundary perception. *Journal of the Acoustical Society of America*, *63*, 1582–1592.

Syrdal, A. K. and Conkie, A. (2004). Data-driven perceptually based join costs. In *Proceedings of Fifth ISCA Speech Synthesis Workshop*.

Syrdal, A. K., Wightman, C. W., Conkie, A., Stylianou, Y., Beutnagel, M., Schroeter, J., Strom, V., and Lee, K.-S. (2000). Corpus-based techniques in the AT&T NEXTGEN synthesis system. In *ICSLP-00*, Beijing.

Taylor, P. (2000). Analysis and synthesis of intonation using the Tilt model. *Journal of the Acoustical Society of America*, *107*(3), 1697–1714.

Taylor, P. (2005). Hidden Markov Models for grapheme to phoneme conversion. In *INTERSPEECH-05*, Lisbon, Portugal, pp. 1973–1976.

Taylor, P. (2008). Text-to-speech synthesis. Manuscript.

Taylor, P. and Black, A. W. (1998). Assigning phrase breaks from part of speech sequences. *Computer Speech and Language*, *12*, 99–117.

Taylor, P. and Isard, S. (1991). Automatic diphone segmentation. In *EUROSPEECH-91*, Genova, Italy.

Teranishi, R. and Umeda, N. (1968). Use of pronouncing dictionary in speech synthesis experiments. In *6th International Congress on Acoustics*, Tokyo, Japan, pp. B155–158. †.

Tokuda, K., Kobayashi, T., and Imai, S. (1995a). Speech parameter generation from hmm using dynamic features. In *IEEE ICASSP-95*.

Tokuda, K., Masuko, T., and Yamada, T. (1995b). An algorithm for speech parameter generation from continuous mixture hmms with dynamic features. In *EUROSPEECH-95*, Madrid.

Tokuda, K., Yoshimura, T., Masuko, T., Kobayashi, T., and Kitamura, T. (2000). Speech parameter generation algorithms for hmm-based speech synthesis. In *IEEE ICASSP-00*.

Tokuda, K., Zen, H., and Kitamura, T. (2003). Trajectory modeling based on hmms with the explicit relationship between static and dynamic features. In *EUROSPEECH-03*.

Umeda, N., Matui, E., Suzuki, T., and Omura, H. (1968). Synthesis of fairy tale using an analog vocal tract. In *6th International Congress on Acoustics*, Tokyo, Japan, pp. B159–162. †.

Umeda, N. (1976). Linguistic rules for text-to-speech synthesis. *Proceedings of the IEEE*, *64*(4), 443–451.

van Santen, J. P. H. (1994). Assignment of segmental duration in text-to-speech synthesis. *Computer Speech and Language*, *8*(95–128).

van Santen, J. P. H. (1997). Segmental duration and speech timing. In Sagisaka, Y., Campbell, N., and Higuchi, N. (Eds.), *Computing Prosody: Computational Models for Processing Spontaneous Speech*. Springer.

van Santen, J. P. H. (1998). Timing. In Sproat, R. (Ed.), *Multilingual Text-To-Speech Synthesis: The Bell Labs Approach*, pp. 115–140. Kluwer, Dordrecht.

van Santen, J. P. H., Sproat, R., Olive, J. P., and Hirschberg, J. (Eds.). (1997). *Progress in Speech Synthesis*. Springer.

Veldhuis, R. (2000). Consistent pitch marking. In *ICSLP-00*, Beijing, China.

Venditti, J. J. (2005). The j_tobi model of japanese intonation. In Jun, S.-A. (Ed.), *Prosodic Typology and Transcription: A Unified Approach*. Oxford University Press.

Voiers, W., Sharpley, A., and Hehmsoth, C. (1975). Research on diagnostic evaluation of speech intelligibility. Research Report AFCRL-72-0694.

Wang, M. Q. and Hirschberg, J. (1992). Automatic classification of intonational phrasing boundaries. *Computer Speech and Language*, *6*(2), 175–196.

Wouters, J. and Macon, M. (1998). Perceptual evaluation of distance measures for concatenative speech synthesis. In *ICSLP-98*, Sydney, pp. 2747–2750.

Yarowsky, D. (1997). Homograph disambiguation in text-to-speech synthesis. In van Santen, J. P. H., Sproat, R., Olive, J. P., and Hirschberg, J. (Eds.), *Progress in Speech Synthesis*, pp. 157–172. Springer.

Yuan, J., Brenier, J. M., and Jurafsky, D. (2005). Pitch accent prediction: Effects of genre and speaker. In *EUROSPEECH-05*.

9 AUTOMATIC SPEECH RECOGNITION

When Frederic was a little lad he proved so brave and daring,
His father thought he'd 'prentice him to some career seafaring.
I was, alas! his nurs'rymaid, and so it fell to my lot
To take and bind the promising boy apprentice to a **pilot** —
A life not bad for a hardy lad, though surely not a high lot,
Though I'm a nurse, you might do worse than make your boy a pilot.
I was a stupid nurs'rymaid, on breakers always steering,
And I did not catch the word aright, through being hard of hearing;
Mistaking my instructions, which within my brain did gyrate,
I took and bound this promising boy apprentice to a **pirate.**
The Pirates of Penzance, Gilbert and Sullivan, 1877

Alas, this mistake by nurserymaid Ruth led to Frederic's long indenture as a pirate and, due to a slight complication involving 21st birthdays and leap years, nearly led to 63 extra years of apprenticeship. The mistake was quite natural, in a Gilbert-and-Sullivan sort of way; as Ruth later noted, "The two words were so much alike!" True, true; spoken language understanding is a difficult task, and it is remarkable that humans do as well at it as we do. The goal of **automatic speech recognition** (**ASR**) research is to address this problem computationally by building systems that map from an acoustic signal to a string of words. **Automatic speech understanding** (**ASU**) extends this goal to producing some sort of understanding of the sentence, rather than just the words.

ASR

The general problem of automatic transcription of speech by any speaker in any environment is still far from solved. But recent years have seen ASR technology mature to the point where it is viable in certain limited domains. One major application area is in human-computer interaction. While many tasks are better solved with visual or pointing interfaces, speech has the potential to be a better interface than the keyboard for tasks where full natural language communication is useful, or for which keyboards are not appropriate. This includes hands-busy or eyes-busy applications, such as where the user has objects to manipulate or equipment to control. Another important application area is telephony, where speech recognition is already used for example in spoken dialogue systems for entering digits, recognizing "yes" to accept collect calls, finding out airplane or train information, and call-routing ("Accounting, please", "Prof. Regier, please"). In some applications, a multimodal interface combining speech and pointing can be more efficient than a graphical user interface without speech (Cohen et al., 1998). Finally, ASR is applied to dictation, that is, transcription of extended monologue by a single specific speaker. Dictation is common in fields such as law and is also important as part of augmentative communication (interaction between computers and humans with some disability resulting in the inability to type, or the inability to speak). The blind Milton famously dictated *Paradise Lost* to his daughters, and Henry James dictated his later novels after a repetitive stress injury.

DIGITS TASK

Before turning to architectural details, let's discuss some of the parameters and the state of the art of the speech recognition task. One dimension of variation in speech recognition tasks is the vocabulary size. Speech recognition is easier if the number of distinct words we need to recognize is smaller. So tasks with a two word vocabulary, like *yes* versus *no* detection, or an eleven word vocabulary, like recognizing sequences of digits, in what is called the **digits task** task, are relatively easy. On the other end, tasks with large vocabularies, like transcribing human-human telephone conversations, or transcribing broadcast news, tasks with vocabularies of 64,000 words or more, are much harder.

ISOLATED WORD

CONTINUOUS SPEECH

READ SPEECH

CONVERSATIONAL SPEECH

A second dimension of variation is how fluent, natural, or conversational the speech is. **Isolated word** recognition, in which each word is surrounded by some sort of pause, is much easier than recognizing **continuous speech**, in which words run into each other and have to be segmented. Continuous speech tasks themselves vary greatly in difficulty. For example, human-to-machine speech turns out to be far easier to recognize than human-to-human speech. That is, recognizing speech of humans talking to machines, either reading out loud in **read speech** (which simulates the dictation task), or conversing with speech dialogue systems, is relatively easy. Recognizing the speech of two humans talking to each other, in **conversational speech** recognition, for example for transcribing a business meeting or a telephone conversation, is much harder. It seems that when humans talk to machines, they simplify their speech quite a bit, talking more slowly and more clearly.

A third dimension of variation is channel and noise. Commercial dictation systems, and much laboratory research in speech recognition, is done with high quality, head mounted microphones. Head mounted microphones eliminate the distortion that occurs in a table microphone as the speakers head moves around. Noise of any kind also makes recognition harder. Thus recognizing a speaker dictating in a quiet office is much easier than recognizing a speaker dictating in a noisy car on the highway with the window open.

A final dimension of variation is accent or speaker-class characteristics. Speech is easier to recognize if the speaker is speaking a standard dialect, or in general one that matches the data the system was trained on. Recognition is thus harder on foreign-accented speech, or speech of children (unless the system was specifically trained on exactly these kinds of speech).

Table 9.1 shows the rough percentage of incorrect words (the **word error rate**, or WER, defined on page 45) from state-of-the-art systems on a range of different ASR tasks.

Variation due to noise and accent increases the error rates quite a bit. The word error rate on strongly Japanese-accented or Spanish accented English has been reported to be about 3 to 4 times higher than for native speakers on the same task (Tomokiyo, 2001). And adding automobile noise with a 10dB SNR (signal-to-noise ratio) can cause error rates to go up by 2 to 4 times.

Task	Vocabulary	Error Rate %
TI Digits	11 (zero-nine, oh)	.5
Wall Street Journal read speech	5,000	3
Wall Street Journal read speech	20,000	3
Broadcast News	64,000+	10
Conversational Telephone Speech (CTS)	64,000+	20

Figure 9.1 Rough word error rates (% of words misrecognized) reported around 2006 for ASR on various tasks; the error rates for Broadcast News and CTS are based on particular training and test scenarios and should be taken as ballpark numbers; error rates for differently defined tasks may range up to a factor of two.

In general, these error rates go down every year, as speech recognition performance has improved quite steadily. One estimate is that performance has improved roughly 10 percent a year over the last decade (Deng and Huang, 2004), due to a combination of algorithmic improvements and Moore's law.

LVCSR

SPEAKER-INDEPENDENT

While the algorithms we describe in this chapter are applicable across a wide variety of these speech tasks, we chose to focus this chapter on the fundamentals of one crucial area: **Large-Vocabulary Continuous Speech Recognition** (**LVCSR**). Large-vocabulary generally means that the systems have a vocabulary of roughly 20,000 to 60,000 words. We saw above that **continuous** means that the words are run together naturally. Furthermore, the algorithms we will discuss are generally **speaker-independent**; that is, they are able to recognize speech from people whose speech the system has never been exposed to before.

The dominant paradigm for LVCSR is the HMM, and we will focus on this approach in this chapter. Previous chapters have introduced most of the core algorithms used in HMM-based speech recognition. Ch. 7 introduced the key phonetic and phonological notions of **phone**, **syllable**, and intonation. Ch. 5 and Ch. 6 introduced the use of the **Bayes rule**, the **Hidden Markov Model** (**HMM**), the **Viterbi** algorithm, and the Baum-Welch training algorithm. Ch. 4 introduced the *N*-**gram** language model and the **perplexity** metric. In this chapter we begin with an overview of the architecture for HMM speech recognition, offer an all-too-brief overview of signal processing for feature extraction and the extraction of the important MFCC features, and then introduce Gaussian acoustic models. We then continue with how Viterbi decoding works in the ASR context, and give a complete summary of the training procedure for ASR, called **embedded training**. Finally, we introduce word error rate, the standard evaluation metric. The next chapter will continue with some advanced ASR topics.

9.1 Speech Recognition Architecture

NOISY CHANNEL

The task of speech recognition is to take as input an acoustic waveform and produce as output a string of words. HMM-based speech recognition systems view this task using the metaphor of the noisy channel. The intuition of the **noisy channel** model (see Fig. 9.2) is to treat the acoustic waveform as an "noisy" version of the string of words, i.e.. a version that has been passed through a noisy communications channel.

This channel introduces “noise” which makes it hard to recognize the “true” string of words. Our goal is then to build a model of the channel so that we can figure out how it modified this “true” sentence and hence recover it.

The insight of the noisy channel model is that if we know how the channel distorts the source, we could find the correct source sentence for a waveform by taking every possible sentence in the language, running each sentence through our noisy channel model, and seeing if it matches the output. We then select the best matching source sentence as our desired source sentence.

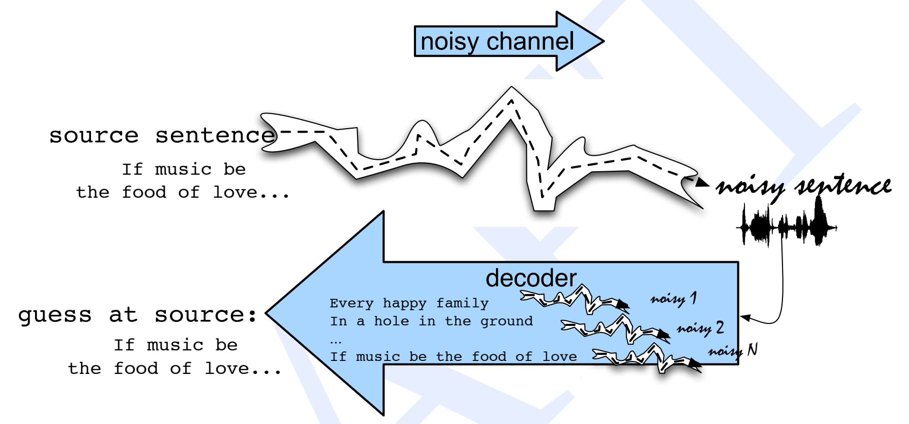


Figure 9.2 The noisy channel model. We search through a huge space of potential “source” sentences and choose the one which has the highest probability of generating the “noisy” sentence. We need models of the prior probability of a source sentence (*N*-grams), the probability of words being realized as certain strings of phones (HMM lexicons), and the probability of phones being realized as acoustic or spectral features (Gaussian Mixture Models).

Implementing the noisy-channel model as we have expressed it in Fig. 9.2 requires solutions to two problems. First, in order to pick the sentence that best matches the noisy input we will need a complete metric for a “best match”. Because speech is so variable, an acoustic input sentence will never exactly match any model we have for this sentence. As we have suggested in previous chapters, we will use probability as our

BAYESIAN

metric. This makes the speech recognition problem a special case of **Bayesian inference**, a method known since the work of Bayes (1763). Bayesian inference or Bayesian classification was applied successfully by the 1950s to language problems like optical character recognition (Bledsoe and Browning, 1959) and to authorship attribution tasks like the seminal work of Mosteller and Wallace (1964) on determining the authorship of the Federalist papers. Our goal will be to combine various probabilistic models to get a complete estimate for the probability of a noisy acoustic observation-sequence given a candidate source sentence. We can then search through the space of all sentences, and choose the source sentence with the highest probability.

Second, since the set of all English sentences is huge, we need an efficient algorithm that will not search through all possible sentences, but only ones that have a good chance of matching the input. This is the **decoding** or **search** problem, which we have already explored with the Viterbi decoding algorithm for HMMs in Ch. 5 and Ch. 6. Since the search space is so large in speech recognition, efficient search is an important part of the task, and we will focus on a number of areas in search.

In the rest of this introduction we will review the probabilistic or Bayesian model for speech recognition that we introduced for part-of-speech tagging in Ch. 5. We then introduce the various components of a modern HMM-based ASR system.

Recall that the goal of the probabilistic noisy channel architecture for speech recognition can be summarized as follows:

> *"What is the most likely sentence out of all sentences in the language $\mathscr{L}$ given some acoustic input O?"*

We can treat the acoustic input O as a sequence of individual "symbols" or "observations" (for example by slicing up the input every 10 milliseconds, and representing each slice by floating-point values of the energy or frequencies of that slice). Each index then represents some time interval, and successive o_i indicate temporally consecutive slices of the input (note that capital letters will stand for sequences of symbols and lower-case letters for individual symbols):

$$O = o_1, o_2, o_3, \ldots, o_t \tag{9.1}$$

Similarly, we treat a sentence as if it were composed of a string of words:

$$W = w_1, w_2, w_3, \ldots, w_n \tag{9.2}$$

Both of these are simplifying assumptions; for example dividing sentences into words is sometimes too fine a division (we'd like to model facts about groups of words rather than individual words) and sometimes too gross a division (we need to deal with morphology). Usually in speech recognition a word is defined by orthography (after mapping every word to lower-case): *oak* is treated as a different word than *oaks*, but the auxiliary *can* ("can you tell me...") is treated as the same word as the noun *can* ("i need a can of...").

The probabilistic implementation of our intuition above, then, can be expressed as follows:

$$\hat{W} = \operatorname*{argmax}_{W \in \mathscr{L}} P(W|O) \tag{9.3}$$

Recall that the function $\text{argmax}_x f(x)$ means "the x such that f(x) is largest". Equation (9.3) is guaranteed to give us the optimal sentence W; we now need to make the equation operational. That is, for a given sentence W and acoustic sequence O we need to compute $P(W|O)$. Recall that given any probability $P(x|y)$, we can use Bayes' rule to break it down as follows:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{9.4}$$

We saw in Ch. 5 that we can substitute (9.4) into (9.3) as follows:

$$\hat{W} = \underset{W \in \mathscr{L}}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)} \tag{9.5}$$

The probabilities on the right-hand side of (9.5) are for the most part easier to compute than $P(W|O)$. For example, $P(W)$, the prior probability of the word string itself is exactly what is estimated by the n-gram language models of Ch. 4. And we will see below that $P(O|W)$ turns out to be easy to estimate as well. But $P(O)$, the probability of the acoustic observation sequence, turns out to be harder to estimate. Luckily, we can ignore $P(O)$ just as we saw in Ch. 5. Why? Since we are maximizing over all possible sentences, we will be computing $\frac{P(O|W)P(W)}{P(O)}$ for each sentence in the language. But $P(O)$ doesn't change for each sentence! For each potential sentence we are still examining the same observations O, which must have the same probability $P(O)$. Thus:

$$\hat{W} = \underset{W \in \mathscr{L}}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)} = \underset{W \in \mathscr{L}}{\operatorname{argmax}} P(O|W)\,P(W) \tag{9.6}$$

To summarize, the most probable sentence W given some observation sequence O can be computed by taking the product of two probabilities for each sentence, and choosing the sentence for which this product is greatest. The general components of the speech recognizer which compute these two terms have names; $P(W)$, the **prior probability**, is computed by the **language model**. while $P(O|W)$, the **observation likelihood**, is computed by the **acoustic model**.

LANGUAGE MODEL

ACOUSTIC MODEL

$$\hat{W} = \underset{W \in \mathscr{L}}{\operatorname{argmax}} \ \overbrace{P(O|W)}^{\text{likelihood}} \ \overbrace{P(W)}^{\text{prior}} \tag{9.7}$$

The language model (LM) prior $P(W)$ expresses how likely a given string of words is to be a source sentence of English. We have already seen in Ch. 4 how to compute such a language model prior $P(W)$ by using N-gram grammars. Recall that an N-gram grammar lets us assign a probability to a sentence by computing:

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-N+1}^{k-1}) \tag{9.8}$$

This chapter will show how the HMM we covered in Ch. 6 can be used to build an Acoustic Model (AM) which computes the likelihood $P(O|W)$. Given the AM and LM probabilities, the probabilistic model can be operationalized in a search algorithm so as to compute the maximum probability word string for a given acoustic waveform. Fig. 9.3 shows the components of an HMM speech recognizer as it processes a single utterance, indicating the computation of the prior and likelihood. The figure shows the recognition process in three stages. In the **feature extraction** or **signal processing** stage, the acoustic waveform is sampled into **frames** (usually of 10, 15, or 20 milliseconds) which are transformed into **spectral features**. Each time window is thus represented by a vector of around 39 features representing this spectral information as well as information about energy and spectral change. Sec. 9.3 gives an (unfortunately brief) overview of the feature extraction process.

In the **acoustic modeling** or **phone recognition** stage, we compute the likelihood of the observed spectral feature vectors given linguistic units (words, phones, subparts of phones). For example, we use Gaussian Mixture Model (GMM) classifiers to compute for each HMM state q, corresponding to a phone or subphone, the likelihood of a given feature vector given this phone $p(o|q)$. A (simplified) way of thinking of the output of this stage is as a sequence of probability vectors, one for each time frame, each vector at each time frame containing the likelihoods that each phone or subphone unit generated the acoustic feature vector observation at that time.

Finally, in the **decoding** phase, we take the acoustic model (AM), which consists of this sequence of acoustic likelihoods, plus an HMM dictionary of word pronunciations, combined with the language model (LM) (generally an N-gram grammar), and output the most likely sequence of words. An HMM dictionary, as we will see in Sec. 9.2, is a list of word pronunciations, each pronunciation represented by a string of phones. Each word can then be thought of as an HMM, where the phones (or sometimes subphones) are states in the HMM, and the Gaussian likelihood estimators supply the HMM output likelihood function for each state. Most ASR systems use the Viterbi algorithm for decoding, speeding up the decoding with wide variety of sophisticated augmentations such as pruning, fast-match, and tree-structured lexicons.

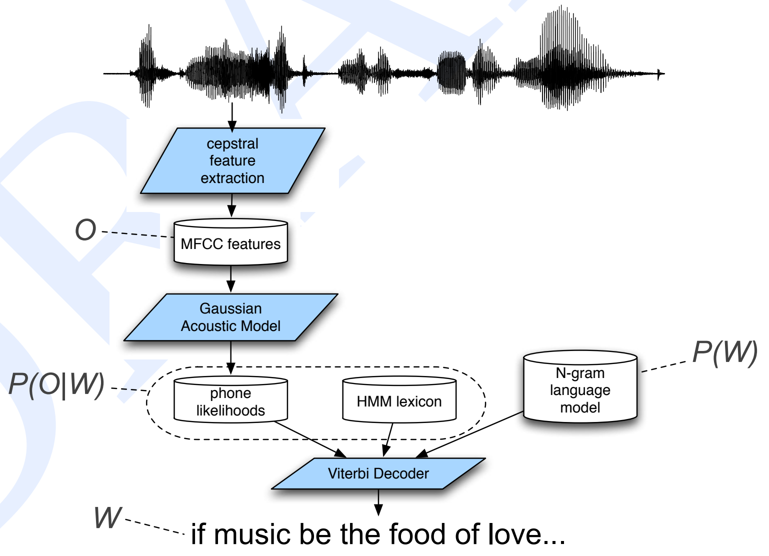


Figure 9.3 Schematic architecture for a (simplified) speech recognizer decoding a single sentence. A real recognizer is more complex since various kinds of pruning and fast matches are needed for efficiency. This architecture is only for decoding; we also need a separate architecture for training parameters.

9.2 Applying the Hidden Markov Model to Speech

Let's turn now to how the HMM model is applied to speech recognition. We saw in Ch. 6 that a Hidden Markov Model is characterized by the following components:

$Q = q_1 q_2 \ldots q_N$	a set of **states**
$A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_N$	a set of **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$.
$B = b_i(o_t)$	A set of **observation likelihoods:**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i.
q_0, q_{end}	a special **start and end state** which are not associated with observations.

Furthermore, the chapter introduced the **Viterbi** algorithm for decoding HMMs, and the **Baum-Welch** or **Forward-Backward** algorithm for training HMMs.

All of these facets of the HMM paradigm play a crucial role in ASR. We begin here by discussing how the states, transitions, and observations map into the speech recognition task. We will return to the ASR applications of Viterbi decoding in Sec. 9.6. The extensions to the Baum-Welch algorithms needed to deal with spoken language are covered in Sec. 9.4 and Sec. 9.7.

Recall the examples of HMMs we saw earlier in the book. In Ch. 5, the hidden states of the HMM were parts-of-speech, the observations were words, and the HMM decoding task mapped a sequence of words to a sequence of parts-of-speech. In Ch. 6, the hidden states of the HMM were weather, the observations were 'ice-cream consumptions', and the decoding task was to determine the weather sequence from a sequence of ice-cream consumption. For speech, the hidden states are phones, parts of phones, or words, each observation is information about the spectrum and energy of the waveform at a point in time, and the decoding process maps this sequence of acoustic information to phones and words.

The observation sequence for speech recognition is a sequence of **acoustic feature vectors**. Each acoustic feature vector represents information such as the amount of energy in different frequency bands at a particular point in time. We will return in Sec. 9.3 to the nature of these observations, but for now we'll simply note that each observation consists of a vector of 39 real-valued features indicating spectral information. Observations are generally drawn every 10 milliseconds, so 1 second of speech requires 100 spectral feature vectors, each vector of length 39.

DIGIT RECOGNITION

The hidden states of Hidden Markov Models can be used to model speech in a number of different ways. For small tasks, like **digit recognition**, (the recognition of the 10 digit words *zero* through *nine*), or for **yes-no** recognition (recognition of the two words **yes** and **no**), we could build an HMM whose states correspond to entire words.

For most larger tasks, however, the hidden states of the HMM correspond to phone-like units, and words are sequences of these phone-like units.

Let's begin by describing an HMM model in which each state of an HMM corresponds to a single phone (if you've forgotten what a phone is, go back and look again at the definition in Ch. 7). In such a model, a word HMM thus consists of a sequence of HMM states concatenated together. Fig. 9.4 shows a schematic of the structure of a basic phone-state HMM for the word *six*.

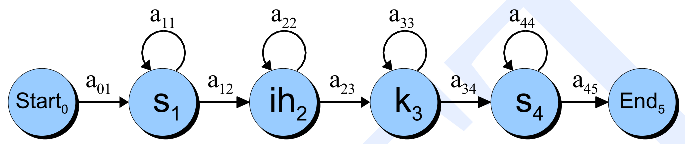


Figure 9.4 An HMM for the word *six*, consisting of four emitting states and two non-emitting states, the transition probabilities A, the observation probabilities *B*, and a sample observation sequence.

Note that only certain connections between phones exist in Fig. 9.4. In the HMMs described in Ch. 6, there were arbitrary transitions between states; any state could transition to any other. This was also in principle true of the HMMs for part-of-speech tagging in Ch. 5; although the probability of some tag transitions was low, any tag could in principle follow any other tag. Unlike in these other HMM applications, HMM models for speech recognition usually do not allow arbitrary transitions. Instead, they place strong constraints on transitions based on the sequential nature of speech. Except in unusual cases, HMMs for speech don't allow transitions from states to go to earlier states in the word; in other words, states can transition to themselves or to successive states. As we saw in Ch. 6, this kind of **left-to-right** HMM structure is called a **Bakis network**.

BAKIS NETWORK

The most common model used for speech, illustrated in a simplified form in Fig. 9.4 is even more constrained, allowing a state to transition only to itself (self-loop) or to a single succeeding state. The use of self-loops allows a single phone to repeat so as to cover a variable amount of the acoustic input. Phone durations vary hugely, dependent on the phone identify, the speaker's rate of speech, the phonetic context, and the level of prosodic prominence of the word. Looking at the Switchboard corpus, the phone [aa] varies in length from 7 to 387 milliseconds (1 to 40 frames), while the phone [z] varies in duration from 7 milliseconds to more than 1.3 seconds (130 frames) in some utterances! Self-loops thus allow a single state to be repeated many times.

For very simple speech tasks (recognizing small numbers of words such as the 10 digits), using an HMM state to represent a phone is sufficient. In general LVCSR tasks, however, a more fine-grained representation is necessary. This is because phones can last over 1 second, i.e., over 100 frames, but the 100 frames are not acoustically identical. The spectral characteristics of a phone, and the amount of energy, vary dramatically across a phone. For example, recall from Ch. 7 that stop consonants have a closure portion, which has very little acoustic energy, followed by a release burst. Similarly, diphthongs are vowels whose F1 and F2 change significantly. Fig. 9.5 shows these large changes in spectral characteristics over time for each of the two phones in the word “Ike”, ARPAbet [ay k].

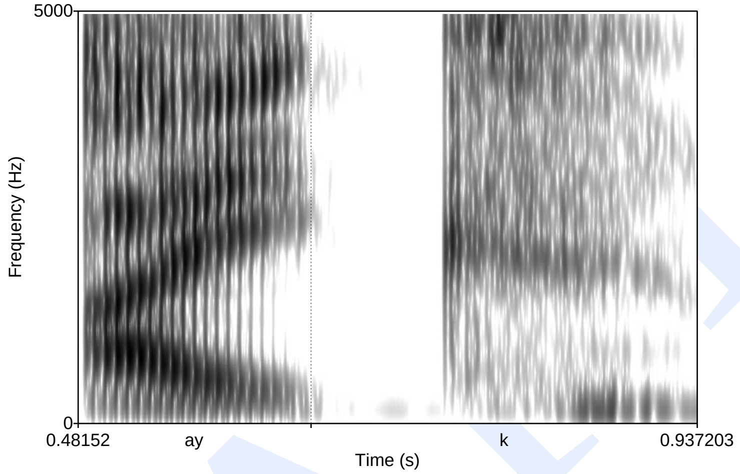


Figure 9.5 The two phones of the word ”Ike”, pronounced [ay k]. Note the continuous changes in the [ay] vowel on the left, as F2 rises and F1 falls, and the sharp differences between the silence and release parts of the [k] stop.

To capture this fact about the non-homogeneous nature of phones over time, in LVCSR we generally model a phone with more than one HMM state. The most common configuration is to use three HMM states, a beginning, middle, and end state. Each phone thus consists of 3 emitting HMM states instead of one (plus two non-emitting states at either end), as shown in Fig. 9.6. It is common to reserve the word **model** or **phone model** to refer to the entire 5-state phone HMM, and use the word **HMM state** (or just **state** for short) to refer to each of the 3 individual subphone HMM states.

MODEL
PHONE MODEL
HMM STATE

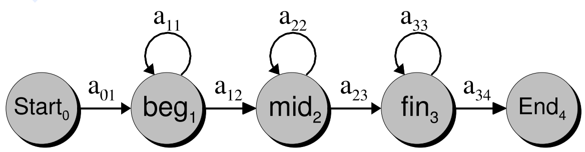


Figure 9.6 A standard 5-state HMM model for a phone, consisting of three emitting states (corresponding to the transition-in, steady state, and transition-out regions of the phone) and two non-emitting states.

To build a HMM for an entire word using these more complex phone models, we can simply replace each phone of the word model in Fig. 9.4 with a 3-state phone HMM. We replace the non-emitting start and end states for each phone model with transitions directly to the emitting state of the preceding and following phone, leaving only two non-emitting states for the entire word. Fig. 9.7 shows the expanded word.

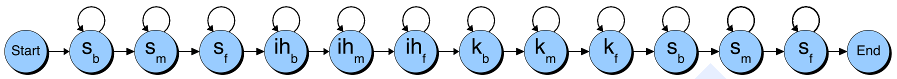


Figure 9.7 A composite word model for "six", [s ih k s], formed by concatenating four phone models, each with three emitting states.

In summary, an HMM model of speech recognition is parameterized by:

$Q = q_1 q_2 \ldots q_N$ a set of **states** corresponding to **subphones**

$A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn}$ a **transition probability matrix** A, each a_{ij} representing the probability for each subphone of taking a **self-loop** or going to the next subphone.

$B = b_i(o_t)$ A set of **observation likelihoods:**, also called **emission probabilities**, each expressing the probability of a cepstral feature vector (observation o_t) being generated from subphone state i.

Another way of looking at the A probabilities and the states Q is that together they represent a **lexicon**: a set of pronunciations for words, each pronunciation consisting of a set of subphones, with the order of the subphones specified by the transition probabilities A.

We have now covered the basic structure of HMM states for representing phones and words in speech recognition. Later in this chapter we will see further augmentations of the HMM word model shown in Fig. 9.7, such as the use of triphone models which make use of phone context, and the use of special phones to model silence. First, though, we need to turn to the next component of HMMs for speech recognition: the observation likelihoods. And in order to discuss observation likelihoods, we first need to introduce the actual acoustic observations: feature vectors. After discussing these in Sec. 9.3, we turn in Sec. 9.4 the acoustic model and details of observation likelihood computation. We then re-introduce Viterbi decoding and show how the acoustic model and language model are combined to choose the best sentence.

9.3 FEATURE EXTRACTION: MFCC VECTORS

FEATURE VECTORS

MFCC
MEL FREQUENCY CEPSTRAL COEFFICIENTS
CEPSTRUM

Our goal in this section is to describe how we transform the input waveform into a sequence of acoustic **feature vectors**, each vector representing the information in a small time window of the signal. While there are many possible such feature representations, by far the most common in speech recognition is the **MFCC**, the **mel frequency cepstral coefficients**. These are based on the important idea of the **cepstrum**. We will give a relatively high-level description of the process of extraction of MFCCs from a

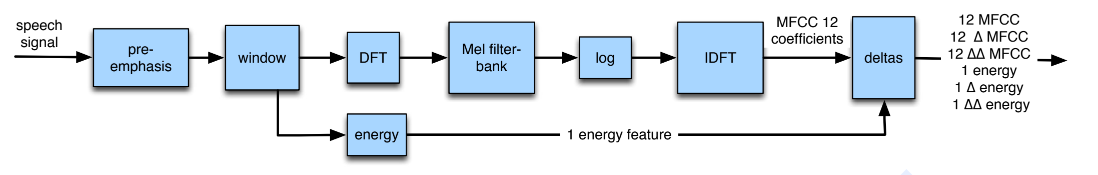


Figure 9.8 Extracting a sequence of 39-dimensional MFCC feature vectors from a quantized digitized waveform

waveform; we strongly encourage students interested in more detail to follow up with a speech signal processing course.

SAMPLING

SAMPLING RATE

NYQUIST FREQUENCY

TELEPHONE-BANDWIDTH

WIDEBAND

We begin by repeating from Sec. **??** the process of digitizing and quantizing an analog speech waveform. Recall that the first step in processing speech is to convert the analog representations (first air pressure, and then analog electric signals in a microphone), into a digital signal. This process of **analog-to-digital conversion** has two steps: **sampling** and **quantization**. A signal is sampled by measuring its amplitude at a particular time; the **sampling rate** is the number of samples taken per second. In order to accurately measure a wave, it is necessary to have at least two samples in each cycle: one measuring the positive part of the wave and one measuring the negative part. More than two samples per cycle increases the amplitude accuracy, but less than two samples will cause the frequency of the wave to be completely missed. Thus the maximum frequency wave that can be measured is one whose frequency is half the sample rate (since every cycle needs two samples). This maximum frequency for a given sampling rate is called the **Nyquist frequency**. Most information in human speech is in frequencies below 10,000 Hz; thus a 20,000 Hz sampling rate would be necessary for complete accuracy. But telephone speech is filtered by the switching network, and only frequencies less than 4,000 Hz are transmitted by telephones. Thus an 8,000 Hz sampling rate is sufficient for **telephone-bandwidth** speech like the Switchboard corpus. A 16,000 Hz sampling rate (sometimes called **wideband**) is often used for microphone speech.

QUANTIZATION

Even an 8,000 Hz sampling rate requires 8000 amplitude measurements for each second of speech, and so it is important to store the amplitude measurement efficiently. They are usually stored as integers, either 8-bit (values from -128–127) or 16 bit (values from -32768–32767). This process of representing real-valued numbers as integers is called **quantization** because there is a minimum granularity (the quantum size) and all values which are closer together than this quantum size are represented identically.

We refer to each sample in the digitized quantized waveform as $x[n]$, where n is an index over time. Now that we have a digitized, quantized representation of the waveform, we are ready to extract MFCC features. The seven steps of this process are shown in Fig. 9.8 and individually described in each of the following sections.

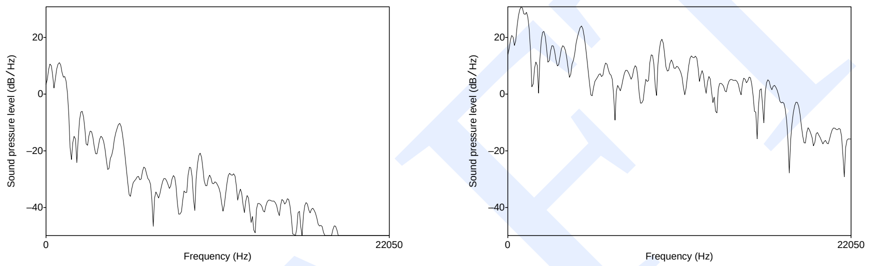
9.3.1 Preemphasis

The first stage in MFCC feature extraction is to boost the amount of energy in the high frequencies. It turns out that if we look at the spectrum for voiced segments like

SPECTRAL TILT

vowels, there is more energy at the lower frequencies than the higher frequencies. This drop in energy across frequencies (which is called **spectral tilt**) is caused by the nature of the glottal pulse. Boosting the high frequency energy makes information from these higher formants more available to the acoustic model and improves phone detection accuracy.

This preemphasis is done by using a filter[1] Fig. 9.9 shows an example of a spectral slice from the first author's pronunciation of the single vowel [aa] before and after preemphasis.



(a) (b)

Figure 9.9 A spectral slice from the vowel [aa] before (a) and after (b) preemphasis.

9.3.2 Windowing

NON-STATIONARY

STATIONARY

Recall that the goal of feature extraction is to provide spectral features that can help us build phone or subphone classifiers. We therefore don't want to extract our spectral features from an entire utterance or conversation, because the spectrum changes very quickly. Technically, we say that speech is a **non-stationary** signal, meaning that its statistical properties are not constant across time. Instead, we want to extract spectral features from a small **window** of speech that characterizes a particular subphone and for which we can make the (rough) assumption that the signal is **stationary** (i.e. its statistical properties are constant within this region).

We'll do this by using a window which is non-zero inside some region and zero elsewhere, running this window across the speech signal, and extracting the waveform inside this window.

FRAME

FRAME SIZE

FRAME SHIFT

We can characterize such a windowing process by three parameters: how **wide** is the window (in milliseconds), what is the **offset** between successive windows, and what is the **shape** of the window. We call the speech extracted from each window a **frame**, and we call the number of milliseconds in the frame the **frame size** and the number of milliseconds between the left edges of successive windows the **frame shift**.

The extraction of the signal takes place by multiplying the value of the signal at

[1] For students who have had signal processing: this preemphasis filter is a first-order high-pass filter. In the time domain, with input $x[n]$ and $0.9 \le \alpha \le 1.0$, the filter equation is $y[n] = x[n] - \alpha x[n-1]$.

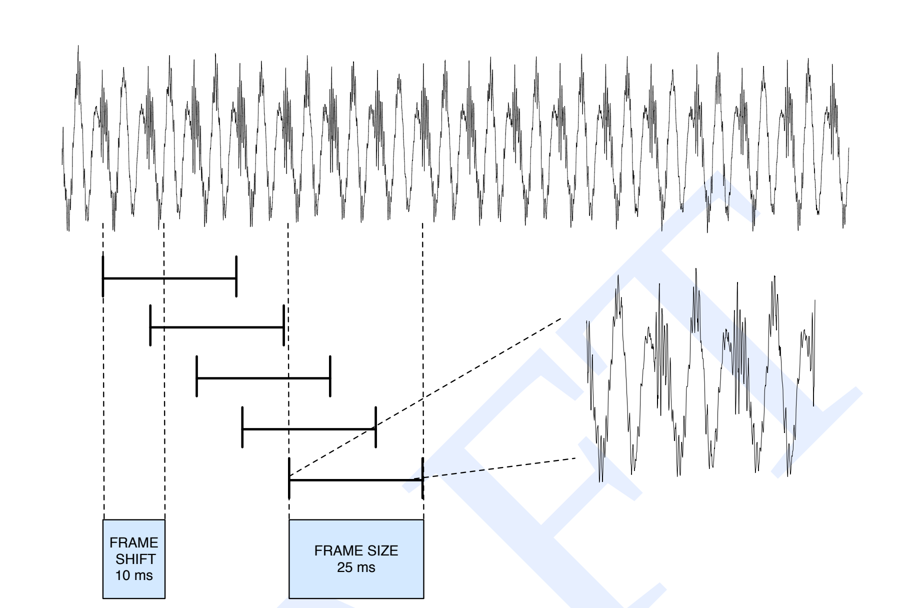


Figure 9.10 The windowing process, showing the frame shift and frame size, assuming a frame shift of 10ms, a frame size of 25 ms, and a rectangular window. After a figure by Bryan Pellom.

time n, $s[n]$, with the value of the window at time n, $w[n]$:

$$y[n] = w[n]s[n] \tag{9.9}$$

Figure 9.10 suggests that these window shapes are rectangular, since the extracted windowed signal looks just like the original signal. Indeed the simplest window is the **rectangular** window. The rectangular window can cause problems, however, because it abruptly cuts of the signal at its boundaries. These discontinuities create problems when we do Fourier analysis. For this reason, a more common window used in MFCC extraction is the **Hamming** window, which shrinks the values of the signal toward zero at the window boundaries, avoiding discontinuities. Fig. 9.11 shows both of these windows; the equations are as follows (assuming a window that is L frames long):

RECTANGULAR

HAMMING

$$\text{(9.10)}\quad rectangular \qquad w[n] = \begin{cases} 1 & 0 \le n \le L-1 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{(9.11)}\quad hamming \qquad w[n] = \begin{cases} 0.54 - 0.46\cos(\frac{2\pi n}{L}) & 0 \le n \le L-1 \\ 0 & \text{otherwise} \end{cases}$$

9.3.3 Discrete Fourier Transform

The next step is to extract spectral information for our windowed signal; we need to know how much energy the signal contains at different frequency bands. The tool for

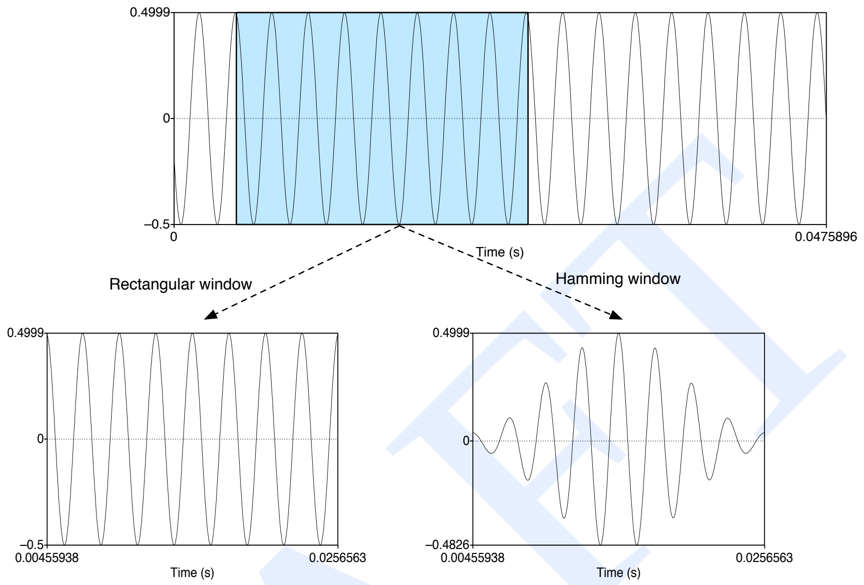


Figure 9.11 Windowing a portion of a pure sine wave with the rectangular and Hamming windows.

DISCRETE FOURIER TRANSFORM

DFT

extracting spectral information for discrete frequency bands for a discrete-time (sampled) signal is the **Discrete Fourier Transform** or **DFT**.

The input to the DFT is a windowed signal $x[n]...x[m]$, and the output, for each of N discrete frequency bands, is a complex number $X[k]$ representing the magnitude and phase of that frequency component in the original signal. If we plot the magnitude against the frequency, we can visualize the **spectrum** that we introduced in Ch. 7. For example, Fig. 9.12 shows a 25 ms Hamming-windowed portion of a signal and its spectrum as computed by a DFT (with some additional smoothing).

EULER'S FORMULA

We will not introduce the mathematical details of the DFT here, except to note that Fourier analysis in general relies on **Euler's formula**:

$$e^{j\theta} = \cos\theta + j\sin\theta \tag{9.12}$$

As a brief reminder for those students who have already had signal processing, the DFT is defined as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\frac{\pi}{N}kn} \tag{9.13}$$

FAST FOURIER TRANSFORM

FFT

A commonly used algorithm for computing the DFT is the the **Fast Fourier Transform** or **FFT**. This implementation of the DFT is very efficient, but only works for values of N which are powers of two.

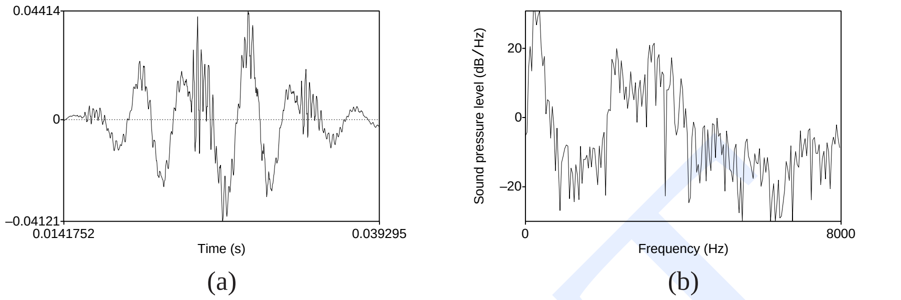


Figure 9.12 (a) A 25 ms Hamming-windowed portion of a signal from the vowel [iy] and (b) its spectrum computed by a DFT.

9.3.4 Mel filter bank and log

MEL

The results of the FFT will be information about the amount of energy at each frequency band. Human hearing, however, is not equally sensitive at all frequency bands. It is less sensitive at higher frequencies, roughly above 1000 Hertz. It turns out that modeling this property of human hearing during feature extraction improves speech recognition performance. The form of the model used in MFCCs is to warp the frequencies output by the DFT onto the **mel** scale mentioned in Ch. 7. A **mel** (Stevens et al., 1937; Stevens and Volkmann, 1940) is a unit of pitch defined so that pairs of sounds which are perceptually equidistant in pitch are separated by an equal number of mels. The mapping between frequency in Hertz and the mel scale is linear below 1000 Hz and the logarithmic above 1000 Hz. The mel frequency *m* can be computed from the raw acoustic frequency as follows:

$$mel(f) = 1127\ln(1+\frac{f}{700}) \qquad (9.14)$$

During MFCC computation, this intuition is implemented by creating a bank of filters which collect energy from each frequency band, with 10 filters spaced linearly below 1000 Hz, and the remaining filters spread logarithmically above 1000 Hz. Fig. 9.13 shows the bank of triangular filters that implement this idea.

Finally, we take the log of each of the mel spectrum values. In general the human response to signal level is logarithmic; humans are less sensitive to slight differences in amplitude at high amplitudes than at low amplitudes. In addition, using a log makes the feature estimates less sensitive to variations in input (for example power variations due to the speaker's mouth moving closer or further from the microphone).

9.3.5 The Cepstrum: Inverse Discrete Fourier Transform

CEPSTRUM

While it would be possible to use the mel spectrum by itself as a feature representation for phone detection, the spectrum also has some problems, as we will see. For this reason, the next step in MFCC feature extraction is the computation of the **cepstrum**. The

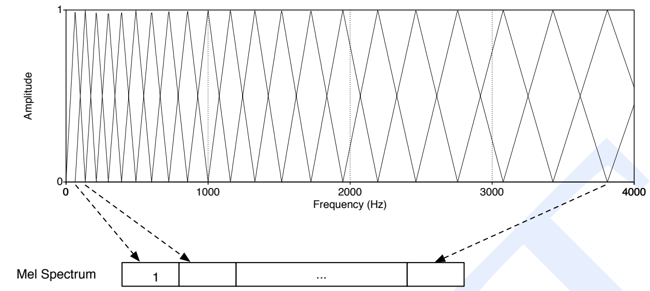
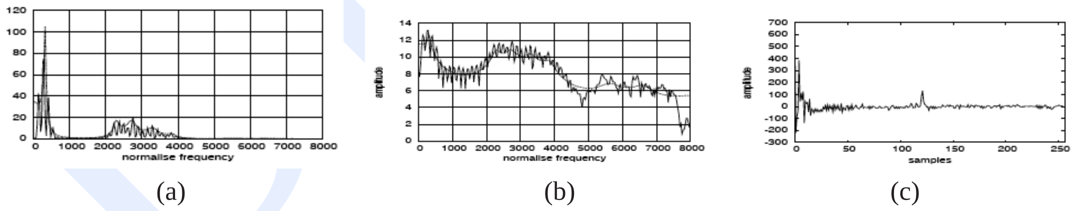


Figure 9.13 The Mel filter bank, after Davis and Mermelstein (1980). Each triangular filter collects energy from a given frequency range. Filters are spaced linearly below 1000 Hz, and logarithmically above 1000 Hz.

cepstrum has a number of useful processing advantages and also significantly improves phone recognition performance.

One way to think about the cepstrum is as a useful way of separating the **source** and **filter**. Recall from Sec. **??** that the speech waveform is created when a glottal source waveform of a particular fundamental frequency is passed through the vocal tract, which because of its shape has a particular filtering characteristic. But many characteristics of the glottal **source** (its fundamental frequency, the details of the glottal pulse, etc) are not important for distinguishing different phones. Instead, the most useful information for phone detection is the **filter**, i.e. the exact position of the vocal tract. If we knew the shape of the vocal tract, we would know which phone was being produced. This suggests that useful features for phone detection would find a way to deconvolve (separate) the source and filter and show us only the vocal tract filter. It turns out that the cepstrum is one way to do this.



(a) (b) (c)

Figure 9.14 PLACEHOLDER FIGURE. The magnitude spectrum (a), the log magnitude spectrum (b), and the cepstrum (c). From Taylor (2008). The two spectra have a smoothed spectral enveloped laid on top of them to help visualize the spectrum.

For simplicity, let's ignore the pre-emphasis and mel-warping that are part of the definition of MFCCs, and look just at the basic definition of the cepstrum. The cepstrum can be thought of as the *spectrum of the log of the spectrum*. This may sound confusing. But let's begin with the easy part: the *log of the spectrum*. That is, the cepstrum begins with a standard magnitude spectrum, such as the one for a vowel shown in Fig. 9.14(a) from Taylor (2008). We then take the log, i.e. replace each amplitude value in the magnitude spectrum with its log, as shown in Fig. 9.14(b).

The next step is to visualize the log spectrum *as if itself were a waveform*. In other words, consider the log spectrum in Fig. 9.14(b). Let's imagine removing the axis labels that tell us that this is a spectrum (frequency on the x-axis) and imagine that we are dealing with just a normal speech signal with time on the x-axis. Now what can we say about the spectrum of this 'pseudo-signal'? Notice that there is a high-frequency repetitive component in this wave: small waves that repeat about 8 times in each 1000 along the x-axis, for a frequency of about 120 Hz. This high-frequency component is caused by the fundamental frequency of the signal, and represents the little peaks in the spectrum at each harmonic of the signal. In addition, there are some lower frequency components in this 'pseudo-signal'; for example the envelope or formant structure has about four large peaks in the window, for a much lower frequency.

Fig. 9.14(c) shows the **cepstrum**: the spectrum that we have been describing of the log spectrum. This cepstrum (the word **cepstrum** is formed by reversing the first letters of **spectrum**) is shown with **samples** along the x-axis. This is because by taking the spectrum of the log spectrum, we have left the frequency domain of the spectrum, and gone back to the time domain. It turns out that the correct unit of a cepstrum is the sample.

Examining this cepstrum, we see that there is indeed a large peak around 120, corresponding to the F0 and representing the glottal pulse. There are other various components at lower values on the x-axis. These represent the vocal tract filter (the position of the tongue and the other articulators). Thus if we are interested in detecting phones, we can make use of just the lower cepstral values. If we are interested in detecting pitch, we can use the higher cepstral values.

For the purposes of MFCC extraction, we generally just take the first 12 cepstral values. These 12 coefficients will represent information solely about the vocal tract filter, cleanly separated from information about the glottal source.

It turns out that cepstral coefficients have the extremely useful property that the variance of the different coefficients tends to be uncorrelated. This is not true for the spectrum, where spectral coefficients at different frequency bands are correlated. The fact that cepstral features are uncorrelated means, as we will see in the next section, that the Gaussian acoustic model (the Gaussian Mixture Model, or GMM) doesn't have to represent the covariance between all the MFCC features, which hugely reduces the number of parameters.

For those who have had signal processing, the cepstrum is more formally defined as the **inverse DFT of the log magnitude of the DFT of a signal**, hence for a windowed frame of speech $x[n]$:

$$c[n] = \sum_{n=0}^{N-1} log\left(\left|\sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}kn}\right|\right) e^{j\frac{2\pi}{N}kn} \tag{9.15}$$

9.3.6 Deltas and Energy

The extraction of the cepstrum via the Inverse DFT from the previous section results in 12 cepstral coefficients for each frame. We next add a thirteenth feature: the energy

ENERGY

from the frame. Energy correlates with phone identity and so is a useful cue for phone detection (vowels and sibilants have more energy than stops, etc). The **energy** in a frame is the sum over time of the power of the samples in the frame; thus for a signal x in a window from time sample t_1 to time sample t_2, the energy is:

$$Energy = \sum_{t=t_1}^{t_2} x^2[t] \tag{9.16}$$

Another important fact about the speech signal is that it is not constant from frame to frame. This change, such as the slope of a formant at its transitions, or the nature of the change from a stop closure to stop burst, can provide a useful cue for phone identity. For this reason we also add features related to the change in cepstral features over time.

DELTA
VELOCITY
DOUBLE DELTA
ACCELERATION

We do this by adding for each of the 13 features (12 cepstral features plus energy) a **delta** or **velocity** feature, and a **double delta** or **acceleration** feature. Each of the 13 delta features represents the change between frames in the corresponding cepstral/energy feature, while each of the 13 double delta features represents the change between frames in the corresponding delta features.

A simple way to compute deltas would be just to compute the difference between frames; thus the delta value $d(t)$ for a particular cepstral value $c(t)$ at time t can be estimated as:

$$d(t) = \frac{c(t+1) - c(t-1)}{2} \tag{9.17}$$

Instead of this simple estimate, however, it is more common to make more sophisticated estimates of the slope, using a wider context of frames.

9.3.7 Summary: MFCC

After adding energy, and then delta and double-delta features to the 12 cepstral features, we end up with 39 MFCC features:

12	cepstral coefficients
12	delta cepstral coefficients
12	double delta cepstral coefficients
1	energy coefficient
1	delta energy coefficient
1	double delta energy coefficient
39	MFCC features

Again, one of the most useful facts about MFCC features is that the cepstral coefficients tend to be uncorrelated, which will turn out to make our acoustic model much simpler.

9.4 Computing Acoustic Likelihoods

The last section showed how we can extract MFCC features representing spectral information from a wavefile, and produce a 39-dimensional vector every 10 milliseconds. We are now ready to see how to compute the likelihood of these feature vectors given an HMM state. Recall from Ch. 6 that this output likelihood is computed by the B probability function of the HMM. Given an individual state q_i and an observation o_t, the observation likelihoods in B matrix gave us $p(o_t|q_i)$, which we called $b_t(i)$.

For part-of-speech tagging in Ch. 5, each observation o_t is a discrete symbol (a word) and we can compute the likelihood of an observation given a part-of-speech tag just by counting the number of times a given tag generates a given observation in the training set. But for speech recognition, MFCC vectors are real-valued numbers; we can't compute the likelihood of a given state (phone) generating an MFCC vector by counting the number of times each such vector occurs (since each one is likely to be unique).

In both decoding and training, we need an observation likelihood function that can compute $p(o_t|q_i)$ on real-valued observations. In decoding, we are given an observation o_t and we need to produce the probability $p(o_t|q_i)$ for each possible HMM state, so we can choose the most likely sequence of states. Once we have this observation likelihood B function, we need to figure out how to modify the Baum-Welch algorithm of Ch. 6 to train it as part of training HMMs.

9.4.1 Vector Quantization

VECTOR QUANTIZATION

VQ

One way to make MFCC vectors look like symbols that we could count is to build a mapping function that maps each input vector into one of a small number of symbols. Then we could just compute probabilities on these symbols by counting, just as we did for words in part-of-speech tagging. This idea of mapping input vectors to discrete quantized symbols is called **vector quantization** or **VQ** (Gray, 1984). Although vector quantization is too simple to act as the acoustic model in modern LVCSR systems, it is a useful pedagogical step, and plays an important role in various areas of ASR, so we use it to begin our discussion of acoustic modeling.

In vector quantization, we create the small symbol set by mapping each training feature vector into a small number of classes, and then we represent each class by a discrete symbol. More formally, a vector quantization system is characterized by a **codebook**, a **clustering algorithm**, and a **distance metric**.

CODEBOOK

PROTOTYPE VECTOR

CODEWORD

A **codebook** is a list of possible classes, a set of symbols constituting a vocabulary $V = \{v_1, v_2, ..., v_n\}$. For each symbol v_k in the codebook we list a **prototype vector**, also known as a **codeword**, which is a specific feature vector. For example if we choose to use 256 codewords we could represent each vector by a value from 0 to 255; (this is referred to as 8-bit VQ, since we can represent each vector by a single 8-bit value). Each of these 256 values would be associated with a prototype feature vector.

CLUSTERING

The codebook is created by using a **clustering** algorithm to cluster all the feature vectors in the training set into the 256 classes. Then we chose a representative feature vector from the cluster, and make it the prototype vector or codeword for that cluster.

K-MEANS CLUSTERING

K-means clustering is often used, but we won't define clustering here; see Huang et al. (2001) or Duda et al. (2000) for detailed descriptions.

Once we've built the codebook, for each incoming feature vector, we compare it to each of the 256 prototype vectors, select the one which is closest (by some **distance metric**), and replace the input vector by the index of this prototype vector. A schematic of this process is shown in Fig. 9.15.

The advantage of VQ is that since there are a finite number of classes, for each class v_k, we can compute the probability that it is generated by a given HMM state/sub-phone by simply counting the number of times it occurs in some training set when labeled by that state, and normalizing.

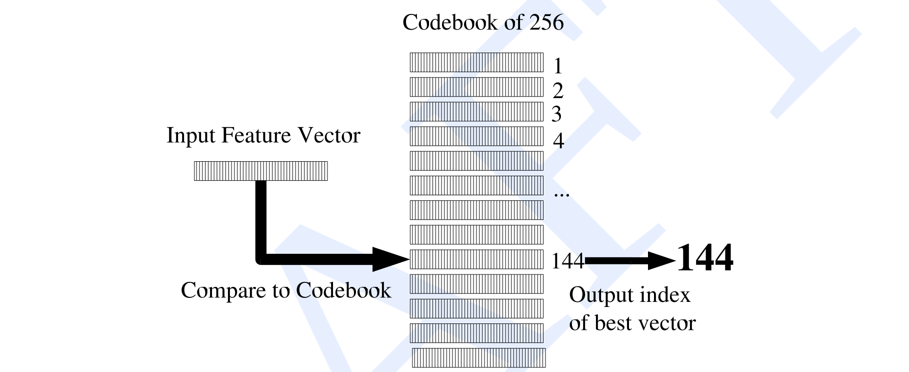


Figure 9.15 Schematic architecture of the (trained) vector quantization (VQ) process for choosing a symbol v_q for each input feature vector. The vector is compared to each codeword in the codebook, the closest entry (by some distance metric) is selected, and the index of the closest codeword is output.

DISTANCE METRIC

Both the clustering process and the decoding process require a **distance metric** or **distortion** metric, that specifies how similar two acoustic feature vectors are. The distance metric is used to build clusters, to find a prototype vector for each cluster, and to compare incoming vectors to the prototypes.

EUCLIDEAN DISTANCE

The simplest distance metric for acoustic feature vectors is **Euclidean distance**. Euclidean distance is the distance in N-dimensional space between the two points defined by the two vectors. In practice we use the phrase 'Euclidean distance' even though we actually often use the square of the Euclidean distance. Thus given a vector x and a vector y of length D, the (square of the) Euclidean distance between them is defined as:

$$d_{\text{euclidean}}(x,y) = \sum_{i=1}^{D}(x_i - y_i)^2 \tag{9.18}$$

The (squared) Euclidean distance described in (9.18) (and shown for two dimensions in Fig. 9.16) is also referred to as the sum-squared error, and can also be expressed using the vector transpose operator as:

$$d_{\text{euclidean}}(x,y) = (x-y)^T(x-y) \tag{9.19}$$

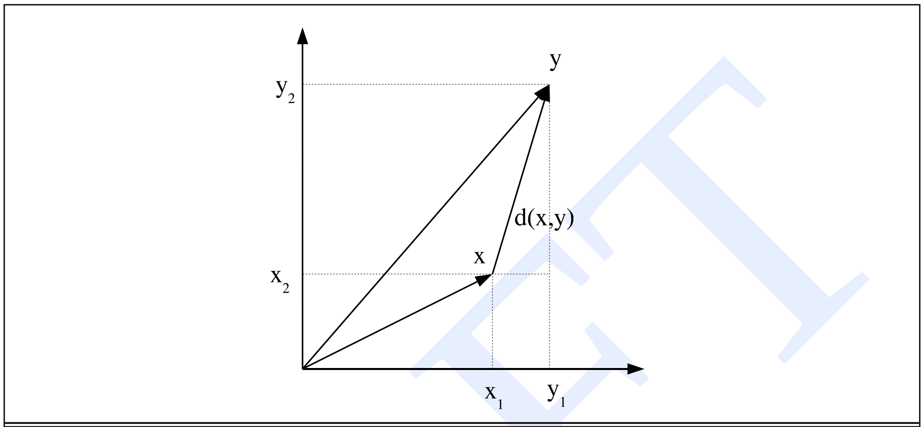


Figure 9.16 Euclidean distance in two dimensions; by the Pythagorean theorem, the distance between two points in a plane $x = (x1,y1)$ and $y = (x2,y2)$ $d(x,y) = \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$.

The Euclidean distance metric assumes that each of the dimensions of a feature vector are equally important. But actually each of the dimensions have very different variances. If a dimension tends to have a lot of variance, then we'd like it to count less in the distance metric; a large difference in a dimension with low variance should count more than a large difference in a dimension with high variance. A slightly more complex distance metric, the **Mahalanobis distance**, takes into account the different variances of each of the dimensions.

MAHALANOBIS DISTANCE

If we assume that each dimension i of the acoustic feature vectors has a variance σ_i^2, then the Mahalanobis distance is:

$$d_{\text{mahalanobis}}(x,y) = \sum_{i=1}^{D} \frac{(x_i-y_i)^2}{\sigma_i^2} \tag{9.20}$$

For those readers with more background in linear algebra here's the general form of Mahalanobis distance, which includes a full covariance matrix (covariance matrices will be defined below):

$$d_{\text{mahalanobis}}(x,y) = (x-y)^T\Sigma^{-1}(x-y) \tag{9.21}$$

In summary, when decoding a speech signal, to compute an acoustic likelihood of a feature vector o_t given an HMM state q_j using VQ, we compute the Euclidean or Mahalanobis distance between the feature vector and each of the N codewords, choose the closest codeword, getting the codeword index v_k. We then look up the likelihood of the codeword index v_k given the HMM state j in the pre-computed B likelihood matrix defined by the HMM:

$$\hat{b}_j(o_t) = b_j(v_k) \text{ s.t. } v_k \text{ is codeword of closest vector to } o_t \tag{9.22}$$

Since VQ is so rarely used, we don't use up space here giving the equations for modifying the EM algorithm to deal with VQ data; instead, we defer discussion of EM training of continuous input parameters to the next section, when we introduce Gaussians.

9.4.2 Gaussian PDFs

Vector quantization has the advantage of being extremely easy to compute and requires very little storage. Despite these advantages, vector quantization turns out not to be a good model of speech. A small number of codewords is insufficient to capture the wide variability in the speech signal. Speech is simply not a categorical, symbolic process.

PROBABILITY DENSITY FUNCTION

GAUSSIAN MIXTURE MODEL

GMM

Modern speech recognition algorithms therefore do not use vector quantization to compute acoustic likelihoods. Instead, they are based on computing observation probabilities directly on the real-valued, continuous input feature vector. These acoustic models are based on computing a **probability density function** or **pdf** over a continuous space. By far the most common method for computing acoustic likelihoods is the **Gaussian Mixture Model** (**GMM**) pdfs, although neural networks, support vector machines (SVMs) and conditional random fields (CRFs) are also used.

Let's begin with the simplest use of Gaussian probability estimators, slowly building up the more sophisticated models that are used.

Univariate Gaussians

GAUSSIAN

NORMAL DISTRIBUTION

MEAN

VARIANCE

The **Gaussian** distribution, also known as the **normal distribution**, is the bell-curve function familiar from basic statistics. A Gaussian distribution is a function parameterized by a **mean**, or average value, and a **variance**, which characterizes the average spread or dispersal from the mean. We will use μ to indicate the mean, and σ^2 to indicate the variance, giving the following formula for a Gaussian function:

$$f(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{9.23}$$

Recall from basic statistics that the mean of a random variable X is the expected value of X. For a discrete variable X, this is the weighted sum over the values of X (for a continuous variable, it is the integral):

$$\mu = E(X) = \sum_{i=1}^{N} p(X_i)X_i \tag{9.24}$$

The variance of a random variable X is the weigthed squared average deviation from the mean:

$$\sigma^2 = E(X_i - E(X))^2 = \sum_{i=1}^{N} p(X_i)(X_i - E(X))^2 \tag{9.25}$$

When a Gaussian function is used as a probability density function, the area under the curve is constrained to be equal to one. Then the probability that a random variable

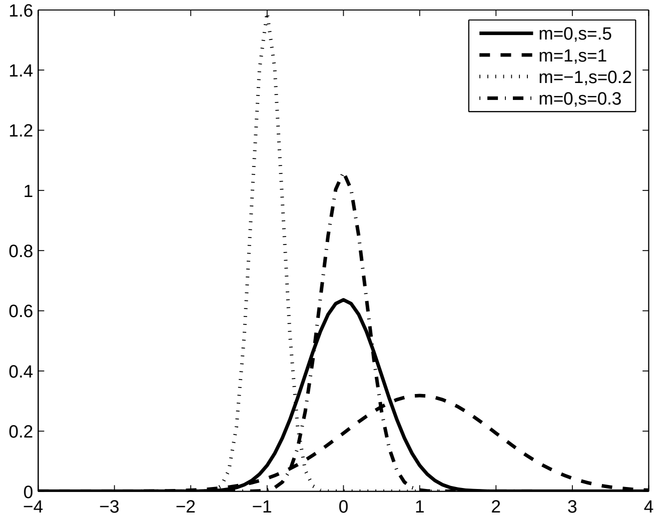


Figure 9.17 Gaussian functions with different means and variances.

takes on any particular range of values can be computed by summing the area under the curve for that range of values. Fig. 9.18 shows the probability expressed by the area under an interval of a Gaussian.

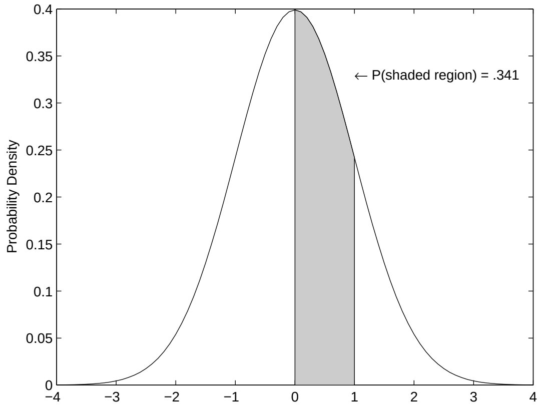


Figure 9.18 A Gaussian probability density function, showing a region from 0 to 1 with a total probability of .341. Thus for this sample Gaussian, the probability that a value on the X axis lies between 0 and 1 is .341.

We can use a univariate Gaussian pdf to estimate the probability that a particular HMM state j generates the value of a single dimension of a feature vector by assuming that the possible values of (this one dimension of the) observation feature vector o_t are normally distributed. In other words we represent the observation likelihood function $b_j(o_t)$ for one dimension of the acoustic vector as a Gaussian. Taking, for the moment, our observation as a single real valued number (a single cepstral feature), and assuming that each HMM state j has associated with it a mean value μ_j and variance σ_j^2, we compute the likelihood $b_j(o_t)$ via the equation for a Gaussian pdf:

$$b_j(o_t) = \frac{1}{\sqrt{2\pi\sigma_j^2}} exp\left(-\frac{(o_t-\mu_j)^2}{2\sigma_j^2}\right) \tag{9.26}$$

Equation (9.26) shows us how to compute $b_j(o_t)$, the likelihood of an individual acoustic observation given a single univariate Gaussian from state j with its mean and variance. We can now use this probability in HMM decoding.

But first we need to solve the training problem; how do we compute this mean and variance of the Gaussian for each HMM state q_i? Let's start by imagining the simpler situation of a completely labeled training set, in which each acoustic observation was labeled with the HMM state that produced it. In such a training set, we could compute the mean of each state by just taking the average of the values for each o_t that corresponded to state i, as show in (9.27). The variance could just be computed from the sum-squared error between each observation and the mean, as shown in (9.28).

$$\hat{\mu}_i = \frac{1}{T}\sum_{t=1}^{T} o_t \text{ s.t. } q_t \text{ is state } i \tag{9.27}$$

$$\hat{\sigma}_j^2 = \frac{1}{T}\sum_{t=1}^{T} (o_t-\mu_i)^2 \text{ s.t. } q_t \text{ is state } i \tag{9.28}$$

But since states are hidden in an HMM, we don't know exactly which observation vector o_t was produced by which state. What we would like to do is assign each observation vector o_t to every possible state i, prorated by the probability that the HMM was in state i at time t. Luckily, we already know how to do this prorating; the probability of being in state i at time t was defined in Ch. 6 as $\xi_t(i)$, and we saw how to compute $\xi_t(i)$ as part of the Baum-Welch algorithm using the forward and backward probabilities. Baum-Welch is an iterative algorithm, and we will need to do the probability computation of $\xi_t(i)$ iteratively since getting a better observation probability b will also help us be more sure of the probability ξ of being in a state at a certain time. Thus we give equations for computing an updated mean and variance $\hat{\mu}$ and $\hat{\sigma^2}$:

$$\hat{\mu}_i = \frac{\sum_{t=1}^{T}\xi_t(i)o_t}{\sum_{t=1}^{T}\xi_t(i)} \tag{9.29}$$

$$\hat{\sigma}_i^2 = \frac{\sum_{t=1}^{T}\xi_t(i)(o_t-\mu_i)^2}{\sum_{t=1}^{T}\xi_t(i)} \tag{9.30}$$

Equations (9.29) and (9.30) are then used in the forward-backward (Baum-Welch)training of the HMM. As we will see, the values of μ_i and σ_i are first set to some initial estimate, which is then re-estimated until the numbers converge.

Multivariate Gaussians

Equation (9.26) shows how to use a Gaussian to compute an acoustic likelihood for a single cepstral feature. Since an acoustic observation is a vector of 39 features, we'll need to use a multivariate Gaussian, which allows us to assign a probability to a 39-valued vector. Where a univariate Gaussian is defined by a mean μ and a variance σ^2, a multivariate Gaussian is defined by a mean vector $\vec{\mu}$ of dimensionality D and a covariance matrix Σ, defined below. As we discussed in the previous section, for a typical cepstral feature vector in LVCSR, D is 39:

$$f(\vec{x}|\vec{\mu},\Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)\right) \tag{9.31}$$

The covariance matrix Σ captures the variance of each dimension as well as the covariance between any two dimensions.

Recall again from basic statistics that the covariance of two random variables X and Y is the expected value of the product of their average deviations from the mean:

$$\Sigma = E[(X-E(X))(Y-E(Y))] = \sum_{i=1}^{N} p(X_iY_i)(X_i-E(X))(Y_i-E(Y)) \tag{9.32}$$

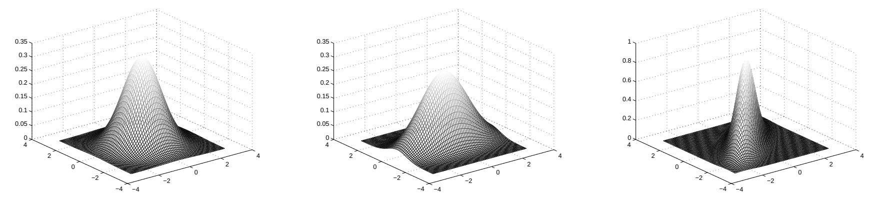
Thus for a given HMM state with mean vector μ_j and covariance matrix Σ_j, and a given observation vector o_t, the multivariate Gaussian probability estimate is:

$$b_j(o_t) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(o_t-\mu_j)^T\Sigma_j^{-1}(o_t-\mu_j)\right) \tag{9.33}$$

The covariance matrix Σ_j expresses the variance between each pair of feature dimensions. Suppose we made the simplifying assumption that features in different dimensions did not covary, i.e., that there was no correlation between the variances of different dimensions of the feature vector. In this case, we could simply keep a distinct variance for each feature dimension. It turns out that keeping a separate variance for each dimension is equivalent to having a covariance matrix that is **diagonal**, i.e. non-zero elements only appear along the main diagonal of the matrix. The main diagonal of such a diagonal covariance matrix contains the variances of each dimension, $\sigma_1^2, \sigma_2^2, \ldots \sigma_D^2$;

DIAGONAL

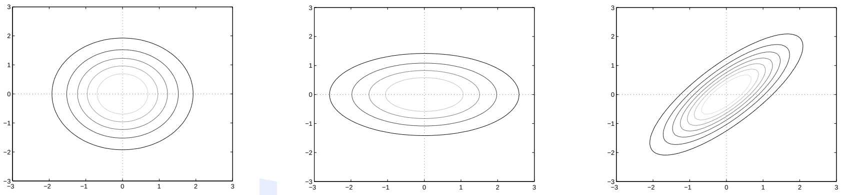
Let's look at some illustrations of multivariate Gaussians, focusing on the role of the full versus diagonal covariance matrix. We'll explore a simple multivariate Gaussian with only 2 dimensions, rather than the 39 that are typical in ASR. Fig. 9.19 shows three different multivariate Gaussians in two dimensions. The leftmost figure shows a Gaussian with a diagonal covariance matrix, in which the variances of the two dimensions are equal. Fig. 9.20 shows 3 contour plots corresponding to the Gaussians in Fig. 9.19; each is a slice through the Gaussian. The leftmost graph in Fig. 9.20 shows a slice through the diagonal equal-variance Gaussian. The slice is circular, since the variances are equal in both the X and Y directions.



(a) (b) (c)

Figure 9.19 Three different multivariate Gaussians in two dimensions. The first two have diagonal covariance matrices, one with equal variance in the two dimensions $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, the second with different variances in the two dimensions, $\begin{bmatrix} .6 & 0 \\ 0 & 2 \end{bmatrix}$, and the third with non-zero elements in the off-diagonal of the covariance matrix: $\begin{bmatrix} 1 & .8 \\ .8 & 1 \end{bmatrix}$.

The middle figure in Fig. 9.19 shows a Gaussian with a diagonal covariance matrix, but where the variances are not equal. It is clear from this figure, and especially from the contour slice show in Fig. 9.20, that the variance is more than 3 times greater in one dimension than the other.



(a) (b) (c)

Figure 9.20 The same three multivariate Gaussians as in the previous figure. From left to right, a diagonal covariance matrix with equal variance, diagonal with unequal variance, and and nondiagonal covariance. With non-diagonal covariance, knowing the value on dimension X tells you something about the value on dimension Y.

The rightmost graph in Fig. 9.19 and Fig. 9.20 shows a Gaussian with a non-diagonal covariance matrix. Notice in the contour plot in Fig. 9.20 that the contour is not lined up with the two axes, as it is in the other two plots. Because of this, knowing the value in one dimension can help in predicting the value in the other dimension. Thus having a non-diagonal covariance matrix allows us to model correlations between the values of the features in multiple dimensions.

A Gaussian with a full covariance matrix is thus a more powerful model of acoustic likelihood than one with a diagonal covariance matrix. And indeed, speech recognition performance is better using full-covariance Gaussians than diagonal-covariance Gaussians. But there are two problems with full-covariance Gaussians that makes them difficult to use in practice. First, they are slow to compute. A full covariance matrix has D^2 parameters, where a diagonal covariance matrix has only D. This turns out to make a large difference in speed in real ASR systems. Second, a full covariance matrix has many more parameters and hence requires much more data to train than a diagonal covariance matrix. Using a diagonal covariance model means we can save room for using our parameters for other things like triphones (context-dependent phones) to be introduced in Sec. **??**.

For this reason, in practice most ASR systems use diagonal covariance. We will assume diagonal covariance for the remainder of this section.

Equation (9.33) can thus be simplified to the version in (9.34) in which instead of a covariance matrix, we simply keep a mean and variance for each dimension. Equation (9.34) thus describes how to estimate the likelihood $b_j(o_t)$ of a D-dimensional feature vector o_t given HMM state j, using a diagonal-covariance multivariate Gaussian.

$$b_j(o_t) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{jd}^2}} exp\left(-\frac{1}{2}[\frac{(o_{td}-\mu_{jd})^2}{{\sigma_{jd}}^2}]\right) \tag{9.34}$$

Training a diagonal-covariance multivariate Gaussian is a simple generalization of training univariate Gaussians. We'll do the same Baum-Welch training, where we use the value of $\xi_t(i)$ to tell us the likelihood of being in state i at time t. Indeed, we'll use exactly the same equation as in (9.30), except that now we are dealing with vectors instead of scalars; the observation o_t is a vector of cepstral features, the mean vector $\vec{\mu}$ is a vector of cepstral means, and the variance vector $\vec{\sigma_i^2}$ is a vector of cepstral variances.

$$\hat{\mu}_i = \frac{\sum_{t=1}^{T}\xi_t(i)o_t}{\sum_{t=1}^{T}\xi_t(i)} \tag{9.35}$$

$$\hat{\sigma}_i^2 = \frac{\sum_{t=1}^{T}\xi_t(i)(o_t-\mu_i)(o_t-\mu_i)^T}{\sum_{t=1}^{T}\xi_t(i)} \tag{9.36}$$

Gaussian Mixture Models

The previous subsection showed that we can use a multivariate Gaussian model to assign a likelihood score to an acoustic feature vector observation. This models each dimension of the feature vector as a normal distribution. But a particular cepstral feature might have a very non-normal distribution; the assumption of a normal distribution may be too strong an assumption. For this reason, we often model the observation likelihood not with a single multivariate Gaussian, but with a weighted mixture of multivariate Gaussians. Such a model is called a **Gaussian Mixture Model** or **GMM**. Equation (9.37) shows the equation for the GMM function; the resulting function is the sum of M Gaussians. Fig. 9.21 shows an intuition of how a mixture of Gaussians can model arbitrary functions.

GAUSSIAN MIXTURE MODEL

GMM

$$f(x|\mu,\Sigma) = \sum_{k=1}^{M} c_k \frac{1}{\sqrt{2\pi|\Sigma_k|}} exp[(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)] \tag{9.37}$$

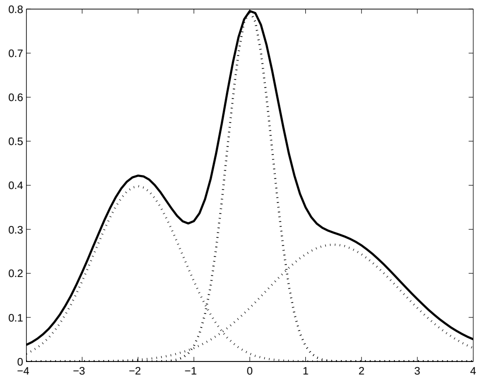


Figure 9.21 An arbitrary function approximated by a mixture of 3 gaussians.

Equation (9.38) shows the definition of the output likelihood function $b_j(o_t)$

$$b_j(o_t) = \sum_{m=1}^{M} c_{jm} \frac{1}{\sqrt{2\pi|\Sigma_{jm}|}} exp[(x-\mu_{jm})^T \Sigma_{jm}^{-1}(o_t-\mu_{jm})] \tag{9.38}$$

Let's turn to training the GMM likelihood function. This may seem hard to do; how can we train a GMM model if we don't know in advance which mixture is supposed to account for which part of each distribution? Recall that a single multivariate Gaussian could be trained even if we didn't know which state accounted for each output, simply by using the Baum-Welch algorithm to tell us the likelihood of being in each state j at time t. It turns out the same trick will work for GMMs; we can use Baum-Welch to tell us the probability of a certain mixture accounting for the observation, and iteratively update this probability.

We used the ξ function above to help us compute the state probability. By analogy with this function, let's define $\xi_{tm}(j)$ to mean the probability of being in state j at time t with the mth mixture component accounting for the output observation o_t. We can compute $\xi_{tm}(j)$ as follows:

$$\xi_{tm}(j) = \frac{\sum_{i=1} N\alpha_{t-1}(j)a_{ij}c_{jm}b_{jm}(o_t)\beta_t(j)}{\alpha_T(F)} \tag{9.39}$$

Now if we had the values of ξ from a previous iteration of Baum-Welch, we can use $\xi_{tm}(j)$ to recompute the mean, mixture weight, and covariance using the following equations:

$$\hat{\mu}_{im} = \frac{\sum_{t=1}^{T} \xi_{tm}(i) o_t}{\sum_{t=1}^{T}\sum_{m=1}^{M} \xi_{tm}(i)} \tag{9.40}$$

$$\hat{c}_{im} = \frac{\sum_{t=1}^{T} \xi_{tm}(i)}{\sum_{t=1}^{T}\sum_{k=1}^{M} \xi_{tk}(i)} \tag{9.41}$$

$$\hat{\Sigma}_{im} \;=\; \frac{\sum_{t=1}^{T} \xi_t(i)(o_t-\mu_{im})(o_t-\mu_{im})^T}{\sum_{t=1}^{T}\sum_{k=1}^{M} \xi_{tm}(i)} \tag{9.42}$$

9.4.3 Probabilities, log probabilities and distance functions

LOGPROB

Up to now, all the equations we have given for acoustic modeling have used probabilities. It turns out, however, that a **log probability** (or **logprob**) is much easier to work with than a probability. Thus in practice throughout speech recognition (and related fields) we compute log-probabilities rather than probabilities.

One major reason that we can't use probabilities is numeric underflow. To compute a likelihood for a whole sentence, say, we are multiplying many small probability values, one for each 10ms frame. Multiplying many probabilities results in smaller and smaller numbers, leading to underflow. The log of a small number like $.00000001 = 10^{-8}$, on the other hand, is a nice easy-to-work-with-number like -8. A second reason to use log probabilities is computational speed. Instead of multiplying probabilities, we add log-probabilities, and adding is faster than multiplying. Log-probabilities are particularly efficient when we are using Gaussian models, since we can avoid exponentiating.

Thus for example for a single multivariate diagonal-covariance Gaussian model, instead of computing:

$$b_j(o_t) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{jd}^2}} exp\left(-\frac{1}{2}\frac{(o_{td}-\mu_{jd})^2}{\sigma_{jd}^2}\right) \tag{9.43}$$

we would compute

$$\log b_j(o_t) = -\frac{1}{2}\sum_{d=1}^{D}\left[log(2\pi)+\sigma_{jd}^2+\frac{(o_{td}-\mu_{jd})^2}{\sigma_{jd}^2}\right] \tag{9.44}$$

With some rearrangement of terms, we can rewrite this equation to pull out a constant C:

$$\log b_j(o_t) = C - \frac{1}{2}\sum_{d=1}^{D}\frac{(o_{td}-\mu_{jd})^2}{\sigma_{jd}^2} \tag{9.45}$$

where C can be precomputed:

$$C = -\frac{1}{2}\sum_{d=1}^{D}\left(\log(2\pi)+\sigma_{jd}^2\right) \tag{9.46}$$

In summary, computing acoustic models in log domain means a much simpler computation, much of which can be precomputed for speed.

The perceptive reader may have noticed that equation (9.45) looks very much like the equation for Mahalanobis distance (9.20). Indeed, one way to think about Gaussian logprobs is as just a weighted distance metric.

A further point about Gaussian pdfs, for those readers with calculus. Although the equations for observation likelihood such as (9.26) are motivated by the use of Gaussian probability density functions, the values they return for the observation likelihood,

$b_j(o_t)$, are not technically probabilities; they may in fact be greater than one. This is because we are computing the value of $b_j(o_t)$ at a single point, rather than integrating over a region. While the total area under the Gaussian PDF curve is constrained to one, the actual value at any point could be greater than one. (Imagine a very tall skinny Gaussian; the value could be greater than one at the center, although the area under the curve is still 1.0). If we were integrating over a region, we would be multiplying each point by its width dx, which would bring the value down below one. The fact that the Gaussian estimate is not a true probability doesn't matter for choosing the most likely HMM state, since we are comparing different Gaussians, each of which is missing this dx factor.

In summary, the last few subsections introduced Gaussian models for acoustic training in speech recognition. Beginning with simple univariate Gaussian, we extended first to multivariate Gaussians to deal with the multidimensionality acoustic feature vectors. We then introduced the diagonal covariance simplification of Gaussians, and then introduced Gaussians mixtures (GMMs).

9.5 The Lexicon and Language Model

Since previous chapters had extensive discussions of the N-gram language model (Ch. 4) and the pronunciation lexicon (Ch. 7), in this section we just briefly recall them to the reader.

Language models for LVCSR tend to be trigrams or even fourgrams; good toolkits are available to build and manipulate them (Stolcke, 2002; Young et al., 2005). Bigrams and unigram grammars are rarely used for large-vocabulary applications. Since trigrams require huge amounts of space, however, language models for memory-constrained applications like cell phones tend to use smaller contexts (or use compression techniques). As we will discuss in Ch. 24, some simple dialogue applications take advantage of their limited domain to use very simple finite state or weighted-finite state grammars.

Lexicons are simply lists of words, with a pronunciation for each word expressed as a phone sequence. Publicly available lexicons like the CMU dictionary (CMU, 1993) can be used to extract the 64,000 word vocabularies commonly used for LVCSR. Most words have a single pronunciation, although some words such as homonyms and frequent function words may have more; the average number of pronunciations per word in most LVCSR systems seems to range from 1 to 2.5. Sec. **??** in Ch. 10 discusses the issue of pronunciation modeling.

9.6 Search and Decoding

We are now very close to having described all the parts of a complete speech recognizer. We have shown how to extract cepstral features for a frame, and how to compute the acoustic likelihood $b_j(o_t)$ for that frame. We also know how to represent lexical knowledge, that each word HMM is composed of a sequence of phone models, and each phone model of a set of subphone states. Finally, in Ch. 4 we showed how to use N-grams to build a model of word predictability.

DECODING

In this section we show how to combine all of this knowledge to solve the problem of **decoding**: combining all these probability estimators to produce the most probable string of words. We can phrase the decoding question as: ‘Given a string of acoustic observations, how should we choose the string of words which has the highest posterior probability?’

Recall from the beginning of the chapter the noisy channel model for speech recognition. In this model, we use Bayes rule, with the result that the best sequence of words is the one that maximizes the product of two factors, a language model prior and an acoustic likelihood:

$$\hat{W} = \underset{W \in \mathscr{L}}{\operatorname{argmax}} \overbrace{P(O|W)}^{\text{likelihood}} \overbrace{P(W)}^{\text{prior}} \tag{9.47}$$

Now that we have defined both the acoustic model and language model we are ready to see how to find this maximum probability sequence of words. First, though, it turns out that we’ll need to make a modification to Equation (9.47), because it relies on some incorrect independence assumptions. Recall that we trained a multivariate Gaussian mixture classifier to compute the likelihood of a particular acoustic observation (a frame) given a particular state (subphone). By computing separate classifiers for each acoustic frame and multiplying these probabilities to get the probability of the whole word, we are severely underestimating the probability of each subphone. This is because there is a lot of continuity across frames; if we were to take into account the acoustic context, we would have a greater expectation for a given frame and hence could assign it a higher probability. We must therefore reweight the two probabilities. We do this by adding in a **language model scaling factor** or **LMSF**, also called the **language weight**. This factor is an exponent on the language model probability $P(W)$. Because $P(W)$ is less than one and the LMSF is greater than one (between 5 and 15, in many systems), this has the effect of decreasing the value of the LM probability:

LMSF

$$\hat{W} = \underset{W \in \mathscr{L}}{\operatorname{argmax}} P(O|W)P(W)^{LMSF} \tag{9.48}$$

Reweighting the language model probability $P(W)$ in this way requires us to make one more change. This is because $P(W)$ has a side-effect as a penalty for inserting words. It’s simplest to see this in the case of a uniform language model, where every word in a vocabulary of size $|V|$ has an equal probability $\frac{1}{|V|}$. In this case, a sentence with N words will have a language model probability of $\frac{1}{|V|}$ for each of the N words, for a total penalty of of $\frac{N}{|V|}$. The larger N is (the more words in the sentence), the more times this $\frac{1}{V}$ penalty multiplier is taken, and the less probable the sentence will be. Thus if (on average) the language model probability decreases (causing a larger penalty), the decoder will prefer fewer, longer words. If the language model probability increases (larger penalty), the decoder will prefer more shorter words. Thus our use of a LMSF to balance the acoustic model has the side-effect of decreasing the word insertion penalty. To offset this, we need to add back in a separate **word insertion penalty**:

WORD INSERTION PENALTY

$$\hat{W} = \underset{W \in \mathscr{L}}{\operatorname{argmax}} P(O|W)P(W)^{LMSF}\,WIP^{N} \tag{9.49}$$

Since in practice we use logprobs, the goal of our decoder is:

$$\hat{W} = \underset{W \in \mathscr{L}}{\operatorname{argmax}} \log P(O|W) + LMSF \times \log P(W) + N \times \log WIP \tag{9.50}$$

Now that we have an equation to maximize, let's look at how to decode. It's the job of a decoder to simultaneously segment the utterance into words and identify each of these words. This task is made difficult by variation, both in terms of how words are pronounced in terms of phones, and how phones are articulated in acoustic features. Just to give an intuition of the difficulty of the problem imagine a massively simplified version of the speech recognition task, in which the decoder is given a series of discrete phones. In such a case, we would know what each phone was with perfect accuracy, and yet decoding is still difficult. For example, try to decode the following sentence from the (hand-labeled) sequence of phones from the Switchboard corpus (don't peek ahead!):

[ay d ih s hh er d s ah m th ih ng ax b aw m uh v ih ng r ih s en l ih]

The answer is in the footnote.[2] The task is hard partly because of coarticulation and fast speech (e.g., [d] for the first phone of *just*!). But it's also hard because speech, unlike English writing, has no spaces indicating word boundaries. The true decoding task, in which we have to identify the phones at the same time as we identify and segment the words, is of course much harder.

For decoding, we will start with the Viterbi algorithm that we introduced in Ch. 6, in the domain of **digit recognition**, a simple task with with a vocabulary size of 11 (the numbers *one* through *nine* plus *zero* and *oh*).

Recall the basic components of an HMM model for speech recognition:

$Q = q_1 q_2 \ldots q_N$ a set of **states** corresponding to **subphones**

$A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn}$ a **transition probability matrix** A, each a_{ij} representing the probability for each subphone of taking a **self-loop** or going to the next subphone. Together, Q and A implement a **pronunciation lexicon**, an HMM state graph structure for each word that the system is capable of recognizing.

$B = b_i(o_t)$ A set of **observation likelihoods:**, also called **emission probabilities**, each expressing the probability of a cepstral feature vector (observation o_t) being generated from subphone state i.

The HMM structure for each word comes from a lexicon of word pronunciations. Generally we use an off-the-shelf pronunciation dictionary such as the free CMUdict dictionary described in Ch. 7. Recall from page 9 that the HMM structure for words in

[2] I just heard something about moving recently.

speech recognition is a simple concatenation of phone HMMs, each phone consisting of 3 subphone states, where every state has exactly two transitions: a self-loop and a loop to the next phones. Thus the HMM structure for each digit word in our digit recognizer is computed simply by taking the phone string from the dictionary, expanding each phone into 3 subphones, and concatenating together. In addition, we generally add an optional silence phone at the end of each word, allowing the possibility of pausing between words. We usually define the set of states Q from some version of the ARPAbet, augmented with silence phones, and expanded to create three subphones for each phone.

The A and B matrices for the HMM are trained by the Baum-Welch algorithm in the **embedded training** procedure that we will describe in Sec. 9.7. For now we'll assume that these probabilities have been trained.

Fig. 9.22 shows the resulting HMM for digit recognition. Note that we've added non-emitting start and end states, with transitions from the end of each word to the end state, and a transition from the end state back to the start state to allow for sequences of digits. Note also the optional silence phones at the end of each word.

Digit recognizers often don't use word probabilities, since in many digit situations (phone numbers or credit card numbers) each digit may have an equal probability of appearing. But we've included transition probabilities into each word in Fig. 9.22, mainly to show where such probabilities would be for other kinds of recognition tasks. As it happens, there are cases where digit probabilities do matter, such as in addresses (which are often likely to end in 0 or 00) or in cultures where some numbers are lucky and hence more frequent, such as the lucky number '8' in Chinese.

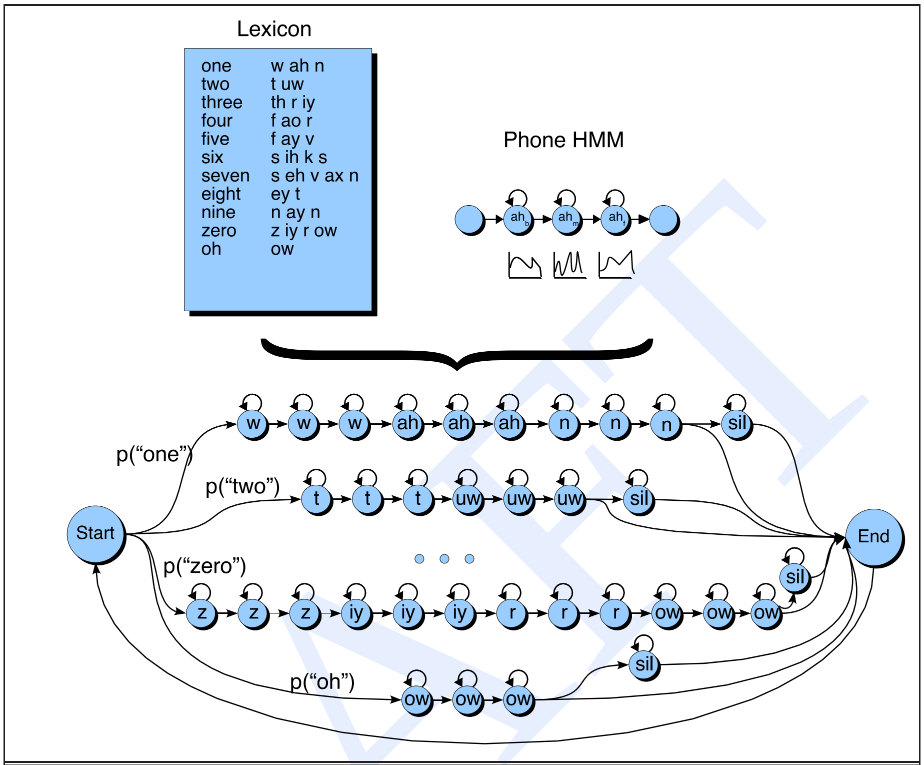
Now that we have an HMM, we can use the same forward and Viterbi algorithms that we introduced in Ch. 6. Let's see how to use the forward algorithm to generate $P(O|W)$, the likelihood of an observation sequence O given a sequence of words W; we'll use the single word "five". In order to compute this likelihood, we need to sum over all possible sequences of states; assuming *five* has the states [f], [ay], and [v], a 10-observation sequence includes many sequences such as the following:

```
f ay ay ay ay v  v  v  v  v
f f  ay ay ay ay v  v  v  v
f f  f  f  ay ay ay ay v  v
f f  ay ay ay ay ay ay v  v
f f  ay ay ay ay ay ay ay v
f f  ay ay ay ay ay v  v  v
...
```

The forward algorithm efficiently sums over this large number of sequences in $O(N^2T)$ time.

Let's quickly review the forward algorithm. It is a dynamic programming algorithm, i.e. an algorithm that uses a table to store intermediate values as it builds up the probability of the observation sequence. The forward algorithm computes the observation probability by summing over the probabilities of all possible paths that could generate the observation sequence.

Each cell of the forward algorithm trellis $\alpha_t(j)$ or *forward*$[t,j]$ represents the probability of being in state j after seeing the first t observations, given the automaton λ. The



Lexicon

one	w ah n
two	t uw
three	th r iy
four	f ao r
five	f ay v
six	s ih k s
seven	s eh v ax n
eight	ey t
nine	n ay n
zero	z iy r ow
oh	ow

Figure 9.22 An HMM for the digit recognition task. A lexicon specifies the phone sequence, and each phone HMM is composed of three subphones each with a Gaussian emission likelihood model. Combining these and adding an optional silence at the end of each word, results in a single HMM for the whole task. Note the transition from the End state to the Start state to allow digit sequences of arbitrary length.

value of each cell $\alpha_t(j)$ is computed by summing over the probabilities of every path that could lead us to this cell. Formally, each cell expresses the following probability:

$$\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j|\lambda) \tag{9.51}$$

Here $q_t = j$ means "the probability that the tth state in the sequence of states is state j". We compute this probability by summing over the extensions of all the paths that lead to the current cell. For a given state q_j at time t, the value $\alpha_t(j)$ is computed as:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \tag{9.52}$$

The three factors that are multiplied in Eq˙ 9.52 in extending the previous paths to compute the forward probability at time t are:

$\alpha_{t-1}(i)$ the **previous forward path probability** from the previous time step
a_{ij} the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$ the **state observation likelihood** of the observation symbol o_t given the current state j

The algorithm is described in Fig. 9.23.

```
function FORWARD(observations of len T, state-graph of len N) returns forward-prob

  create a probability matrix forward[N+2,T]
  for each state s from 1 to N do                          ;initialization step
      $forward[s,1] \leftarrow a_{0,s} * b_s(o_1)$
  for each time step t from 2 to T do                      ;recursion step
     for each state s from 1 to N do
            $forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$
  $forward[q_F,T] \leftarrow \sum_{s=1}^{N} forward[s,T] * a_{s,q_F}$          ; termination step
  return $forward[q_F,T]$
```

Figure 9.23 The forward algorithm for computing likelihood of observation sequence given a word model. $a[s,s']$ is the transition probability from current state s to next state s', and $b[s',o_t]$ is the observation likelihood of s' given o_t. The observation likelihood $b[s',o_t]$ is computed by the **acoustic model**.

Let's see a trace of the forward algorithm running on a simplified HMM for the single word *five* given 10 observations; assuming a frame shift of 10ms, this comes to 100ms. The HMM structure is shown vertically along the left of Fig. 9.24, followed by the first 3 time-steps of the forward trellis. The complete trellis is shown in Fig. 9.25, together with B values giving a vector of observation likelihoods for each frame. These likelihoods could be computed by any acoustic model (GMMs or other); in this example we've hand-created simple values for pedagogical purposes.

Let's now turn to the question of decoding. Recall the Viterbi decoding algorithm from our description of HMMs in Ch. 6. The Viterbi algorithm returns the most likely state sequence (which is not the same as the most likely word sequence, but is often a good enough approximation) in time $O(N^2T)$.

Each cell of the Viterbi trellis, $v_t(j)$ represents the probability that the HMM is in state j after seeing the first t observations and passing through the most likely state sequence $q_1...q_{t-1}$, given the automaton λ. The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that could lead us to this cell. Formally, each cell expresses the following probability:

$$v_t(j) = P(q_0,q_1...q_{t-1},o_1,o_2\ldots o_t,q_t = j|\lambda) \tag{9.53}$$

Like other dynamic programming algorithms, Viterbi fills each cell recursively. Given that we had already computed the probability of being in every state at time

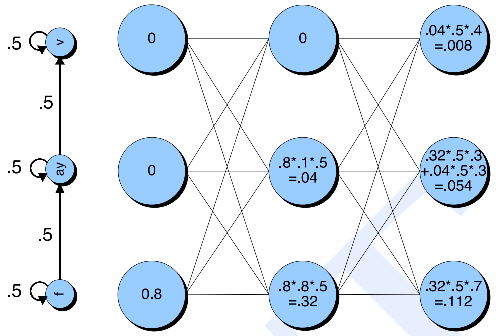


Figure 9.24 The first 3 time-steps of the forward trellis computation for the word *five*. The *A* transition probabilities are shown along the left edge; the *B* observation likelihoods are shown in Fig. 9.25.

V	0	0	0.008	0.0093	0.0114	0.00703	0.00345	0.00306	0.00206	0.00117
AY	0	0.04	0.054	0.0664	0.0355	0.016	0.00676	0.00208	0.000532	0.000109
F	0.8	0.32	0.112	0.0224	0.00448	0.000896	0.000179	4.48e-05	1.12e-05	2.8e-06
Time	1	2	3	4	5	6	7	8	9	10
	f 0.8	*f* 0.8	*f* 0.7	*f* 0.4	*f* 0.4	*f* 0.4	*f* 0.4	*f* 0.5	*f* 0.5	*f* 0.5
	ay 0.1	*ay* 0.1	*ay* 0.3	*ay* 0.8	*ay* 0.8	*ay* 0.8	*ay* 0.8	*ay* 0.6	*ay* 0.5	*ay* 0.4
B	*v* 0.6	*v* 0.6	*v* 0.4	*v* 0.3	*v* 0.3	*v* 0.3	*v* 0.3	*v* 0.6	*v* 0.8	*v* 0.9
	p 0.4	*p* 0.4	*p* 0.2	*p* 0.1	*p* 0.1	*p* 0.1	*p* 0.1	*p* 0.1	*p* 0.3	*p* 0.3
	iy 0.1	*iy* 0.1	*iy* 0.3	*iy* 0.6	*iy* 0.6	*iy* 0.6	*iy* 0.6	*iy* 0.5	*iy* 0.5	*iy* 0.4

Figure 9.25 The forward trellis for 10 frames of the word *five*, consisting of 3 emitting states (*f*, *ay*, *v*), plus non-emitting start and end states (not shown). The bottom half of the table gives part of the *B* observation likelihood vector for the observation *o* at each frame, $p(o|q)$ for each phone *q*. *B* values are created by hand for pedagogical purposes. This table assumes the HMM structure for *five* shown in Fig. 9.24, each emitting state having a .5 loopback probability.

$t-1$, We compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell. For a given state q_j at time t, the value $v_t(j)$ is computed as:

$$v_t(j) \;=\; \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t) \qquad (9.54)$$

The three factors that are multiplied in Eq. 9.54 for extending the previous paths to compute the Viterbi probability at time t are:

$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

Fig. 9.26 shows the Viterbi algorithm, repeated from Ch. 6.

```
function VITERBI(observations of len T,state-graph of len N) returns best-path

  create a path probability matrix viterbi[N+2,T]
  for each state s from 1 to N do                          ;initialization step
      $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
      $backpointer[s,1] \leftarrow 0$
  for each time step t from 2 to T do                      ;recursion step
    for each state s from 1 to N do
      $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
      $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
  $viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$          ; termination step
  $backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$          ; termination step
   return the backtrace path by following backpointers to states back in time from
backpointer[$q_F$, $T$]
```

Figure 9.26 Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence of words and an HMM (as defined by the A and B matrices), the algorithm returns the state-path through the HMM which assigns maximum likelihood to the observation sequence. $a[s',s]$ is the transition probability from previous state s' to current state s, and $b_s(o_t)$ is the observation likelihood of s given o_t. Note that states 0 and F are non-emitting start and end states.

Recall that the goal of the Viterbi algorithm is to find the best state sequence $q = (q_1 q_2 q_3 \ldots q_T)$ given the set of observations $o = (o_1 o_2 o_3 \ldots o_T)$. It needs to also find the probability of this state sequence. Note that the Viterbi algorithm is identical to the forward algorithm except that it takes the MAX over the previous path probabilities where forward takes the SUM.

Fig. 9.27 shows the computation of the first three time-steps in the Viterbi trellis corresponding to the forward trellis in Fig. 9.24. We have again used the made-up probabilities for the cepstral observations; here we also follow common convention in not showing the zero cells in the upper left corner. Note that only the middle cell in the third column differs from Viterbi to forward. Fig. 9.25 shows the complete trellis.

Note the difference between the final values from the Viterbi and forward algorithms for this (made-up) example. The forward algorithm gives the probability of the observation sequence as .00128, which we get by summing the final column. The Viterbi algorithm gives the probability of the observation sequence given the best path, which we get from the Viterbi matrix as .000493. The Viterbi probability is much smaller than the forward probability, as we should expect since Viterbi comes from a single path, where the forward probability is the sum over all paths.

The real usefulness of the Viterbi decoder, of course, lies in its ability to decode a string of words. In order to do cross-word decoding, we need to augment the A matrix, which only has intra-word state transitions, with the inter-word probability of

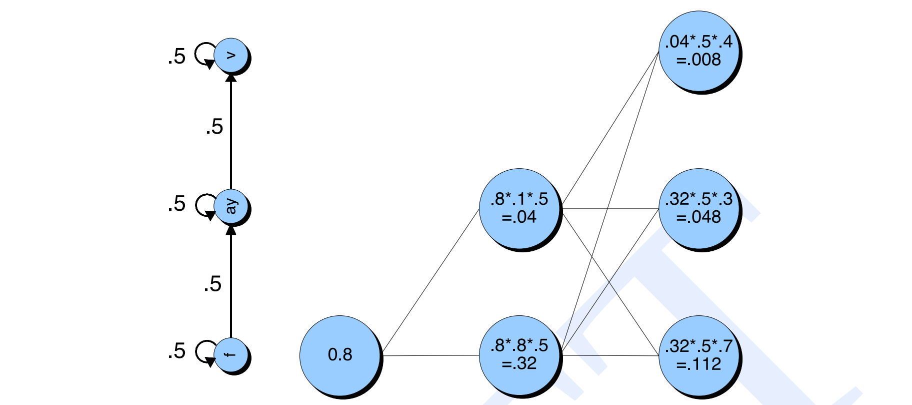


Figure 9.27 The first 3 time-steps of the viterbi trellis computation for the word *five*. The *A* transition probabilities are shown along the left edge; the *B* observation likelihoods are shown in Fig. 9.28. In this computation we make the simplifying assumption that the probability of starting in state 1 (phone [f]) is 1.0

V	0	0	0.008	0.0072	0.00672	0.00403	0.00188	0.00161	0.000667	0.000493
AY	0	0.04	0.048	0.0448	0.0269	0.0125	0.00538	0.00167	0.000428	8.78e-05
F	0.8	0.32	0.112	0.0224	0.00448	0.000896	0.000179	4.48e-05	1.12e-05	2.8e-06
Time	1	2	3	4	5	6	7	8	9	10
B	*f* 0.8 *ay* 0.1 *v* 0.6 *p* 0.4 *iy* 0.1	*f* 0.8 *ay* 0.1 *v* 0.6 *p* 0.4 *iy* 0.1	*f* 0.7 *ay* 0.3 *v* 0.4 *p* 0.2 *iy* 0.3	*f* 0.4 *ay* 0.8 *v* 0.3 *p* 0.1 *iy* 0.6	*f* 0.4 *ay* 0.8 *v* 0.3 *p* 0.1 *iy* 0.6	*f* 0.4 *ay* 0.8 *v* 0.3 *p* 0.1 *iy* 0.6	*f* 0.4 *ay* 0.8 *v* 0.3 *p* 0.1 *iy* 0.6	*f* 0.5 *ay* 0.6 *v* 0.6 *p* 0.1 *iy* 0.5	*f* 0.5 *ay* 0.5 *v* 0.8 *p* 0.3 *iy* 0.5	*f* 0.5 *ay* 0.4 *v* 0.9 *p* 0.3 *iy* 0.4

Figure 9.28 The Viterbi trellis for 10 frames of the word *five*, consisting of 3 emitting states (*f*, *ay*, *v*), plus non-emitting start and end states (not shown). The bottom half of the table gives part of the *B* observation likelihood vector for the observation *o* at each frame, $p(o|q)$ for each phone *q*. *B* values are created by hand for pedagogical purposes. This table assumes the HMM structure for *five* shown in Fig. 9.24, each emitting state having a .5 loopback probability.

transitioning from the end of one word to the beginning of another word. The digit HMM model in Fig. 9.22 showed that we could just treat each word as independent, and use only the unigram probability. Higher-order *N*-grams are much more common. Fig. 9.29, for example, shows an augmentation of the digit HMM with bigram probabilities.

A schematic of the HMM trellis for such a multi-word decoding task is shown in Fig. 9.30. The intraword transitions are exactly as shown in Fig. 9.27. But now between words we've added a transition. The transition probability on this arc, rather than coming from the *A* matrix inside each word, comes from the language model $P(W)$.

Once the entire Viterbi trellis has been computed for the utterance, we can start from the most-probable state at the final time step and follow the backtrace pointers

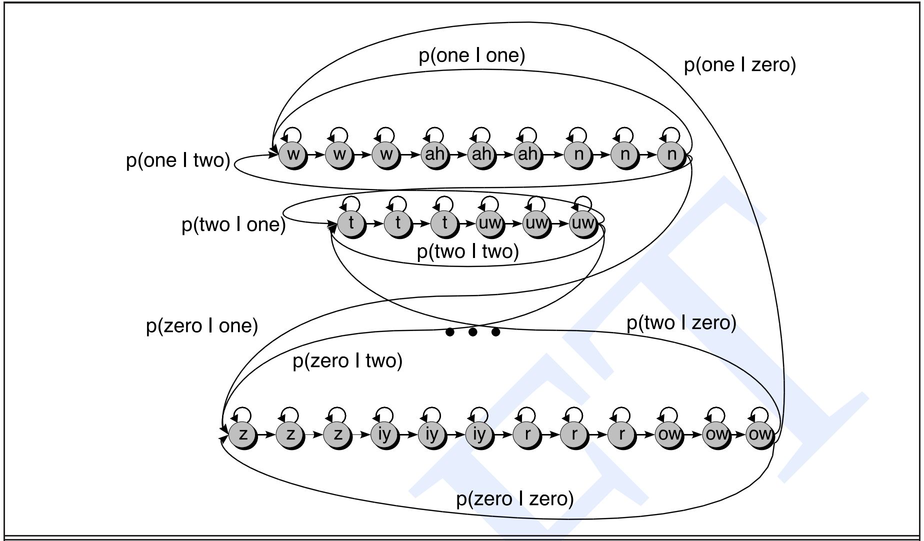


Figure 9.29 A bigram grammar network for the digit recognition task. The bigrams give the probability of transitioning from the end of one word to the beginning of the next.

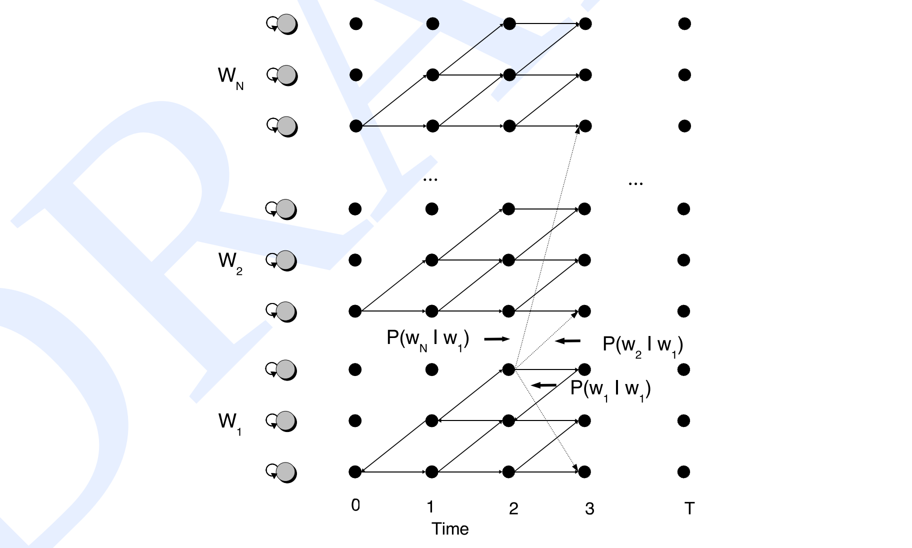


Figure 9.30 The HMM Viterbi trellis for a bigram language model. The intraword transitions are exactly as shown in Fig. 9.27. Between words, a potential transition is added (shown as a dotted line) from the end state of each word to the beginning state of every word, labeled with the bigram probability of the word pair.

backwards to get the most probable string of states, and hence the most probable string of words. Fig. 9.31 shows the backtrace pointers being followed back from the best state, which happens to be at w_2, eventually through w_N and w_1, resulting in the final word string $w_1 w_N \cdots w_2$.

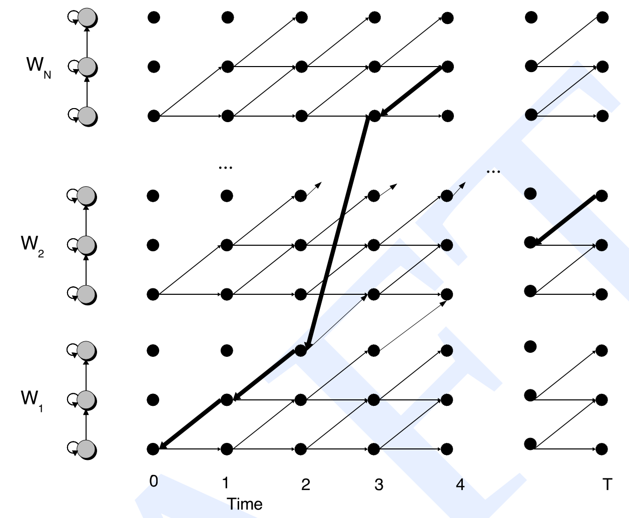


Figure 9.31 Viterbi backtrace in the HMM trellis. The backtrace starts in the final state, and results in a best phone string from which a word string is derived.

The Viterbi algorithm is much more efficient than exponentially running the forward algorithm for each possible word string. Nonetheless, it is still slow, and much modern research in speech recognition has focused on speeding up the decoding process. For example in practice in large-vocabulary recognition we do not consider all possible words when the algorithm is extending paths from one state-column to the next. Instead, low-probability paths are **pruned** at each time step and not extended to the next state column.

PRUNING

BEAM SEARCH

This pruning is usually implemented via **beam search** (Lowerre, 1968). In beam search, at each time t, we first compute the probability of the best (most-probable) state/path D. We then prune away any state which is worse than D by some fixed threshold (**beam width**) θ. We can talk about beam-search in both the probability and negative log probability domain. In the probability domain any path/state whose probability is less than $\theta * D$ is pruned away; in the negative log domain, any path whose cost is greater then $\theta + D$ is pruned. Beam search is implemented by keeping for each time step an **active list** of states. Only transitions from these words are extended when moving to the next time step.

BEAM WIDTH

ACTIVE LIST

Making this beam search approximation allows a significant speed-up at the cost of a degradation to the decoding performance. Huang et al. (2001) suggest that empirically a beam size of 5-10% of the search space is sufficient; 90-95% of the states are thus not considered. Because in practice most implementations of Viterbi use beam search, some of the literature uses the term **beam search** or **time-synchronous beam search** instead of Viterbi.

9.7 Embedded Training

We turn now to see how an HMM-based speech recognition system is trained. We've already seen some aspects of training. In Ch. 4 we showed how to train a language model. In Sec. 9.4, we saw how GMM acoustic models are trained by augmenting the EM algorithm to deal with training the means, variances, and weights. We also saw how posterior AM classifiers like SVMs or neural nets could be trained, although for neural nets we haven't yet seen how we get training data in which each frame is labeled with a phone identity.

In this section we complete the picture of HMM training by showing how this augmented EM training algorithm fits into the whole process of training acoustic models. For review, here are the three components of the **acoustic model**:

$Q = q_1 q_2 \dots q_N$ the **subphones** represented as a set of **states**

$A = a_{01} a_{02} \dots a_{n1} \dots a_{nn}$ a **subphone transition probability matrix** A, each a_{ij} representing the probability for each subphone of taking a **self-loop** or going to the next subphone. Together, Q and A implement a **pronunciation lexicon**, an HMM state graph structure for each word that the system is capable of recognizing.

$B = b_i(o_t)$ A set of **observation likelihoods:**, also called **emission probabilities**, each expressing the probability of a cepstral feature vector (observation o_t) being generated from subphone state i.

We will assume that the pronunciation lexicon, and thus the basic HMM state graph structure for each word, is pre-specified as the simple linear HMM structures with loopbacks on each state that we saw in Fig. 9.7 and Fig. 9.22. In general, speech recognition systems do not attempt to learn the structure of the individual word HMMs. Thus we only need to train the B matrix, and we need to train the probabilities of the non-zero (self-loop and next-subphone) transitions in the A matrix. All the other probabilities in the A matrix are set to zero and never change.

The simplest possible training method, is **hand-labeled isolated word** training, in which we train separate the B and A matrices for the HMMs for each word based on hand-aligned training data. We are given a training corpus of digits, where each instance of a spoken digit is stored in a wavefile, and with the start and end of each word and phone hand-segmented. Given such a hand-labeled database, we can compute the B Gaussians observation likelihoods and the A transition probabilities by merely counting in the training data! The A transition probability are specific to each word, but the B Gaussians would be shared across words if the same phone occurred in multiple words.

Unfortunately, hand-segmented training data is rarely used in training systems for continuous speech. One reason is that it is very expensive to use humans to hand-label phonetic boundaries; it can take up to 400 times real time (i.e. 400 labeling hours to label each 1 hour of speech). Another reason is that humans don't do phonetic labeling very well for units smaller than the phone; people are bad at consistently finding the boundaries of subphones. ASR systems aren't better than humans at finding boundaries, but their errors are at least consistent between the training and test sets.

EMBEDDED TRAINING

For this reason, speech recognition systems train each phone HMM embedded in an entire sentence, and the segmentation and phone alignment are done automatically as part of the training procedure. This entire acoustic model training process is therefore called **embedded training**. Hand phone segmentation do still play some role, however, for example for bootstrapping initial systems for discriminative (SVM; non-Gaussian) likelihood estimators, or for tasks like phone recognition.

In order to train a simple digits system, we'll need a training corpus of spoken digit sequences. For simplicity assume that the training corpus is separated into separate wavefiles, each containing a sequence of spoken digits. For each wavefile, we'll need to know the correct sequence of digit words. We'll thus associate with each wavefile a transcription (a string of words). We'll also need a pronunciation lexicon and a phoneset, defining a set of (untrained) phone HMMs. From the transcription, lexicon, and phone HMMs, we can build a "whole sentence" HMM for each sentence, as shown in Fig. 9.32.

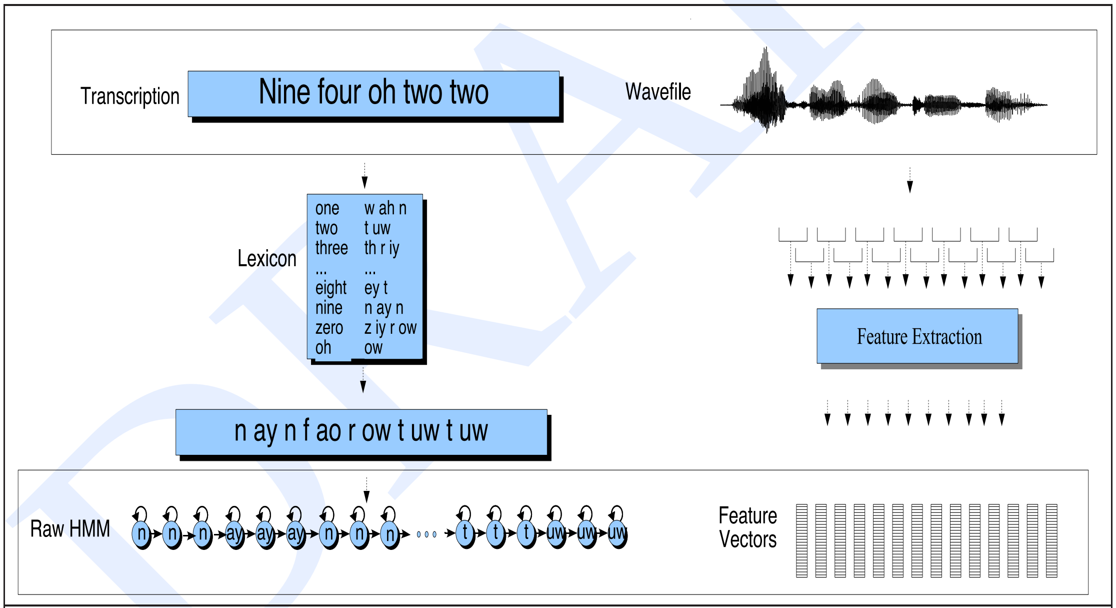


Figure 9.32 The input to the embedded training algorithm; a wavefile of spoken digits with a corresponding transcription. The transcription is converted into a raw HMM, ready to be aligned and trained against the cepstral features extracted from the wavefile.

We are now ready to train the transition matrix A and output likelihood estimator B for the HMMs. The beauty of the Baum-Welch-based paradigm for embedded training of HMMs is that this is all the training data we need. In particular, we don't need phonetically transcribed data. We don't even need to know where each word starts and ends. The Baum-Welch algorithm will sum over all possible segmentations of words and phones, using $\xi_j(t)$, the probability of being in state j at time t and generating the observation sequence O.

FLAT START

We will, however, need an initial estimate for the transition and observation probabilities a_{ij} and $b_j(o_t)$. The simplest way to do this is with a **flat start**. In flat start, we first set to zero any HMM transitions that we want to be 'structurally zero', such as transitions from later phones back to earlier phones. The γ probability computation in Baum-Welch includes the previous value of a_{ij}, so those zero values will never change. Then we make all the rest of the (non-zero) HMM transitions equiprobable. Thus the two transitions out of each state (the self-loop and the transition to the following subphone) each would have a probability of 0.5. For the Gaussians, a flat start initializes the mean and variance for each Gaussian identically, to the global mean and variance for the entire training data.

Now we have initial estimates for the A and B probabilities. For a standard Gaussian HMM system, we now run multiple iterations of the Baum-Welch algorithm on the entire training set. Each iteration modifies the HMM parameters, and we stop when the system converges. During each iteration, as discussed in Ch. 6, we compute the forward and backward probabilities for each sentence given the initial A and B probabilities, and use them to re-estimate the A and B probabilities. We also apply the various modifications to EM discussed in the previous section to correctly update the Gaussian means and variances for multivariate Gaussians. We will discuss in Sec. **??** in Ch. 10 how to modify the embedded training algorithm to handle mixture Gaussians.

In summary, the basic **embedded training procedure** is as follows:

Given: phoneset, pronunciation lexicon, and the transcribed wavefiles

1. Build a "whole sentence" HMM for each sentence, as shown in Fig. 9.32.
2. Initialize A probabilities to 0.5 (for loop-backs or for the correct next subphone) or to zero (for all other transitions).
3. Initialize B probabilities by setting the mean and variance for each Gaussian to the global mean and variance for the entire training set.
4. Run multiple iterations of the Baum-Welch algorithm.

The Baum-Welch algorithm is used repeatedly as a component of the embedded training process. Baum-Welch computes $\xi_t(i)$, the probability of being in state i at time t, by using forward-backward to sum over all possible paths that were in state i emitting symbol o_t at time t. This lets us accumulate counts for re-estimating the emission probability $b_j(o_t)$ from all the paths that pass through state j at time t. But Baum-Welch itself can be time-consuming.

VITERBI TRAINING

There is an efficient approximation to Baum-Welch training that makes use of the Viterbi algorithm. In **Viterbi training**, instead of accumulating counts by a sum over all paths that pass through a state j at time t, we approximate this by only choosing the Viterbi (most-probable) path. Thus instead of running EM at every step of the embedded training, we repeatedly run Viterbi.

FORCED ALIGNMENT

Running the Viterbi algorithm over the training data in this way is called **forced Viterbi alignment** or just **forced alignment**. In Viterbi training (unlike in Viterbi decoding on the test set) we know which word string to assign to each observation sequence, So we can ‘force’ the Viterbi algorithm to pass through certain words, by setting the a_{ij}s appropriately. A forced Viterbi is thus a simplification of the regular Viterbi decoding algorithm, since it only has to figure out the correct state (subphone) sequence, but doesn’t have to discover the word sequence. The result is a **forced alignment**: the single best state path corresponding to the training observation sequence. We can now use this alignment of HMM states to observations to accumulate counts for re-estimating the HMM parameters. We saw earlier that forcd alignment can also be used in other speech applications like text-to-speech, whenever we have a word transcript and a wavefile in which we want to find boundaries.

The equations for retraining a (non-mixture) Gaussian from a Viterbi alignment are as follows:

$$\hat{\mu}_i = \frac{1}{T}\sum_{t=1}^{T} o_t \text{ s.t. } q_t \text{ is state } i \tag{9.55}$$

$$\hat{\sigma}_j^2 = \frac{1}{T}\sum_{t=1}^{T} (o_t - \mu_i)^2 \text{ s.t. } q_t \text{ is state } i \tag{9.56}$$

We saw these equations already, as (9.27) and (9.28) on page 25, when we were ‘imagining the simpler situation of a completely labeled training set’.

It turns out that this forced Viterbi algorithm is also used in the embedded training of hybrid models like HMM/MLP or HMM/SVM systems. We begin with an untrained MLP, and using its noisy outputs as the B values for the HMM, perform a forced Viterbi alignment of the training data. This alignment will be quite errorful, since the MLP was random. Now this (quite errorful) Viterbi alignment give us a labeling of feature vectors with phone labels. We use this labeling to retrain the MLP. The counts of the transitions which are taken in the forced alignments can be used to estimate the HMM transition probabilities. We continue this hill-climbing process of neural-net training and Viterbi alignment until the HMM parameters begin to converge.

9.8 Evaluation: Word Error Rate

WORD ERROR

The standard evaluation metric for speech recognition systems is the **word error** rate. The word error rate is based on how much the word string returned by the recognizer (often called the **hypothesized** word string) differs from a correct or **reference** transcription. Given such a correct transcription, the first step in computing word error is to compute the **minimum edit distance** in words between the hypothesized and correct strings, as described in Ch. 3. The result of this computation will be the minimum number of word **substitutions**, word **insertions**, and word **deletions** necessary to map between the correct and hypothesized strings. The word error rate (WER) is then defined as follows (note that because the equation includes insertions, the error rate can be greater than 100%):

$$\text{Word Error Rate} = 100 \times \frac{\text{Insertions} + \text{Substitutions} + \text{Deletions}}{\text{Total Words in Correct Transcript}}$$

We sometimes also talk about the SER (Sentence Error Rate), which tells us how many sentences had at least one error:

$$\text{Sentence Error Rate} = 100 \times \frac{\text{\# of sentences with at least one word error}}{\text{total \# of sentences}}$$

ALIGNMENTS

Here is an example of the **alignments** between a reference and a hypothesized utterance from the CALLHOME corpus, showing the counts used to compute the word error rate:

REF:	i	***	**	UM	the	PHONE	IS	i	LEFT	THE	portable	****	PHONE	UPSTAIRS	last	night
HYP:	i	GOT	IT	TO	the	*****	FULLEST	i	LOVE	TO	portable	FORM	OF	STORES	last	night
Eval:		I	I	S		D	S		S	S		I	S	S		

This utterance has six substitutions, three insertions, and one deletion:

$$\text{Word Error Rate} = 100\frac{6+3+1}{13} = 76.9\%$$

The standard method for implementing minimum edit distance and computing word error rates is a free script called `sclite`, available from the National Institute of Standards and Technologies (NIST) (NIST, 2005). `sclite` is given a series of reference (hand-transcribed, gold-standard) sentences and a matching set of hypothesis sentences. Besides performing alignments, and computing word error rate, sclite performs a number of other useful tasks. For example, it gives useful information for **error analysis**, such as confusion matrices showing which words are often misrecognized for others, and gives summary statistics of words which are often inserted or deleted. `sclite` also gives error rates by speaker (if sentences are labeled for speaker id), as well as useful statistics like the **sentence error rate**, the percentage of sentences with at least one word error.

SENTENCE ERROR RATE

Finally, `sclite` can be used to compute significance tests. Suppose we make some changes to our ASR system and find that our word error rate has decreased by 1%. In order to know if our changes really improved things, we need a statistical test to make sure that the 1% difference is not just due to chance. The standard statistical test for determining if two word error rates are different is the Matched-Pair Sentence Segment Word Error (MAPSSWE) test, which is also available in `sclite` (although the **McNemar test** is sometimes used as well).

MCNEMAR TEST

The MAPSSWE test is a parametric test that looks at the difference between the number of word errors the two systems produce, averaged across a number of segments. The segments may be quite short or as long as an entire utterance; in general we want to have the largest number of (short) segments in order to justify the normality assumption and for maximum power. The test requires that the errors in one segment be statistically independent of the errors in another segment. Since ASR systems tend to use trigram LMs, this can be approximated by defining a segment as a region bounded on both sides by words that both recognizers get correct (or turn/utterance boundaries).

Here's an example from NIST (2007) with four segments, labeled in roman numerals:

```
         I             II             III                 IV
REF:  |it was|the best|of|times it|was the worst|of times|  |it was
      |      |        |  |        |             |        |  |
SYS A:|ITS   |the best|of|times it|IS the worst |of times|OR|it was
      |      |        |  |        |             |        |  |
SYS B:|it was|the best|  |times it|WON the TEST |of times|  |it was
```

In region I, system A has 2 errors (a deletion and an insertion) and system B has 0; in region III system A has 1 (substitution) error and system B has 2. Let's define a sequence of variables Z representing the difference between the errors in the two systems as follows:

N_A^i the number of errors made on segment i by system A
N_B^i the number of errors made on segment i by system B
Z $N_A^i - N_B^i, i = 1, 2, \cdots, n$ where n is the number of segments

For example in the example above the sequence of Z values is $\{2, -1, -1, 1\}$. Intuitively, if the two systems are identical, we would expect the average difference, i.e. the average of the Z values, to be zero. If we call the true average of the differences mu_z, we would thus like to know whether $mu_z = 0$. Following closely the original proposal and notation of Gillick and Cox (1989), we can estimate the true average from our limited sample as $\hat{\mu}_z = \sum_{i=1}^{n} Z_i/n$.

The estimate of the variance of the Z_i's is:

$$\sigma_z^2 = \frac{1}{n-1}\sum_{i=1}^{n}(Z_i - \mu_z)^2 \tag{9.57}$$

Let

$$W = \frac{\hat{\mu}_z}{\sigma_z/\sqrt{n}} \tag{9.58}$$

For a large enough n (> 50) W will approximately have a normal distribution with unit variance. The null hypothesis is $H_0 : \mu_z = 0$, and it can thus be rejected if $2 * P(Z \geq |w|) \leq 0.05$ (two-tailed) or $P(Z \geq |w|) \leq 0.05$ (one-tailed). where Z is standard normal and w is the realized value W; these probabilities can be looked up in the standard tables of the normal distribution.

Could we improve on word error rate as a metric? It would be nice, for example, to have something which didn't give equal weight to every word, perhaps valuing content words like *Tuesday* more than function words like *a* or *of*. While researchers generally agree that this would be a good idea, it has proved difficult to agree on a metric that works in every application of ASR. For dialogue systems, however, where the desired semantic output is more clear, a metric called *concept error rate* has proved extremely useful, and will be discussed in Ch. 24 on page **??**.

9.9 Summary

Together with Ch. 4 and Ch. 6, this chapter introduced the fundamental algorithms for addressing the problem of **Large Vocabulary Continuous Speech Recognition**.

- The input to a speech recognizer is a series of acoustic waves. The **waveform**, **spectrogram** and **spectrum** are among the visualization tools used to understand the information in the signal.
- In the first step in speech recognition, sound waves are **sampled**, **quantized**, and converted to some sort of **spectral representation**; A commonly used spectral representation is the **mel cepstrum** or **MFCC** which provides a vector of features for each frame of the input.
- GMM acoustic models are used to estimate the **phonetic likelihoods** (also called **observation likelihoods**) of these **feature vectors** for each frame.
- **Decoding** or **search** or **inference** is the process of finding the optimal sequence of model states which matches a sequence of input observations. (The fact that there are three terms for this process is a hint that speech recognition is inherently inter-disciplinary, and draws its metaphors from more than one field; **decoding** comes from information theory, and **search** and **inference** from artificial intelligence).
- We introduced two decoding algorithms: time-synchronous **Viterbi** decoding (which is usually implemented with pruning and can then be called **beam search**) and **stack** or **A*** decoding. Both algorithms take as input a sequence of cepstral feature vectors, a GMM acoustic model, and an N-gram language model, and produce a string of words.
- The **embedded training** paradigm is the normal method for training speech recognizers. Given an initial lexicon with hand-built pronunciation structures, it will train the HMM transition probabilities and the HMM observation probabilities.

Bibliographical and Historical Notes

The first machine which recognized speech was probably a commercial toy named “Radio Rex” which was sold in the 1920s. Rex was a celluloid dog that moved (via a spring) when the spring was released by 500 Hz acoustic energy. Since 500 Hz is roughly the first formant of the vowel [eh] in “Rex”, the dog seemed to come when he was called (David and Selfridge, 1962).

By the late 1940s and early 1950s, a number of machine speech recognition systems had been built. An early Bell Labs system could recognize any of the 10 digits from a single speaker (Davis et al., 1952). This system had 10 speaker-dependent stored patterns, one for each digit, each of which roughly represented the first two vowel formants in the digit. They achieved 97–99% accuracy by choosing the pattern which had the highest relative correlation coefficient with the input. Fry (1959) and Denes (1959) built a phoneme recognizer at University College, London, which recognized four vowels and nine consonants based on a similar pattern-recognition principle. Fry and Denes’s system was the first to use phoneme transition probabilities to constrain the recognizer.

The late 1960s and early 1970s produced a number of important paradigm shifts. First were a number of feature-extraction algorithms, include the efficient Fast Fourier

Transform (FFT) (Cooley and Tukey, 1965), the application of cepstral processing to speech (Oppenheim et al., 1968), and the development of LPC for speech coding (Atal and Hanauer, 1971). Second were a number of ways of handling **warping**; stretching or shrinking the input signal to handle differences in speaking rate and segment length when matching against stored patterns. The natural algorithm for solving this problem was dynamic programming, and, as we saw in Ch. 6, the algorithm was reinvented multiple times to address this problem. The first application to speech processing was by Vintsyuk (1968), although his result was not picked up by other researchers, and was reinvented by Velichko and Zagoruyko (1970) and Sakoe and Chiba (1971) (and (1984)). Soon afterward, Itakura (1975) combined this dynamic programming idea with the LPC coefficients that had previously been used only for speech coding. The resulting system extracted LPC features for incoming words and used dynamic programming to match them against stored LPC templates. The non-probabistic use of dynamic programming to match a template against incoming speech is called **dynamic time warping**.

WARPING

DYNAMIC TIME WARPING

The third innovation of this period was the rise of the HMM. Hidden Markov Models seem to have been applied to speech independently at two laboratories around 1972. One application arose from the work of statisticians, in particular Baum and colleagues at the Institute for Defense Analyses in Princeton on HMMs and their application to various prediction problems (Baum and Petrie, 1966; Baum and Eagon, 1967). James Baker learned of this work and applied the algorithm to speech processing (Baker, 1975) during his graduate work at CMU. Independently, Frederick Jelinek, Robert Mercer, and Lalit Bahl (drawing from their research in information-theoretical models influenced by the work of Shannon (1948)) applied HMMs to speech at the IBM Thomas J. Watson Research Center (Jelinek et al., 1975). IBM's and Baker's systems were very similar, particularly in their use of the Bayesian framework described in this chapter. One early difference was the decoding algorithm; Baker's DRAGON system used Viterbi (dynamic programming) decoding, while the IBM system applied Jelinek's stack decoding algorithm (Jelinek, 1969). Baker then joined the IBM group for a brief time before founding the speech-recognition company Dragon Systems. The HMM approach to speech recognition would turn out to completely dominate the field by the end of the century; indeed the IBM lab was the driving force in extending statistical models to natural language processing as well, including the development of class-based *N*-grams, HMM-based part-of-speech tagging, statistical machine translation, and the use of entropy/perplexity as an evaluation metric.

The use of the HMM slowly spread through the speech community. One cause was a number of research and development programs sponsored by the Advanced Research Projects Agency of the U.S. Department of Defense (ARPA). The first five-year program starting in 1971, and is reviewed in Klatt (1977). The goal of this first program was to build speech understanding systems based on a few speakers, a constrained grammar and lexicon (1000 words), and less than 10% semantic error rate. Four systems were funded and compared against each other: the System Development Corporation (SDC) system, Bolt, Beranek & Newman (BBN)'s HWIM system, Carnegie-Mellon University's Hearsay-II system, and Carnegie-Mellon's Harpy system (Lowerre, 1968). The Harpy system used a simplified version of Baker's HMM-based DRAGON system and was the best of the tested systems, and according to Klatt the only one to meet the original goals of the ARPA project (with a semantic accuracy rate of 94% on a simple task).

Beginning in the mid-1980s, ARPA funded a number of new speech research programs. The first was the “Resource Management” (RM) task (Price et al., 1988), which like the earlier ARPA task involved transcription (recognition) of read-speech (speakers reading sentences constructed from a 1000-word vocabulary) but which now included a component that involved speaker-independent recognition. Later tasks included recognition of sentences read from the Wall Street Journal (WSJ) beginning with limited systems of 5,000 words, and finally with systems of unlimited vocabulary (in practice most systems use approximately 60,000 words). Later speech-recognition tasks moved away from read-speech to more natural domains; the Broadcast News domain (LDC, 1998; Graff, 1997) (transcription of actual news broadcasts, including quite difficult passages such as on-the-street interviews) and the Switchboard, CALLHOME, CALLFRIEND, and Fisher domains (Godfrey et al., 1992; Cieri et al., 2004) (natural telephone conversations between friends or strangers) . The Air Traffic Information System (ATIS) task (Hemphill et al., 1990) was an earlier speech understanding task whose goal was to simulate helping a user book a flight, by answering questions about potential airlines, times, dates, and so forth.

BAKE-OFF

Each of the ARPA tasks involved an approximately annual **bake-off** at which all ARPA-funded systems, and many other ‘volunteer’ systems from North American and Europe, were evaluated against each other in terms of word error rate or semantic error rate. In the early evaluations, for-profit corporations did not generally compete, but eventually many (especially IBM and ATT) competed regularly. The ARPA competitions resulted in widescale borrowing of techniques among labs, since it was easy to see which ideas had provided an error-reduction the previous year, and were probably an important factor in the eventual spread of the HMM paradigm to virtual every major speech recognition lab. The ARPA program also resulted in a number of useful databases, originally designed for training and testing systems for each evaluation (TIMIT, RM, WSJ, ATIS, BN, CALLHOME, Switchboard, Fisher) but then made available for general research use.

SPEAKER IDENTIFICATION

SPEAKER VERIFICATION

LANGUAGE IDENTIFICATION

Speech research includes a number of areas besides speech recognition; we already saw computational phonology in Ch. 7, speech synthesis in Ch. 8, and we will discuss spoken dialogue systems in Ch. 24. Another important area is **speaker identification** and **speaker verification**, in which we identify a speaker (for example for security when accessing personal information over the telephone) (Reynolds and Rose, 1995; Shriberg et al., 2005; Doddington, 2001). This task is related to **language identification**, in which we are given a wavefile and have to identify which language is being spoken; this is useful for automatically directing callers to human operators that speak appropriate languages.

There are a number of textbooks and reference books on speech recognition that are good choices for readers who seek a more in-depth understanding of the material in this chapter: Huang et al. (2001) is by far the most comprehensive and up-to-date reference volume and is highly recommended. Jelinek (1997), Gold and Morgan (1999), and Rabiner and Juang (1993) are good comprehensive textbooks. The last two textbooks also have discussions of the history of the field, and together with the survey paper of Levinson (1995) have influenced our short history discussion in this chapter. Our description of the forward-backward algorithm was modeled after Rabiner (1989), and we were also influenced by another useful tutorial paper, Knill and Young (1997). Research in the speech recognition field often appears in the proceedings of the annual INTERSPEECH conference, (which is called ICSLP and EUROSPEECH in alternate years) as well as the annual IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP). Journals include *Speech Communication*, *Computer Speech and Language*, the *IEEE Transactions on Audio, Speech, and Language Processing*, and the *ACM Transactions on Speech and Language Processing*.

EXERCISES

9.1 Analyze each of the errors in the incorrectly recognized transcription of "um the phone is I left the..." on page 46. For each one, give your best guess as to whether you think it is caused by a problem in signal processing, pronunciation modeling, lexicon size, language model, or pruning in the decoding search.

LOGPROB

9.2 In practice, speech recognizers do all their probability computation using the **log probability** (or **logprob**) rather than actual probabilities. This helps avoid underflow for very small probabilities, but also makes the Viterbi algorithm very efficient, since all probability multiplications can be implemented by adding log probabilities. Rewrite the pseudocode for the Viterbi algorithm in Fig. 9.26 on page 38 to make use of logprobs instead of probabilities.

9.3 Now modify the Viterbi algorithm in Fig. 9.26 on page 38 to implement the beam search described on page 41. Hint: You will probably need to add in code to check whether a given state is at the end of a word or not.

9.4 Finally, modify the Viterbi algorithm in Fig. 9.26 on page 38 with more detailed pseudocode implementing the array of backtrace pointers.

9.5 Using the tutorials available as part of a publicly available recognizer like HTK or Sonic, build a digit recognizer.

9.6 Take the digit recognizer above and dump the phone likelihoods for a sentence. Now take your implementation of the Viterbi algorithm and show that you can successfully decode these likelihoods.

Atal, B. S. and Hanauer, S. (1971). Speech analysis and synthesis by prediction of the speech wave. *Journal of the Acoustical Society of America*, *50*, 637–655.

Baker, J. K. (1975). The DRAGON system – An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *ASSP-23*(1), 24–29.

Baum, L. E. and Eagon, J. A. (1967). An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, *73*(3), 360–363.

Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite-state Markov chains. *Annals of Mathematical Statistics*, *37*(6), 1554–1563.

Bayes, T. (1763). *An Essay Toward Solving a Problem in the Doctrine of Chances*, Vol. 53. Reprinted in *Facsimiles of two papers by Bayes*, Hafner Publishing Company, New York, 1963.

Bledsoe, W. W. and Browning, I. (1959). Pattern recognition and reading by machine. In *1959 Proceedings of the Eastern Joint Computer Conference*, pp. 225–232. Academic.

Cieri, C., Miller, D., and Walker, K. (2004). The Fisher Corpus: a Resource for the Next Generations of Speech-to-Text. In *LREC-04*.

CMU (1993). The Carnegie Mellon Pronouncing Dictionary v0.1. Carnegie Mellon University.

Cohen, P. R., Johnston, M., McGee, D., Oviatt, S. L., Clow, J., and Smith, I. (1998). The efficiency of multimodal interaction: a case study. In *ICSLP-98*, Sydney, Vol. 2, pp. 249–252.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, *19*(90), 297–301.

David, Jr., E. E. and Selfridge, O. G. (1962). Eyes and ears for computers. *Proceedings of the IRE (Institute of Radio Engineers)*, *50*, 1093–1101.

Davis, K. H., Biddulph, R., and Balashek, S. (1952). Automatic recognition of spoken digits. *Journal of the Acoustical Society of America*, *24*(6), 637–642.

Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *28*(4), 357–366.

Denes, P. (1959). The design and operation of the mechanical speech recognizer at University College London. *Journal of the British Institution of Radio Engineers*, *19*(4), 219–234. Appears together with companion paper (Fry 1959).

Deng, L. and Huang, X. (2004). Challenges in adopting speech recognition. *Communications of the ACM*, *47*(1), 69–75.

Doddington, G. (2001). Speaker recognition based on idiolectal differences between speakers. In *EUROSPEECH-01*, Budapest, pp. 2521–2524.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification*. Wiley-Interscience Publication.

Fry, D. B. (1959). Theoretical aspects of mechanical speech recognition. *Journal of the British Institution of Radio Engineers*, *19*(4), 211–218. Appears together with companion paper (Denes 1959).

Gillick, L. and Cox, S. (1989). Some statistical issues in the comparison of speech recognition algorithms. In *IEEE ICASSP-89*, pp. 532–535.

Godfrey, J., Holliman, E., and McDaniel, J. (1992). SWITCHBOARD: Telephone speech corpus for research and development. In *IEEE ICASSP-92*, San Francisco, pp. 517–520. IEEE.

Gold, B. and Morgan, N. (1999). *Speech and Audio Signal Processing*. Wiley Press.

Graff, D. (1997). The 1996 Broadcast News speech and language-model corpus. In *Proceedings DARPA Speech Recognition Workshop*, Chantilly, VA, pp. 11–14. Morgan Kaufmann.

Gray, R. M. (1984). Vector quantization. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *ASSP-1*(2), 4–29.

Hemphill, C. T., Godfrey, J., and Doddington, G. (1990). The ATIS spoken language systems pilot corpus. In *Proceedings DARPA Speech and Natural Language Workshop*, Hidden Valley, PA, pp. 96–101. Morgan Kaufmann.

Huang, X., Acero, A., and Hon, H.-W. (2001). *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall, Upper Saddle River, NJ.

Itakura, F. (1975). Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *ASSP-32*, 67–72.

Jelinek, F. (1969). A fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, *13*, 675–685.

Jelinek, F. (1997). *Statistical Methods for Speech Recognition*. MIT Press.

Jelinek, F., Mercer, R. L., and Bahl, L. R. (1975). Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory*, *IT-21*(3), 250–256.

Klatt, D. H. (1977). Review of the ARPA speech understanding project. *Journal of the Acoustical Society of America*, *62*(6), 1345–1366.

Knill, K. and Young, S. J. (1997). Hidden Markov Models in speech and language processing. In Young, S. J. and Bloothooft, G. (Eds.), *Corpus-based Methods in Language and Speech Processing*, pp. 27–68. Kluwer, Dordrecht.

LDC (1998). *LDC Catalog: Hub4 project*. University of Pennsylvania. `www.ldc.upenn.edu/Catalog/LDC98S71.html` or `www.ldc. upenn.edu/Catalog/Hub4.html`.

Levinson, S. E. (1995). Structural methods in automatic speech recognition. *Proceedings of the IEEE*, *73*(11), 1625–1650.

Lowerre, B. T. (1968). *The Harpy Speech Recognition System*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA.

Mosteller, F. and Wallace, D. L. (1964). *Inference and Disputed Authorship: The Federalist*. Springer-Verlag. A second edition appeared in 1984 as *Applied Bayesian and Classical Inference*.

NIST (2005). Speech recognition scoring toolkit (sctk) version 2.1. Available at http://www.nist.gov/speech/tools/.

NIST (2007). Matched Pairs Sentence-Segment Word Error (MAPSSWE) Test. `http://www.nist.gov/speech/tests/sigtests/mapsswe.htm`.

Oppenheim, A. V., Schafer, R. W., and Stockham, T. G. J. (1968). Nonlinear filtering of multiplied and convolved signals. *Proceedings of the IEEE*, *56*(8), 1264–1291.

Price, P. J., Fisher, W., Bernstein, J., and Pallet, D. (1988). The DARPA 1000-word resource management database for continuous speech recognition. In *IEEE ICASSP-88*, New York, Vol. 1, pp. 651–654.

Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286.

Rabiner, L. R. and Juang, B. H. (1993). *Fundamentals of Speech Recognition*. Prentice Hall.

Reynolds, D. A. and Rose, R. C. (1995). Robust text- independent speaker identification using gaussian mixture speaker models. *IEEE Transactions on Speech and Audio Processing*, *3*(1), 72–83.

Sakoe, H. and Chiba, S. (1971). A dynamic programming approach to continuous speech recognition. In *Proceedings of the Seventh International Congress on Acoustics, Budapest*, Budapest, Vol. 3, pp. 65–69. Akadémiai Kiadó.

Sakoe, H. and Chiba, S. (1984). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *ASSP-26*(1), 43–49.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, *27*(3), 379–423. Continued in the following volume.

Shriberg, E., Ferrer, L., Kajarekar, S., Venkataraman, A., and Stolcke, A. (2005). Modeling prosodic feature sequences for speaker recognition. *Speech Communication*, *46*(3-4), 455–472.

Stevens, S. S. and Volkmann, J. (1940). The relation of pitch to frequency: A revised scale. *The American Journal of Psychology*, *53*(3), 329–353.

Stevens, S. S., Volkmann, J., and Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *Journal of the Acoustical Society of America*, *8*, 185–190.

Stolcke, A. (2002). Srilm - an extensible language modeling toolkit. In *ICSLP-02*, Denver, CO.

Taylor, P. (2008). Text-to-speech synthesis. Manuscript.

Tomokiyo, L. M. (2001). *Recognizing non-native speech: Characterizing and adapting to non-native usage in speech recognition*. Ph.D. thesis, Carnegie Mellon University.

Velichko, V. M. and Zagoruyko, N. G. (1970). Automatic recognition of 200 words. *International Journal of Man-Machine Studies*, *2*, 223–234.

Vintsyuk, T. K. (1968). Speech discrimination by dynamic programming. *Cybernetics*, *4*(1), 52–57. Russian Kibernetika 4(1):81-88 (1968).

Young, S. J., Evermann, G., Gales, M., Hain, T., Kershaw, D., Moore, G., Odell, J. J., Ollason, D., Povey, D., Valtchev, V., and Woodland, P. C. (2005). *The HTK Book*. Cambridge University Engineering Department.

10 SPEECH RECOGNITION: ADVANCED TOPICS

> True, their voice-print machine was unfortunately a crude one. It could discriminate among only a few frequencies, and it indicated amplitude by indecipherable blots. But it had never been intended for such vitally important work.
>
> Aleksandr I. Solzhenitsyn, *The First Circle*, p. 505

The *keju* civil service examinations of Imperial China lasted almost 1300 years, from the year 606 until it was abolished in 1905. In its peak, millions of would-be officials from all over China competed for high-ranking government positions by participating in a uniform examination. For the final 'metropolitan' part of this exam in the capital city, the candidates would be locked into an examination compound for a grueling 9 days and nights answering questions about history, poetry, the Confucian classics, and policy.

Naturally all these millions of candidates didn't all show up in the capital. Instead, the exam had progressive levels; candidates who passed a one-day local exam in their local prefecture could then sit for the biannual provincial exam, and only upon passing that exam in the provincial capital was a candidate eligible for the metropolitan and palace examinations.

This algorithm for selecting capable officials is an instance of multi-stage search. The final 9-day process requires far too many resources (in both space and time) to examine every candidate. Instead, the algorithm uses an easier, less intensive 1-day process to come up with a preliminary list of potential candidates, and applies the final test only to this list.

The *keju* algorithm can also be applied to speech recognition. We'd like to be able to apply very expensive algorithms in the speech recognition process, such as 4-gram, 5-gram, or even parser-based language models, or context-dependent phone models that can see two or three phones into the future or past. But there are a huge number of potential transcriptions sentences for any given waveform, and it's too expensive (in time, space, or both) to apply these powerful algorithms to every single candidate. Instead, we'll introduce **multipass decoding** algorithms in which efficient but dumber decoding algorithms produce shortlists of potential candidates to be rescored by slow but smarter algorithms. We'll also introduce the **context-dependent acoustic model**, which is one of these smarter knowledge sources that turns out to be essential in large-vocabulary speech recognition. We'll also briefly introduce the important topics of discriminative training and the modeling of variation.

10.1 Multipass Decoding: *N*-best lists and lattices

The previous chapter applied the Viterbi algorithm for HMM decoding. There are two main limitations of the Viterbi decoder, however. First, the Viterbi decoder does not actually compute the sequence of words which is most probable given the input acoustics. Instead, it computes an approximation to this: the sequence of *states* (i.e., *phones* or *subphones*) which is most probable given the input. More formally, recall that the true likelihood of an observation sequence O is computed by the forward algorithm by summing over all possible paths:

$$P(O|W) = \sum_{S\in S_1^T} P(O,S|W) \tag{10.1}$$

The Viterbi algorithm only approximates this sum by using the probability of the best path:

$$P(O|W) \approx \max_{S\in S_1^T} P(O,S|W) \tag{10.2}$$

VITERBI APPROXIMATION

It turns out that this **Viterbi approximation** is not too bad, since the most probable sequence of phones usually turns out to correspond to the most probable sequence of words. But not always. Consider a speech recognition system whose lexicon has multiple pronunciations for each word. Suppose the correct word sequence includes a word with very many pronunciations. Since the probabilities leaving the start arc of each word must sum to 1.0, each of these pronunciation-paths through this multiple-pronunciation HMM word model will have a smaller probability than the path through a word with only a single pronunciation path. Thus because the Viterbi decoder can only follow one of these pronunciation paths, it may ignore this many-pronunciation word in favor of an incorrect word with only one pronunciation path. In essence, the Viterbi approximation penalizes words with many pronunciations.

A second problem with the Viterbi decoder is that it is impossible or expensive for it to take advantage of many useful knowledge sources. For example the Viterbi algorithm as we have defined it cannot take complete advantage of any language model more complex than a bigram grammar. This is because of the fact mentioned earlier that a trigram grammar, for example, violates the **dynamic programming invariant**. Recall that this invariant is the simplifying (but incorrect) assumption that if the ultimate best path for the entire observation sequence happens to go through a state q_i, that this best path must include the best path up to and including state q_i. Since a trigram grammar allows the probability of a word to be based on the two previous words, it is possible that the best trigram-probability path for the sentence may go through a word but not include the best path to that word. Such a situation could occur if a particular word w_x has a high trigram probability given w_y, w_z, but that conversely the best path to w_y didn't include w_z (i.e., $P(w_y|w_q,w_z)$ was low for all q). Advanced probabilistic LMs like SCFGs also violate the same dynamic programming assumptions.

There are two solutions to these problems with Viterbi decoding. The most common is to modify the Viterbi decoder to return multiple potential utterances, instead of just the single best, and then use other high-level language model or pronunciation-modeling algorithms to re-rank these multiple outputs (Schwartz and Austin, 1991; Soong and Huang, 1990; Murveit et al., 1993).

STACK DECODER

A^*

The second solution is to employ a completely different decoding algorithm, such as the **stack decoder**, or **A*** decoder (Jelinek, 1969; Jelinek et al., 1975). We begin in this section with multiple-pass decoding, and return to stack decoding in the next section.

In **multiple-pass decoding** we break up the decoding process into two stages. In the first stage we use fast, efficient knowledge sources or algorithms to perform a non-optimal search. So for example we might use an unsophisticated but time-and-space efficient language model like a bigram, or use simplified acoustic models. In the second decoding pass we can apply more sophisticated but slower decoding algorithms on a reduced search space. The interface between these passes is an *N*-**best list** or **word lattice**.

N-BEST

RESCORED

The simplest algorithm for multipass decoding is to modify the Viterbi algorithm to return the **N-best** sentences (word sequences) for a given speech input. Suppose for example a bigram grammar is used with such an *N*-best-Viterbi algorithm to return the 1000 most highly-probable sentences, each with their AM likelihood and LM prior score. This 1000-best list can now be passed to a more sophisticated language model like a trigram grammar. This new LM is used to replace the bigram LM score of each hypothesized sentence with a new trigram LM probability. These priors can be combined with the acoustic likelihood of each sentence to generate a new posterior probability for each sentence. Sentences are thus **rescored** and re-ranked using this more sophisticated probability. Fig. 10.1 shows an intuition for this algorithm.

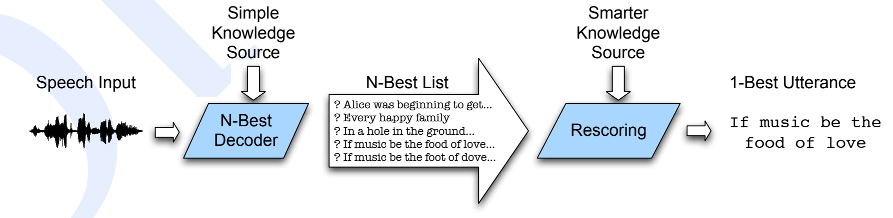

Simple Knowledge Source
Smarter Knowledge Source
Speech Input
N-Best Decoder
N-Best List
? Alice was beginning to get...
? Every happy family
? In a hole in the ground...
? If music be the food of love...
? If music be the foot of dove...
Rescoring
1-Best Utterance
If music be the food of love

Figure 10.1 The use of *N*-best decoding as part of a two-stage decoding model. Efficient but unsophisticated knowledge sources are used to return the *N*-best utterances. This significantly reduces the search space for the second pass models, which are thus free to be very sophisticated but slow.

There are a number of algorithms for augmenting the Viterbi algorithm to generate *N*-best hypotheses. It turns out that there is no polynomial-time admissible algorithm for finding the *N* most likely hypotheses (Young, 1984). There are however, a number of approximate (non-admissible) algorithms; we will introduce just one of them, the

"Exact N-best" algorithm of Schwartz and Chow (1990). In Exact N-best, instead of each state maintaining a single path/backtrace, we maintain up to N different paths for each state. But we'd like to insure that these paths correspond to different word paths; we don't want to waste our N paths on different state sequences that map to the same words. To do this, we keep for each path the **word history**, the entire sequence of words up to the current word/state. If two paths with the same word history come to a state at the same time, we merge the paths and sum the path probabilities. To keep the N best word sequences, the resulting algorithm requires $O(N)$ times the normal Viterbi time. We'll see this merging of paths again when we introducing decoding for statistical machine translation, where it is called **hypothesis recombination**.

HYPOTHESIS RECOMBINATION

		AM	LM
Rank	Path	logprob	logprob
1.	it's an area that's naturally sort of mysterious	-7193.53	-20.25
2.	that's an area that's naturally sort of mysterious	-7192.28	-21.11
3.	it's an area that's not really sort of mysterious	-7221.68	-18.91
4.	that scenario that's naturally sort of mysterious	-7189.19	-22.08
5.	there's an area that's naturally sort of mysterious	-7198.35	-21.34
6.	that's an area that's not really sort of mysterious	-7220.44	-19.77
7.	the scenario that's naturally sort of mysterious	-7205.42	-21.50
8.	so it's an area that's naturally sort of mysterious	-7195.92	-21.71
9.	that scenario that's not really sort of mysterious	-7217.34	-20.70
10.	there's an area that's not really sort of mysterious	-7226.51	-20.01

Figure 10.2 An example 10-Best list from the Broadcast News corpus, produced by the CU-HTK BN system (thanks to Phil Woodland). Logprobs use $\log_{10}$; the language model scale factor (LMSF) is 15.

The result of any of these algorithms is an N-best list like the one shown in Fig. 10.2. In Fig. 10.2 the correct hypothesis happens to be the first one, but of course the reason to use N-best lists is that isn't always the case. Each sentence in an N-best list is also annotated with an acoustic model probability and a language model probability. This allows a second-stage knowledge source to replace one of those two probabilities with an improved estimate.

One problem with an N-best list is that when N is large, listing all the sentences is extremely inefficient. Another problem is that N-best lists don't give quite as much information as we might want for a second-pass decoder. For example, we might want distinct acoustic model information for each word hypothesis so that we can reapply a new acoustic model for the word. Or we might want to have available different start and end times of each word so that we can apply a new duration model.

WORD LATTICE

For this reason, the output of a first-pass decoder is usually a more sophisticated representation called a **word lattice** (Murveit et al., 1993; Aubert and Ney, 1995). A word lattice is a directed graph that efficiently represents much more information about possible word sequences.[1] In some systems, nodes in the graph are words and arcs are

[1] Actually an ASR lattice is not the kind of lattice that may be familiar to you from mathematics, since it is not required to have the properties of a true lattice (i.e., be a partially ordered set with particular properties, such as a unique join for each pair of elements). Really it's just a graph, but it is conventional to call it a

transitions between words. In others, arcs represent word hypotheses and nodes are points in time. Let's use this latter model, and so each arc represents lots of information about the word hypothesis, including the start and end time, the acoustic model and language model probabilities, the sequence of phones (the pronunciation of the word), or even the phone durations. Fig. 10.3 shows a sample lattice corresponding to the *N*-best list in Fig. 10.2. Note that the lattice contains many distinct links (records) for the same word, each with a slightly different starting or ending time. Such lattices are not produced from *N*-best lists; instead, a lattice is produced during first-pass decoding by including some of the word hypotheses which were active (in the beam) at each time-step. Since the acoustic and language models are context-dependent, distinct links need to be created for each relevant context, resulting in a large number of links with the same word but different times and contexts. *N*-best lists like Fig. 10.2 can also be produced by first building a lattice like Fig. 10.3 and then tracing through the paths to produce *N* word strings.

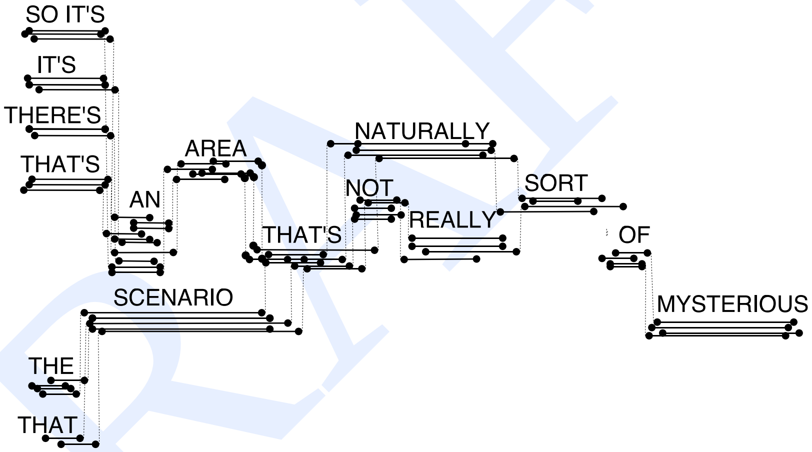


Figure 10.3 Word lattice corresponding to the *N*-best list in Fig. 10.2. The arcs beneath each word show the different start and end times for each word hypothesis in the lattice; for some of these we've shown schematically how each word hypothesis must start at the end of a previous hypothesis. Not shown in this figure are the acoustic and language model probabilities that decorate each arc.

The fact that each word hypothesis in a lattice is augmented separately with its acoustic model likelihood and language model probability allows us to rescore any path through the lattice, using either a more sophisticated language model or a more sophisticated acoustic model. As with *N*-best lists, the goal of this rescoring is to replace the **1-best utterance** with a different utterance that perhaps had a lower score on the first decoding pass. For this second-pass knowledge source to get perfect word error rate, the actual correct sentence would have to be in the lattice or *N*-best list. If the correct sentence isn't there, the rescoring knowledge source can't find it. Thus it

lattice.

LATTICE ERROR RATE

ORACLE

is important when working with a lattice or N-best list to consider the baseline **lattice error rate** (Woodland et al., 1995; Ortmanns et al., 1997): the lower bound word error rate from the lattice. The lattice error rate is the word error rate we get if we chose the lattice path (the sentence) that has the lowest word error rate. Because it relies on perfect knowledge of which path to pick, we call this an **oracle** error rate, since we need some oracle to tell us which sentence/path to pick.

LATTICE DENSITY

Another important lattice concept is the **lattice density**, which is the number of edges in a lattice divided by the number of words in the reference transcript. As we saw schematically in Fig. 10.3, real lattices are often extremely dense, with many copies of individual word hypotheses at slightly different start and end times. Because of this density, lattices are often pruned.

WORD GRAPH

Besides pruning, lattices are often simplified into a different, more schematic kind of lattice that is sometimes called a **word graph** or **finite state machine**, although often it's still just referred to as a word lattice. In these word graphs, the timing information is removed and multiple overlapping copies of the same word are merged. The timing of the words is left implicit in the structure of the graph. In addition, the acoustic model likelihood information is removed, leaving only the language model probabilities. The resulting graph is a weighted FSA, which is a natural extension of an N-gram language model; the word graph corresponding to Fig. 10.3 is shown in Fig. 10.4. This word graph can in fact be used as the language model for another decoding pass. Since such a wordgraph language model vastly restricts the search space, it can make it possible to use a complicated acoustic model which is too slow to use in first-pass decoding.

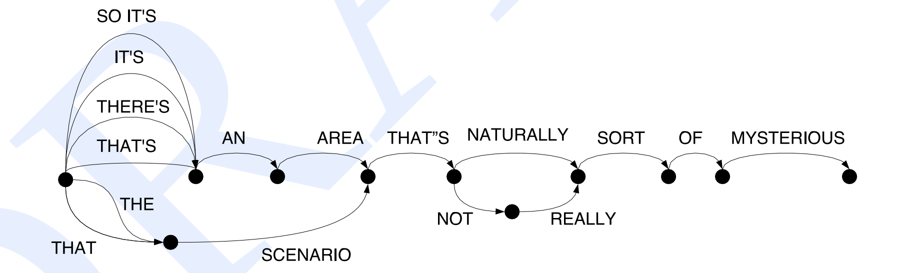


Figure 10.4 Word graph corresponding to the N-best list in Fig. 10.2. Each word hypothesis in the lattice also has language model probabilities (not shown in this figure).

A final type of lattice is used when we need to represent the posterior probability of individual words in a lattice. It turns out that in speech recognition, we almost never see the true posterior probability of anything, despite the fact that the goal of speech recognition is to compute the sentence with the maximum a posteriori probability. This is because in the fundamental equation of speech recognition we ignore the denominator in our maximization:

$$\hat{W} = \underset{W\in\mathscr{L}}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)} = \underset{W\in\mathscr{L}}{\operatorname{argmax}} P(O|W)P(W) \tag{10.3}$$

The product of the likelihood and the prior is **not** the posterior probability of the utterance. It is not even a probability, since it doesn't necessarily lie between 0 and 1. It's just a score. Why does it matter that we don't have a true probability? The reason is that without having true probability, we can choose the best hypothesis, but we can't know how good it is. Perhaps the best hypothesis is still really bad, and we need to ask the user to repeat themselves. If we had the posterior probability of a word it could be used as a **confidence** metric, since the posterior is an absolute rather than relative measure. A confidence metric is a metric that the speech recognizer can give to a higher-level process (like dialogue) to indicate how confident the recognizer is that the word string that it returns is a good one. We'll return to the use of confidence in Ch. 24.

CONFUSION NETWORKS
MESHES
SAUSAGES
PINCHED LATTICES

In order to compute the posterior probability of a word, we'll need to normalize over all the different word hypotheses available at a particular point in the utterances. At each point we'll need to know which words are competing or confusable. The lattices that show these sequences of word confusions are called **confusion networks**, **meshes**, **sausages**, or **pinched lattices**. A confusion network consists of a sequence of word positions. At each position is a set of mutually exclusive word hypotheses. The network represents the set of sentences that can be created by choosing one word from each position.

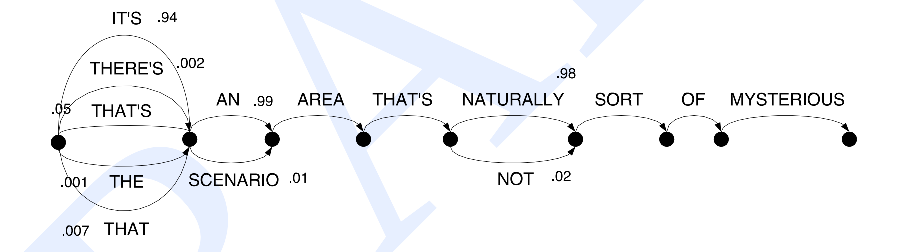


Figure 10.5 Confusion network corresponding to the word lattice in Fig. 10.3. Each word is associated with a posterior probability. Note that some of the words from the lattice have been pruned away. (Probabilities computed by the SRI-LM toolkit).

Note that unlike lattices or word graphs, the process of constructing a confusion network actually adds paths that were not in the original lattice. Confusion networks have other uses besides computing confidence. They were originally proposed for use in minimizing word error rate, by focusing on maximizing improving the word posterior probability rather than the sentence likelihood. Recently confusion networks have been used to train discriminative classifiers that distinguish between words.

Roughly speaking, confusion networks are built by taking the different hypothesis paths in the lattice and aligning them with each other. The posterior probability for each word is computing by first summing over all paths passing through a word, and then normalizing by the sum of the probabilities of all competing words. For further details see Mangu et al. (2000), Evermann and Woodland (2000), Kumar and Byrne (2002), Doumpiotis et al. (2003b).

Standard publicly available language modeling toolkits like SRI-LM (Stolcke, 2002) (`http://www.speech.sri.com/projects/srilm/`) and the HTK language modeling toolkit (Young et al., 2005) (`http://htk.eng.cam.ac.uk/`) can be used to generate and manipulate lattices, *N*-best lists, and confusion networks.

FORWARD-BACKWARD

There are many other kinds of multiple-stage search, such as the **forward-backward** search algorithm (not to be confused with the **forward-backward** algorithm for HMM parameter setting) (Austin et al., 1991) which performs a simple forward search followed by a detailed backward (i.e., time-reversed) search.

10.2 A* ('Stack') Decoding

Recall that the Viterbi algorithm approximated the forward computation, computing the likelihood of the single best (MAX) path through the HMM, while the forward algorithm computes the likelihood of the total (SUM) of all the paths through the HMM. The A* decoding algorithm allows us to use the complete forward probability, avoiding the Viterbi approximation. A* decoding also allows us to use any arbitrary language model. Thus A* is a one-pass alternative to multi-pass decoding.

The A* decoding algorithm is a best-first search of the tree that implicitly defines the sequence of allowable words in a language. Consider the tree in Fig. 10.6, rooted in the START node on the left. Each leaf of this tree defines one sentence of the language; the one formed by concatenating all the words along the path from START to the leaf. We don't represent this tree explicitly, but the stack decoding algorithm uses the tree implicitly as a way to structure the decoding search.

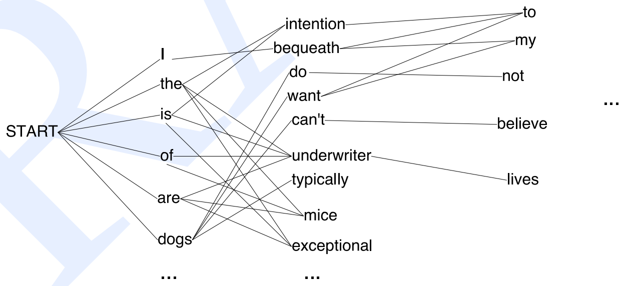


Figure 10.6 A visual representation of the implicit lattice of allowable word sequences that defines a language. The set of sentences of a language is far too large to represent explicitly, but the lattice gives a metaphor for exploring prefixes.

The algorithm performs a search from the root of the tree toward the leaves, looking for the highest probability path, and hence the highest probability sentence. As we proceed from root toward the leaves, each branch leaving a given word node represents a word which may follow the current word. Each of these branches has a probability, which expresses the conditional probability of this next word given the part of the sentence we've seen so far. In addition, we will use the forward algorithm to assign each word a likelihood of producing some part of the observed acoustic data. The A* decoder must thus find the path (word sequence) from the root to a leaf which

PRIORITY QUEUE

has the highest probability, where a path probability is defined as the product of its language model probability (prior) and its acoustic match to the data (likelihood). It does this by keeping a **priority queue** of partial paths (i.e., prefixes of sentences, each annotated with a score). In a priority queue each element has a score, and the *pop* operation returns the element with the highest score. The A* decoding algorithm iteratively chooses the best prefix-so-far, computes all the possible next words for that prefix, and adds these extended sentences to the queue. Fig. 10.7 shows the complete algorithm.

```
function STACK-DECODING() returns min-distance

    Initialize the priority queue with a null sentence.
    Pop the best (highest score) sentence $s$ off the queue.
    If ($s$ is marked end-of-sentence (EOS) ) output $s$ and terminate.
    Get list of candidate next words by doing fast matches.
    For each candidate next word $w$:
        Create a new candidate sentence $s+w$.
        Use forward algorithm to compute acoustic likelihood $L$ of $s+w$
        Compute language model probability $P$ of extended sentence $s+w$
        Compute “score”  for $s+w$ (a function of $L$, $P$, and ???)
        if (end-of-sentence) set EOS flag for $s+w$.
        Insert $s+w$ into the queue together with its score and EOS flag
```

Figure 10.7 The A* decoding algorithm (modified from Paul (1991) and Jelinek (1997)). The evaluation function that is used to compute the score for a sentence is not completely defined here; possible evaluation functions are discussed below.

FAST MATCH

Let’s consider a stylized example of an A* decoder working on a waveform for which the correct transcription is *If music be the food of love*. Fig. 10.8 shows the search space after the decoder has examined paths of length one from the root. A **fast match** is used to select the likely next words. A fast match is one of a class of heuristics designed to efficiently winnow down the number of possible following words, often by computing some approximation to the forward probability (see below for further discussion of fast matching).

At this point in our example, we’ve done the fast match, selected a subset of the possible next words, and assigned each of them a score. The word *Alice* has the highest score. We haven’t yet said exactly how the scoring works.

Fig. 10.9a show the next stage in the search. We have expanded the *Alice* node. This means that the *Alice* node is no longer on the queue, but its children are. Note that now the node labeled *if* actually has a higher score than any of the children of *Alice*. Fig. 10.9b shows the state of the search after expanding the *if* node, removing it, and adding *if music*, *if muscle*, and *if messy* on to the queue.

We clearly want the scoring criterion for a hypothesis to be related to its probability. Indeed it might seem that the score for a string of words w_1^i given an acoustic string y_1^j should be the product of the prior and the likelihood:

$$P(y_1^j|w_1^i)P(w_1^i)$$

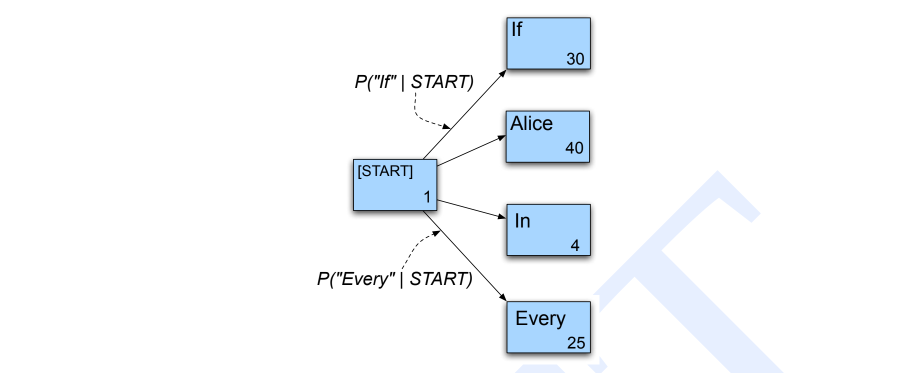
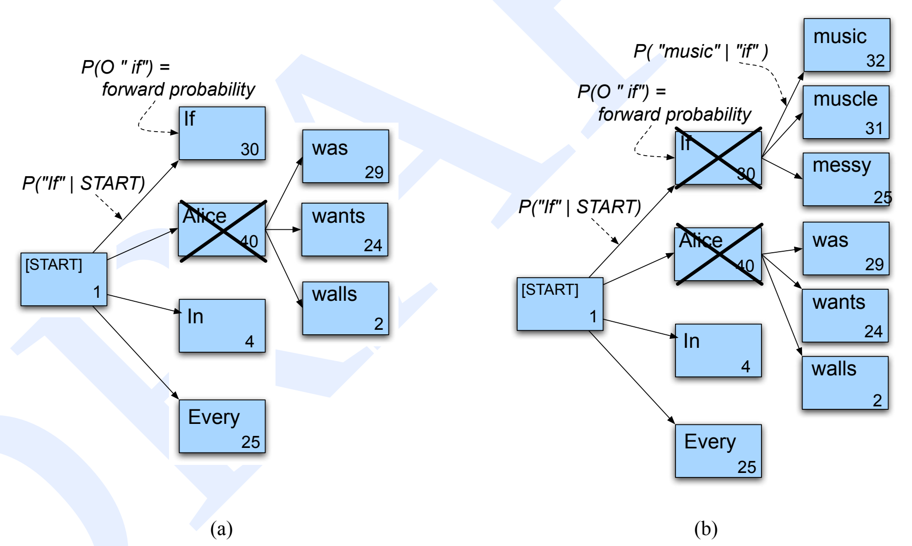


Figure 10.8 The beginning of the search for the sentence *If music be the food of love*. At this early stage *Alice* is the most likely hypothesis. (It has a higher score than the other hypotheses.)



(a) (b)

Figure 10.9 The next steps of the search for the sentence *If music be the food of love*. In (a) we've now expanded the *Alice* node and added three extensions which have a relatively high score; the highest-scoring node is *START if*, which is not along the *START Alice* path at all. In (b) we've expanded the *if* node. The hypothesis *START if music* then has the highest score.

Alas, the score cannot be this probability because the probability will be much smaller for a longer path than a shorter one. This is due to a simple fact about probabilities and substrings; any prefix of a string must have a higher probability than the string itself (e.g., P(START the ...) will be greater than P(START the book)). Thus if we used probability as the score, the A* decoding algorithm would get stuck on the single-word hypotheses.

Instead, we use the A* evaluation function (Nilsson, 1980; Pearl, 1984) $f^*(p)$, given a partial path p:

$$f^*(p) = g(p) + h^*(p)$$

$f^*(p)$ is the *estimated* score of the best complete path (complete sentence) which starts with the partial path p. In other words, it is an estimate of how well this path would do if we let it continue through the sentence. The A* algorithm builds this estimate from two components:

- $g(p)$ is the score from the beginning of utterance to the end of the partial path p. This g function can be nicely estimated by the probability of p given the acoustics so far (i.e., as $P(O|W)P(W)$ for the word string W constituting p).
- $h^*(p)$ is an estimate of the best scoring extension of the partial path to the end of the utterance.

Coming up with a good estimate of h^* is an unsolved and interesting problem. A very simple approach is to chose an h^* estimate which correlates with the number of words remaining in the sentence (Paul, 1991). Slightly smarter is to estimate the expected likelihood per frame for the remaining frames, and multiple this by the estimate of the remaining time. This expected likelihood can be computed by averaging the likelihood per frame in the training set. See Jelinek (1997) for further discussion.

Tree Structured Lexicons

We mentioned above that both the A* and various other two-stage decoding algorithms require the use of a **fast match** for quickly finding which words in the lexicon are likely candidates for matching some portion of the acoustic input. Many fast match algorithms are based on the use of a **tree-structured lexicon**, which stores the pronunciations of all the words in such a way that the computation of the forward probability can be shared for words which start with the same sequence of phones. The tree-structured lexicon was first suggested by Klovstad and Mondshein (1975); fast match algorithms which make use of it include Gupta et al. (1988), Bahl et al. (1992) in the context of A* decoding, and Ney et al. (1992) and Nguyen and Schwartz (1999) in the context of Viterbi decoding. Fig. 10.10 shows an example of a tree-structured lexicon from the Sphinx-II recognizer (Ravishankar, 1996). Each tree root represents the first phone of all words beginning with that context dependent phone (phone context may or may not be preserved across word boundaries), and each leaf is associated with a word.

TREE-STRUCTURED LEXICON

10.3 Context-Dependent Acoustic Models: Triphones

In our discussion in Sec. **??** of how the HMM architecture is applied to ASR, we showed how an HMM could be created for each phone, with its three emitting states corresponding to subphones at the beginning, middle, and end of the phone. We thus

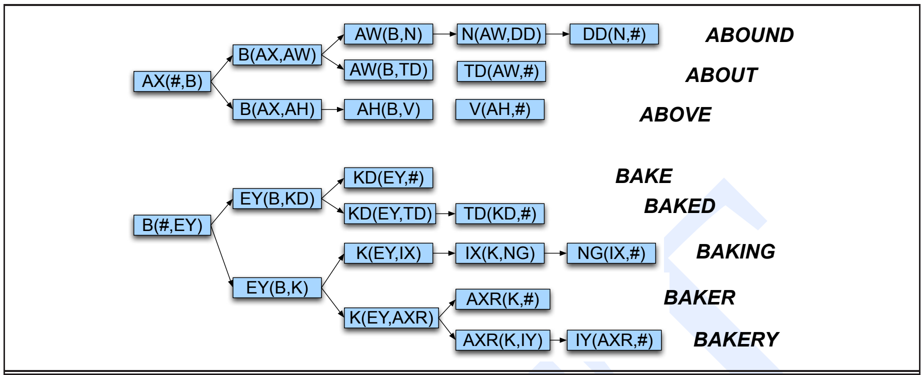


Figure 10.10 A tree-structured lexicon from the Sphinx-II recognizer (after Ravishankar (1996)). Each node corresponds to a particular triphone in the slightly modified version of the ARPAbet used by Sphinx-II. Thus EY(B,KD) means the phone EY preceded by a B and followed by the closure of a K.

represent each subphone ("beginning of [eh]", "beginning of [t]", "middle of [ae]") with its own GMM.

COARTICULATION

There is a problem with using a fixed GMM for a subphone like "beginning of [eh]". The problem is that phones vary enormously based on the phones on either side. This is because the movement of the articulators (tongue, lips, velum) during speech production is continuous and is subject to physical constraints like momentum. Thus an articulator may start moving during one phone to get into place in time for the next phone. In Ch. 7 we defined the word **coarticulation** as the movement of articulators to anticipate the next sound, or perseverating movement from the last sound. Fig. 10.11 shows coarticulation due to neighboring phone contexts for the vowel [eh].

CI PHONE
CD PHONES
TRIPHONE

In order to model the marked variation that a phone exhibits in different contexts, most LVCSR systems replace the idea of a context-independent (**CI phone**) HMM with a context-dependent or **CD phones**. The most common kind of context-dependent model is a **triphone** HMM (Schwartz et al., 1985; Deng et al., 1990). A triphone model represents a phone in a particular left and right context. For example the triphone *[y-eh+l]* means "[eh] preceded by [y] and followed by [l]". In general, [a-b+c] will mean "[b] preceded by [a] and followed by [c]". In situations where we don't have a full triphone context, we'll use [a-b] to mean "[b] preceded by [a]" and [b+c] to mean "[b] followed by [c]".

Context-dependent phones capture an important source of variation, and are a key part of modern ASR systems. But unbridled context-dependency also introduces the same problem we saw in language modeling: training data sparsity. The more complex the model we try to train, the less likely we are to have seen enough observations of each phone-type to train on. For a phoneset with 50 phones, in principle we would need 50^3 or 125,000 triphones. In practice not every sequence of three phones is possible (English doesn't seem to allow triphone sequences like [ae-eh+ow] or [m-j+t]). Young et al. (1994) found that 55,000 triphones are needed in the 20K Wall Street Journal task. But they found that only 18,500 of these triphones, i.e. less than half, actually

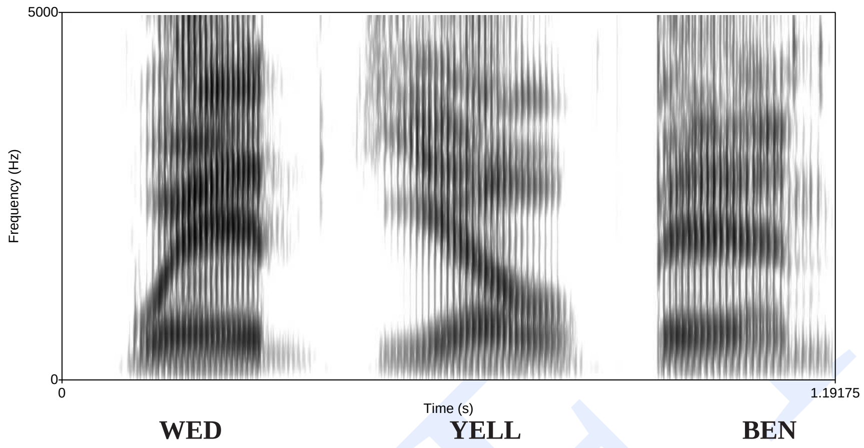


Figure 10.11 The vowel [eh] in three different triphone contexts, in the words *wed*, *yell*, and *Ben*. Notice the marked differences in the second formant (F2) at the beginning and end of the [eh] in all three cases.

occurred in the SI84 section of the WSJ training data.

Because of the problem of data sparsity, we must reduce the number of triphone parameters that we need to train. The most common way to do this is by clustering some of the contexts together and **tying** subphones whose contexts fall into the same cluster (Young and Woodland, 1994). For example, the beginning of a phone with an [n] on its left may look much like the beginning of a phone with an [m] on its left. We can therefore tie together the first (beginning) subphone of, say, the [m-eh+d] and [n-eh+d] triphones. Tying two states together means that they share the same Gaussians. So we only train a single Gaussian model for the first subphone of the [m-eh+d] and [n-eh+d] triphones. Likewise, it turns out that the left context phones [r] and [w] produce a similar effect on the initial subphone of following phones.

TYING

Fig. 10.12 shows, for example the vowel [iy] preceded by the consonants [w], [r], [m], and [n]. Notice that the beginning of [iy] has a similar rise in F2 after [w] and [r]. And notice the similarity of the beginning of [m] and [n]; as Ch. 7 noted, the position of nasal formants varies strongly across speakers, but this speaker (the first author) has a nasal formant (N2) around 1000 Hz.

Fig. 10.13 shows an example of the kind of triphone tying learned by the clustering algorithm. Each mixture Gaussian model is shared by the subphone states of various triphone HMMs.

How do we decide what contexts to cluster together? The most common method is to use a decision tree. For each state (subphone) of each phone, a separate tree is built. Fig. 10.14 shows a sample tree from the first (beginning) state of the phone /ih/, modified from Odell (1995). We begin at the root node of the tree with a single large cluster containing (the beginning state of) all triphones centered on /ih/. At each node in the tree, we split the current cluster into two smaller clusters by asking questions

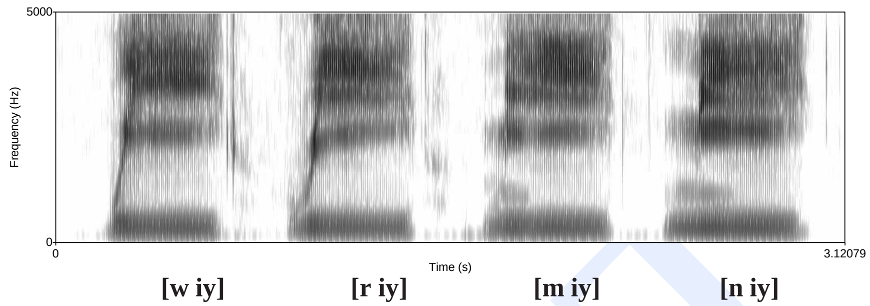


Figure 10.12 The words *we*, *re*, *me*, and *knee*. The glides [w] and [r] have similar effects on the beginning of the vowel [iy], as do the two nasals [n] and [m].

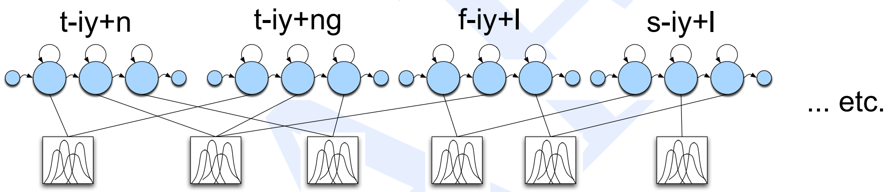


Figure 10.13 Four triphones showing the result of clustering. Notice that the initial subphone of [t-iy+n] and [t-iy+ng] is tied together, i.e. shares the same Gaussian mixture acoustic model. After Young et al. (1994).

about the context. For example the tree in Fig. 10.14 first splits the initial cluster into two clusters, one with nasal phone on the left, and one without. As we descend the tree from the root, each of these clusters is progressively split. The tree in Fig. 10.14 would split all beginning-state /ih/ triphones into 5 clusters, labeled A-E in the figure.

The questions used in the decision tree ask whether the phone to the left or right has a certain **phonetic feature**, of the type introduced in Ch. 7. Fig. 10.15 shows a few decision tree questions; note that there are separate questions for vowels and consonants. Real trees would have many more questions.

How are decision trees like the one in Fig. 10.14 trained? The trees are grown top down from the root. At each iteration, the algorithm considers each possible question q and each node n in the tree. For each such question, it considers how the new split would impact the acoustic likelihood of the training data. The algorithm computes the difference between the current acoustic likelihood of the training data, and the new likelihood if the models were tied based on splitting via question q. The algorithm picks the node n and question q which give the maximum likelihood. The procedure then iterates, stopping when each each leaf node has some minimum threshold number of examples.

We also need to modify the embedded training algorithm we saw in Sec. **??** to deal with context-dependent phones and also to handle mixture Gaussians. In both cases we

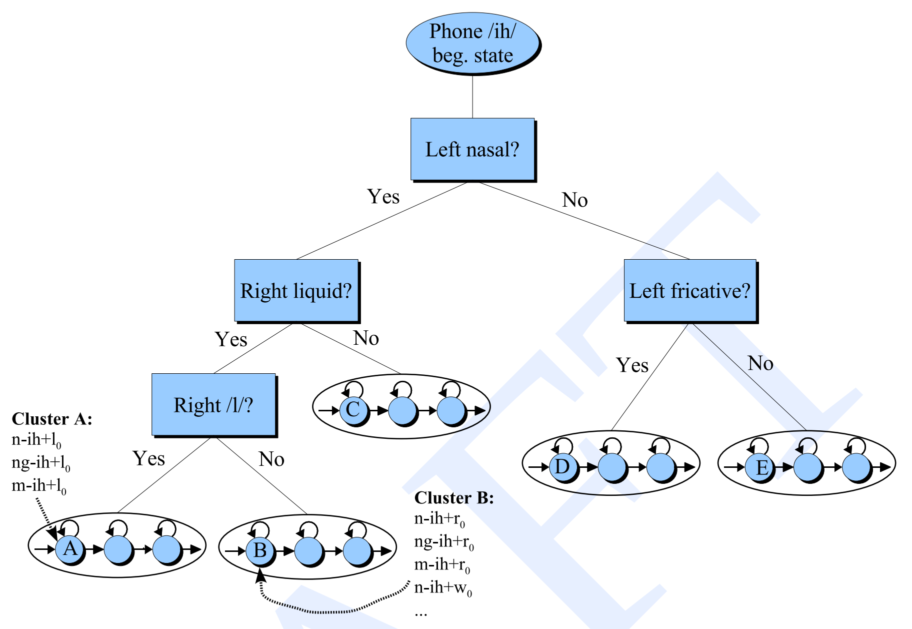


Figure 10.14 Decision tree for choosing which triphone states (subphones) to tie together. This particular tree will cluster state 0 (the beginning state) of the triphones /n-ih+l/, /ng-ih+l/, /m-ih+l/, into cluster class A, and various other triphones into classes B-E. Adapted from Odell (1995).

Feature	Phones
Stop	b d g k p t
Nasal	m n ng
Fricative	ch dh f jh s sh th v z zh
Liquid	l r w y
Vowel	aa ae ah ao aw ax axr ay eh er ey ih ix iy ow oy uh uw
Front Vowel	ae eh ih ix iy
Central Vowel	aa ah ao axr er
Back Vowel	ax ow uh uw
High Vowel	ih ix iy uh uw
Rounded	ao ow oy uh uw w
Reduced	ax axr ix
Unvoiced	ch f hh k p s sh t th
Coronal	ch d dh jh l n r s sh t th z zh

Figure 10.15 Sample decision tree questions on phonetic features. Modified from Odell (1995).

CLONING

use a more complex process that involves **cloning** and using extra iterations of EM, as described in Young et al. (1994).

To train context-dependent models, for example, we first use the standard embedded training procedure to train context-independent models, using multiple passes of EM and resulting in separate single-Gaussians models for each subphone of each monophone /aa/, /ae/, etc. We then **clone** each monophone model, i.e. make identical copies of the model with its 3 substates of Gaussians, one clone for each potential triphone. The *A* transition matrices are not cloned, but tied together for all the triphone clones of a monophone. We then run an iteration of EM again and retrain the triphone Gaussians. Now for each monophone we cluster all the context-dependent triphones using the clustering algorithm described on page 15 to get a set of tied state clusters. One typical state is chosen as the exemplar for this cluster and the rest are tied to it.

We use this same cloning procedure to learn Gaussian mixtures. We first use embedded training with multiple iterations of EM to learn single-mixture Gaussian models for each tied triphone state as described above. We then clone (split) each state into 2 identical Gaussians, perturb the values of each by some epsilon, and run EM again to retrain these values. We then split each of the two mixtures, resulting in four, perturb them, retrain. We continue until we have an appropriate number of mixtures for the amount of observations in each state.

A full context-depending GMM triphone model is thus created by applying these two cloning-and-retraining procedures in series, as shown schematically in Fig. 10.16.

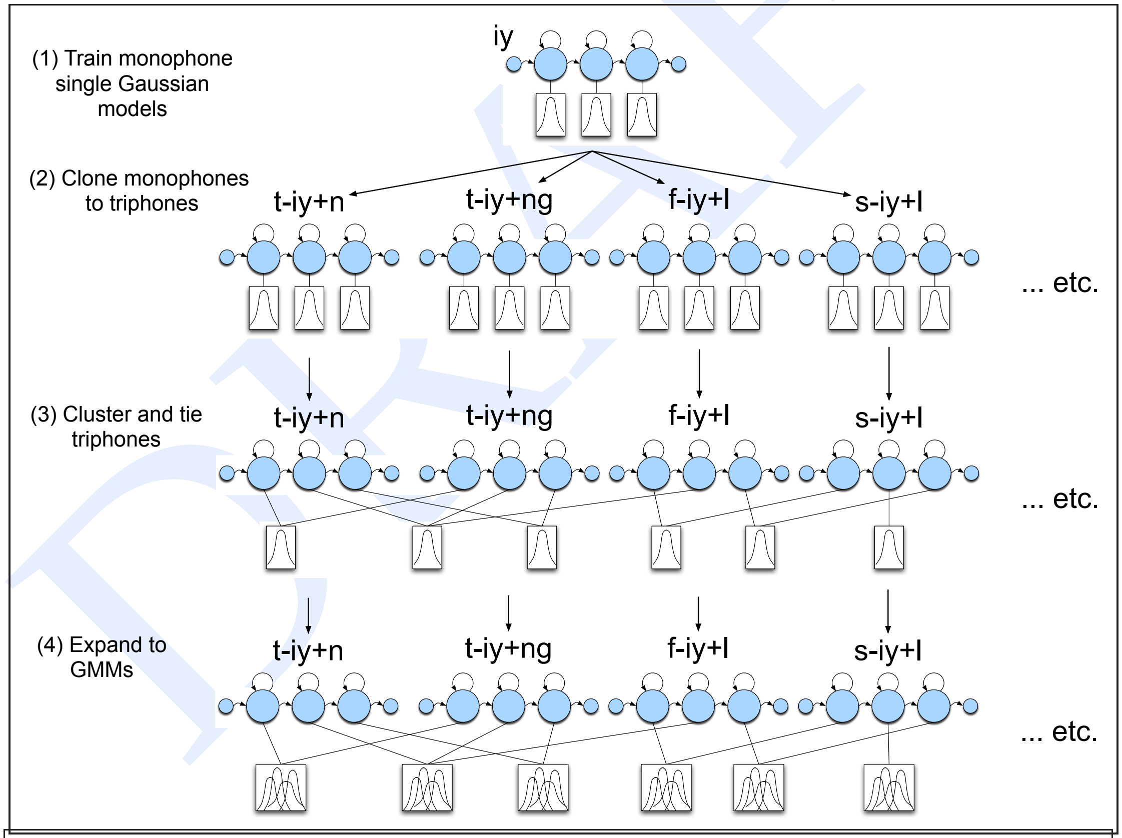


Figure 10.16 The four stages in training a tied-mixture triphone acoustic model. After Young et al. (1994).

10.4 Discriminative Training

MAXIMUM LIKELIHOOD ESTIMATION
MLE
DISCRIMINATING

The Baum-Welch and embedded training models we have presented for training the HMM parameters (the A and B matrices) are based on maximizing the likelihood of the training data. An alternative to this **maximum likelihood estimation** (**MLE**) is to focus not on fitting the best model to the data, but rather on **discriminating** the best model from all the other models. Such training procedures include Maximum Mutual Information Estimation (MMIE) (Woodland and Povey, 2002) the use of neural net/SVM classifiers (Bourlard and Morgan, 1994) as well as other techniques like Minimum Classification Error training (Chou et al., 1993; McDermott and Hazen, 2004) or Minimum Bayes Risk estimation (Doumpiotis et al., 2003a). We summarize the first two of these in the next two subsections.

10.4.1 Maximum Mutual Information Estimation

Recall that in Maximum Likelihood Estimation (MLE), we train our acoustic model parameters (A and B) so as to maximize the likelihood of the training data. Consider a particular observation sequence O, and a particular HMM model M_k corresponding to word sequence W_k, out of all the possible sentences $W' \in \mathscr{L}$. The MLE criterion thus maximizes

$$\mathscr{F}_{\text{MLE}}(\lambda) = P_\lambda(O|M_k) \tag{10.4}$$

Since our goal in speech recognition is to have the correct transcription for the largest number of sentences, we'd like on average for the probability of the **correct** word string W_k to be high; certainly higher than the probability of all the **wrong** word strings $W_j s.t. j \neq k$. But the MLE criterion above does not guarantee this. Thus we'd like to pick some other criterion which will let us chose the model λ which assigns the highest probability to the correct model, i.e. maximizes $P_\lambda(M_k|O)$. Maximizing the probability of the word string rather than the probability of the observation sequence is called **conditional maximum likelihood estimation** or CMLE:

$$\mathscr{F}_{\text{CMLE}}(\lambda) = P_\lambda(M_k|O) \tag{10.5}$$

Using Bayes Law, we can express this as

$$\mathscr{F}_{\text{CMLE}}(\lambda) = P_\lambda(M_k|O) = \frac{P_\lambda(O|M_k)P(M_k)}{P_\lambda(O)} \tag{10.6}$$

Let's now expand $P_\lambda(O)$ by marginalizing (summing over all sequences which could have produced it). The total probability of the observation sequence is the weighted sum over all word strings of the observation likelihood given that word string:

$$P(O) = \sum_{W \in \mathscr{L}} P(O|W)P(W) \tag{10.7}$$

So a complete expansion of Eq. 10.6 is:

$$\mathscr{F}_{\text{CMLE}}(\lambda) = P_\lambda(M_k|O) = \frac{P_\lambda(O|M_k)P(M_k)}{\sum_{M \in \mathscr{L}} P_\lambda(O|M)P(M)} \tag{10.8}$$

In a slightly confusing bit of standard nomenclature, CMLE is generally referred to instead as Maximum Mutual Information Estimation (MMIE). This is because it turns out that maximizing the posterior $P(W|O)$ and maximizing the mutual information $I(W,O)$ are equivalent if we assume that the language model probability of each sentence W is constant (fixed) during acoustic training, an assumption we usually make. Thus from here on we will refer to this criterion as the MMIE criterion rather than the CMLE criterion, and so here is Eq. 10.8 restated:

$$\mathscr{F}_{\text{MMIE}}(\lambda) = P_\lambda(M_k|O) = \frac{P_\lambda(O|M_k)P(M_k)}{\sum_{M\in\mathscr{L}} P_\lambda(O|M)P(M)} \tag{10.9}$$

In a nutshell, then, the goal of MMIE estimation is to maximize (10.9) rather than (10.4). Now if our goal is to maximize $P_\lambda(M_k|O)$, we not only need to maximize the numerator of (10.9), but also minimize the denominator. Notice that we can rewrite the denominator to make it clear that it includes a term equal to the model we are trying to maximize and a term for all other models:

$$P_\lambda(M_k|O) = \frac{P_\lambda(O|M_k)P(M_k)}{P_\lambda(O|M_k)P(M_k) + \sum_{i\neq k} P_\lambda(O|M_i)P(M_i)} \tag{10.10}$$

Thus in order to maximize $P_\lambda(M_k|O)$, we will need to incrementally change λ so that it increases the probability of the correct model, while simultaneously decreasing the probability of each of the incorrect models. Thus training with MMIE clearly fulfills the important goal of **discriminating** between the correct sequence and all other sequences.

The implementation of MMIE is quite complex, and we don't discuss it here except to mention that it relies on a variant of Baum-Welch training called Extended Baum-Welch that maximizes (10.9) instead of (10.4). Briefly, we can view this as a two step algorithm; we first use standard MLE Baum-Welch to compute the forward-backward counts for the training utterances. Then we compute another forward-backward pass using all other possible utterances and subtract these from the counts. Of course it turns out that computing this full denominator is computationally extremely expensive, because it requires running a full recognition pass on all the training data. Recall that in normal EM, we don't need to run decoding on the training data, since we are only trying to maximize the likelihood of the *correct* word sequence; in MMIE, we need to compute the probabilities of *all* possible word sequences. Decoding is very time-consuming because of complex language models. Thus in practice MMIE algorithms estimate the denominator by summing over only the paths that occur in a word lattice, as an approximation to the full set of possible paths.

CMLE was first proposed by Nadas (1983) and MMIE by Bahl et al. (1986), but practical implementations that actually reduced word error rate came much later; see Woodland and Povey (2002) or Normandin (1996) for details.

10.4.2 Acoustic Models based on Posterior Classifiers

Another way to think about discriminative training is to choose a classifier at the frame level which is discriminant. Thus while the Gaussian classifier is by far the most commonly used acoustic likelihood classifier, it is possible to instead use classifiers that are naturally discriminative or posterior estimators, such as neural networks or SVMs (support vector machines).

The posterior classifier (neural net or SVM) is generally integrated with an HMM architecture, is often called a **HMM-SVM** or **HMM-MLP hybrid** approach (Bourlard and Morgan, 1994).

The SVM or MLP approaches, like the Gaussian model, estimate the probability with respect to a cepstral feature vector at a single time t. Unlike the Gaussian model, the posterior approaches often uses a larger window of acoustic information, relying on cepstral feature vectors from neighboring time periods as well. Thus the input to a typical acoustic MLP or SVM might be feature vectors for the current frame plus the four previous and four following frames, i.e. a total of 9 cepstral feature vectors instead of the single one that the Gaussian model uses. Because they have such a wide context, SVM or MLP models generally use phones rather than subphones or triphones, and compute a posterior for each phone.

The SVM or MLP classifiers are thus computing the posterior probability of a state j given the observation vectors, i.e. $P(q_j|o_t)$. (also conditioned on the context, but let's ignore that for the moment). But the observation likelihood we need for the HMM, $b_j(o_t)$, is $P(o_t|q_j)$. The Bayes rule can help us see how to compute one from the other. The net is computing:

$$p(q_j|o_t) = \frac{P(o_t|q_j)p(q_j)}{p(o_t)} \tag{10.11}$$

We can rearrange the terms as follows:

$$\frac{p(o_t|q_j)}{p(o_t)} = \frac{P(q_j|o_t)}{p(q_j)} \tag{10.12}$$

The two terms on the right-hand side of (10.12) can be directly computed from the posterior classifier; the numerator is the output of the SVM or MLP, and the denominator is the total probability of a given state, summing over all observations (i.e., the sum over all t of $\xi_j(t)$). Thus although we cannot directly compute $P(o_t|q_j)$, we *can* use (10.12) to compute $\frac{p(o_t|q_j)}{p(o_t)}$, which is known as a **scaled likelihood** (the likelihood divided by the probability of the observation). In fact, the scaled likelihood is just as good as the regular likelihood, since the probability of the observation $p(o_t)$ is a constant during recognition and doesn't hurt us to have in the equation.

SCALED LIKELIHOOD

The supervised training algorithms for training a SVM or MLP posterior phone classifiers require that we know the correct phone label q_j for each observation o_t. We can use the same **embedded training** algorithm that we saw for Gaussians; we start with some initial version of our classifier and a word transcript for the training sentences. We run a forced alignment of the training data, producing a phone string, and now we retrain the classifier, and iterate.

10.5 Modeling Variation

As we noted at the beginning of this chapter, variation is one of the largest obstacles to successful speech recognition. We mentioned variation due to speaker differences from vocal characteristics or dialect, due to genre (such as spontaneous versus read speech), and due to the environment (such as noisy versus quiet environments). Handling this kind of variation is a major subject of modern research.

10.5.1 Environmental Variation and Noise

SPECTRAL SUBTRACTION

ADDITIVE NOISE

Environmental variation has received the most attention from the speech literature, and a number of techniques have been suggested for dealing with environmental noise. **Spectral subtraction**, for example, is used to combat **additive noise**. Additive noise is noise from external sound sources like engines or wind or fridges that is relatively constant and can be modeled as a noise signal that is just added in the time domain to the speech waveform to produce the observed signal. In spectral subtraction, we estimate the average noise during non-speech regions and then subtract this average value from the speech signal. Interestingly, speakers often compensate for high background noise levels by increasing their amplitude, F0, and formant frequencies. This change in speech production due to noise is called the **Lombard effect**, named for Etienne Lombard who first described it in 1911 (Junqua, 1993).

LOMBARD EFFECT

CEPSTRAL MEAN NORMALIZATION

CONVOLUTIONAL NOISE

Other noise robustness techniques like **cepstral mean normalization** are used to deal with **convolutional noise**, noise introduced by channel characteristics like different microphones. Here we compute the average of the cepstrum over time and subtract it from each frame; the average cepstrum models the fixed spectral characteristics of the microphone and the room acoustics (Atal, 1974).

Finally, some kinds of short non-verbal sounds like coughs, loud breathing, and throat clearing, or environmental sounds like beeps, telephone rings, and door slams, can be modeled explicitly. For each of these non-verbal sounds, we create a special phone and add to the lexicon a word consisting only of that phone. We can then use normal Baum-Welch training to train these phones just by modifying the training data transcripts to include labels for these new non-verbal 'words' (Ward, 1989). These words also need to be added to the language model; often by just allowing them to appear in between any word.

10.5.2 Speaker and Dialect Adaptation: Variation due to speaker differences

Speech recognition systems are generally designed to be speaker-independent, since it's rarely practical to collect sufficient training data to build a system for a single user. But in cases where we have enough data to build speaker-dependent systems, they function better than speaker-independent systems. This only makes sense; we can reduce the variability and increase the precision of our models if we are guaranteed that the test data will look more like the training data.

While it is rare to have enough data to train on an individual speaker, we do have enough data to train separate models for two important groups of speakers: men versus women. Since women and men have different vocal tracts and other acoustic and phonetic characteristics, we can split the training data by gender, and train separate acoustic models for men and for women. Then when a test sentence comes in, we use a gender detector to decide if it is male or female, and switch to those acoustic models. Gender detectors can be built out of binary GMM classifiers based on cepstral features. Such **gender-dependent acoustic modeling** is used in most LVCSR systems.

MLLR

SPEAKER ADAPTATION

Although we rarely have enough data to train on a specific speaker, there are techniques that work quite well at adapting the acoustic models to a new speaker very quickly. For example the **MLLR** (**Maximum Likelihood Linear Regression**) technique (Leggetter and Woodland, 1995) is used to adapt Gaussian acoustic models to a small amount of data from a new speaker. The idea is to use the small amount of data to train a linear transform to warp the means of the Gaussians. MLLR and other such techniques for **speaker adaptation** have been one of the largest sources of improvement in ASR performance in recent years.

The MLLR algorithm begins with a trained acoustic model and a small adaptation dataset from a new speaker. The adaptation set can be as small as 3 sentences or 10 seconds of speech. The idea is to learn a linear transform matrix (W) and a bias vector (ω) to transform the means of the acoustic model Gaussians. If the old mean of a Gaussian is μ, the equation for the new mean $\hat{\mu}$ is thus:

$$\hat{\mu} = W\mu + \omega \tag{10.13}$$

In the simplest case, we can learn a single global transform and apply it to each Gaussian models. The resulting equation for the acoustic likelihood is thus only very slightly modified:

$$b_j(o_t) = \frac{1}{\sqrt{2\pi|\Sigma j|}} \exp\left(-\frac{1}{2}(o_t - (W\mu_j + \omega))^T \Sigma_j^{-1}(o_t - (W\mu_j + \omega))\right) \tag{10.14}$$

The transform is learned by using linear regression to maximize the likelihood of the adaptation dataset. We first run forward-backward alignment on the adaptation set to compute the state occupation probabilities $\xi_j(t)$. We then compute W by solving a system of simultaneous equations involving $\xi_j(t)$. If enough data is available, it's also possible to learn a larger number of transforms.

MLLR is an example of the **linear transform** approach to speaker adaptation, one of the three major classes of speaker adaptation methods; the other two are **MAP adaptation** and **Speaker Clustering/Speaker Space** approaches. See Woodland (2001) for a comprehensive survey of speaker adaptation which covers all three families.

MLLR and other speaker adaptation algorithms can also be used to address another large source of error in LVCSR, the problem of foreign or dialect accented speakers. Word error rates go up when the test set speaker speaks a dialect or accent (such as Spanish-accented English or southern accented Mandarin Chinese) that differs from the (usually standard) training set, Here we can take an adaptation set of a few sentences from say 10 speakers, and adapt to them as a group, creating an MLLR transform that addresses whatever characteristics are present in the dialect or accent (Huang et al., 2000; Tomokiyo and Waibel, 2001; Wang et al., 2003; Zheng et al., 2005).

Another useful speaker adaptation technique is to control for the differing vocal tract lengths of speakers. Cues to the speaker's vocal tract length are present in the signal; for example speakers with longer vocal tracts tend to have lower formants. Vocal tract length can therefore be detected and normalized, in a process called **VTLN** (**Vocal Tract Length Normalization**); see the end notes for details.

VTLN

10.5.3 Pronunciation Modeling: Variation due to Genre

We said at the beginning of the chapter that recognizing conversational speech is harder for ASR systems than recognizing read speech. What are the causes of this difference? Is it the difference in vocabulary? Grammar? Something about the speaker themselves? Perhaps it's a fact about the microphones or telephone used in the experiment.

None of these seems to be the cause. In a well-known experiment, Weintraub et al. (1996) compared ASR performance on natural conversational speech versus performance on read speech, controlling for the influence of possible causal factors. Pairs of subjects in the lab had spontaneous conversations on the telephone. Weintraub et al. (1996) then hand-transcribed the conversations, and invited the participants back into the lab to read their own transcripts to each other over the same phone lines as if they were dictating. Both the natural and read conversations were recorded. Now Weintraub et al. (1996) had two speech corpora from identical transcripts; one original natural conversation, and one read speech. In both cases the speaker, the actual words, and the microphone were identical; the only difference was the naturalness or fluency of the speech. They found that read speech was much easier (WER=29%) than conversational speech (WER=53%). Since the speakers, words, and channel were controlled for, this difference must be modelable somewhere in the acoustic model or pronunciation lexicon.

Saraclar et al. (2000) tested the hypothesis that this difficulty with conversational speech was due to changed pronunciations, i.e., to a mismatch between the phone strings in the lexicon and what people actually said. Recall from Ch. 7 that conversational corpora like Switchboard contain many different pronunciations for words, (such as 12 different pronunciations for *because* and hundreds for *the*). Saraclar et al. (2000) showed in an oracle experiment that if a Switchboard recognizer is told which pronunciations to use for each word, the word error rate drops from 47% to 27%.

If knowing which pronunciation to use improves accuracy, could we improve recognition by simply adding more pronunciations for each word to the lexicon?

Alas, it turns out that adding multiple pronunciations doesn't work well, even if the list of pronunciation is represented as an efficient pronunciation HMM (Cohen, 1989). Adding extra pronunciations adds more confusability; if a common pronunciation of the word "of" is the single vowel [ax], it is now very confusable with the word "a". Another problem with multiple pronunciations is the use of Viterbi decoding. Recall our discussion on 2 that since the Viterbi decoder finds the best phone string, rather than the best word string, it biases against words with many pronunciations. Finally, using multiple pronunciations to model coarticulatory effects may be unnecessary because CD phones (triphones) are already quite good at modeling the contextual effects in phones due to neighboring phones, like the flapping and vowel-reduction handled by Fig. **??** (Jurafsky et al., 2001).

Instead, most current LVCSR systems use a very small number of pronunciations per word. What is commonly done is to start with a multiple pronunciation lexicon, where the pronunciations are found in dictionaries or are generated via phonological rules of the type described in Ch. 7. A forced Viterbi phone alignment is then run of the training set, using this dictionary. The result of the alignment is a phonetic transcription of the training corpus, showing which pronunciation was used, and the frequency of each pronunciation. We can then collapse similar pronunciations (for example if two pronunciations differ only in a single phone substitution we chose the more frequent pronunciation). We then chose the maximum likelihood pronunciation for each word. For frequent words which have multiple high-frequency pronunciations, some systems chose multiple pronunciations, and annotate the dictionary with the probability of these pronunciations; the probabilities are used in computing the acoustic likelihood (Cohen, 1989; Hain et al., 2001; Hain, 2002).

Finding a better method to deal with pronunciation variation remains an unsolved research problem. One promising avenue is to focus on non-phonetic factors that affect pronunciation. For example words which are highly predictable, or at the beginning or end of intonation phrases, or are followed by disfluencies, are pronounced very differently (Jurafsky et al., 1998; Fosler-Lussier and Morgan, 1999; Bell et al., 2003). Fosler-Lussier (1999) shows an improvement in word error rate by using these sorts of factors to predict which pronunciation to use. Another exciting line of research in pronunciation modeling uses a dynamic Bayesian network to model the complex overlap in articulators that produces phonetic reduction (Livescu and Glass, 2004b, 2004a).

Another important issue in pronunciation modeling is dealing with unseen words. In web-based applications such as telephone-based interfaces to the Web, the recognizer lexicon must be automatically augmented with pronunciations for the millions of unseen words, particularly names, that occur on the Web. Grapheme-to-phoneme techniques like those described in Sec. **??** are used to solve this problem.

10.6 Metadata: Boundaries, Punctuation, and Disfluencies

The output of the speech recognition process as we have described it so far is just a string of raw words. Consider the following sample gold-standard transcript (i.e., assuming perfect word recognition) of part of a dialogue (Jones et al., 2003):

> yeah actually um i belong to a gym down here a gold's gym uh-huh and uh exercise i try to exercise five days a week um and i usually do that uh what type of exercising do you do in the gym

Compare the difficult transcript above with the following much clearer version:

> A: Yeah I belong to a gym down here. Gold's Gym. And I try to exercise five days a week. And I usually do that.
> B: What type of exercising do you do in the gym?

The raw transcript is not divided up among speakers, there is no punctuation or capitalization, and disfluencies are scattered among the words. A number of studies have shown that such raw transcripts are harder for people to read Jones et al. (2003, 2005) and that adding, for example, commas back into the transcript improve sthe accuracy of information extraction algorithms on the transcribed text (Makhoul et al., 2005; Hillard et al., 2006). Post-processing ASR output involves tasks including the following:

DIARIZATION

diarization: Many speech tasks have multiple speakers, such as telephone conversations, business meetings, and news reports (with multiple broadcasters). Diarization is the task of breaking up a speech file by speaker assigning parts of the transcript to the relevant speakers, like the **A:** and **B:** labels above.

SENTENCE SEGMENTATION

sentence boundary detection: We discussed the task of breaking speech into sentences (sentence segmentation) in Ch. 3 and Ch. 8. But for those tasks we already add punctuation like periods to help us; from speech we don't already have punctuation, just words. Sentence segmentation from speech has the added difficulty that the transcribed words will be errorful, but has the advantage that prosodic features like pauses and sentence-final intontation can be used as cues.

TRUECASING

truecasing: Words in a clean transcript need to have sentence-initial words starting with an upper-case letter, acronyms all in capitals, and so on. Truecasing is the task of assigning the correct case for a word, and is often addressed as a HMM classification task like part-of-speech tagging, with hidden states like ALL-LOWER CASE, UPPER-CASE-INITIAL, *all-caps*, and so on.

PUNCTUATION DETECTION

punctuation detection: In addition to segmenting sentences, we need to choose sentence-final punctuation (period, question mark, exclamation mark), and insert commas and quotation marks and so on.

DISFLUENCY DETECTION

disfluency detection: Disfluencies can be removed from a transcript for readability, or at least marked off with commas or font changes. Since standard recognizers don't actually include disfluencies (like word fragments) in their transcripts, disfluency detection algorithms can also play an important role in avoiding the misrecognized words that may result.

METADATA

RICH TRANSCRIPTION

Marking these features (punctuation, boundaries, diarization) in the text output is often called **metadata** or sometimes **rich transcription**. Let's look at a couple of these tasks in slightly more detail.

Sentence segmentation can be modeled as a binary classification task, in which each boundary between two words is judged as a sentence boundary or as sentence-internal. Such classifiers can use similar features to the sentence segmentation discussed in Sec. **??**, such as words and part-of-speech tags around each candidate boundary, or length features such as the distance from the previously found boundary. We can also make use of prosodic features, especially pause duration, word duration (recall that sentence-final words are lengthened), and pitch movements.

Fig. 10.17 shows the candidate boundary locations in a sample sentence. Commonly extracted features include:

pause features: duration of the interword pause at the candidate boundary.

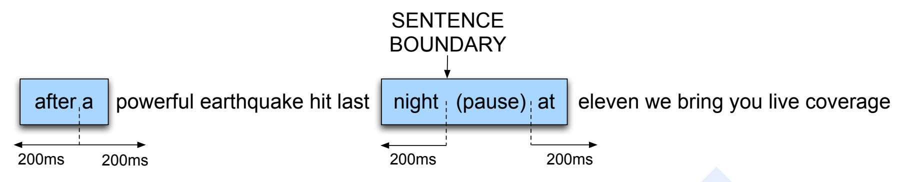

SENTENCE
BOUNDARY
after a
powerful earthquake hit last
night
(pause)
at
eleven we bring you live coverage
200ms
200ms
200ms
200ms

Figure 10.17 Candidate sentence boundaries computed at each inter-word boundary, showing prosodic feature extraction regions. After Shriberg et al. (2000).

duration features: durations of the phone and rime (nucleus plus coda) preceding the candidate boundary. Since some phones are inherently longer than others, each phone is normalized to the mean duration for that phone.

F0 features: the **change in pitch** across the boundary; sentence boundaries often have **pitch reset** (an abrupt change in pitch), while non-boundaries are more likely to have continuous pitch across the boundary. Another useful F0 feature is the **pitch range** of the preboundary word; sentences often end with a **final fall** (Sec. **??**) which is close to the speaker's F0 baseline.

For **punctuation detection**, similar features are used as for sentence boundary detection, but with multiple hidden classes (comma, sentence-final question mark, quotation mark, no punctuation). instead of just two.

For both of these tasks, instead of a simple binary classifier, sequence information can be incorporated by modeling sentence segmentation as an HMM in which the hidden states correspond to sentence boundary or non-boundary decisions. We will describe methods for combining prosodic and lexical features in more detail when we introduce dialogue act detection in Sec. **??**.

DISFLUENCIES
REPAIR

Recall from Sec. **??** that **disfluencies** or **repair** in conversation include phenomena like the following:

Disfluency type	Example
fillers (or **filled pauses**):	But, *uh*, that was absurd
word fragments	A guy went to a *d-*, a landfill
repetitions:	it was just a *change of*, *change of* location
restarts	it's – I find it very strange

The ATIS sentence in Fig. 10.18 shows examples of a restart and the filler *uh*, showing the

Detection methods for disfluencies are very similar to detecting sentence boundaries; a classifier is trained to make a decision at each word boundary, using both text and prosodic features. HMM and CRF classifiers are commonly used, and features are quite similar to the features for boundary detection, including neighoring words and part-of-speech tags, the duration of pauses at the word boundary, the duration of the word and phones preceding the boundary, the difference in pitch values across the boundary, and so on.

JITTER

For detecting fragments, features for detecting voice quality are used (Liu, 2004), such as **jitter**, a measure of perturbation in the pitch period (Rosenberg, 1971), **spectral**

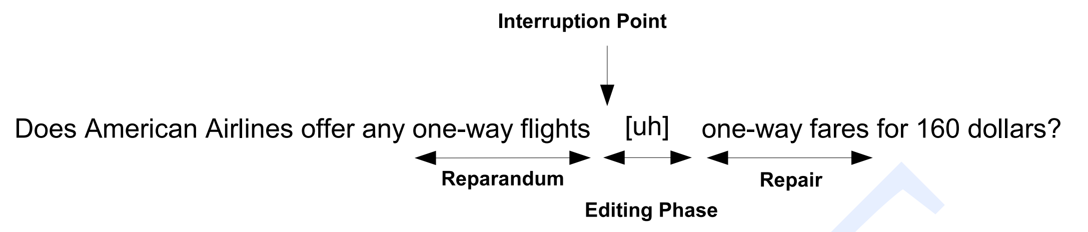

Interruption Point
Does American Airlines offer any one-way flights
[uh]
one-way fares for 160 dollars?
Reparandum
Repair
Editing Phase

Figure 10.18 Repeated from Fig. **??**An example of a disfluency (after Shriberg (1994); terminology is from Levelt (1983)).

SPECTRAL TILT
OPEN QUOTIENT

tilt, the slope of the spectrum, (see Sec. **??**), and **open quotient**, the percentage of the glottal cycle in which the vocal folds are open (Fant, 1997).

10.7 Speech Recognition by Humans

Humans are of course much better at speech recognition than machines; current machines are roughly about five times worse than humans on clean speech, and the gap seems to increase with noisy speech.

LEXICAL ACCESS

Speech recognition in humans shares some features with ASR algorithms. We mentioned above that signal processing algorithms like PLP analysis (Hermansky, 1990) were in fact inspired by properties of the human auditory system. In addition, three properties of human **lexical access** (the process of retrieving a word from the mental lexicon) are also true of ASR models: **frequency**, **parallelism**, and **cue-based processing**. For example, as in ASR with its *N*-gram language models, human lexical access is sensitive to word **frequency**. High-frequency spoken words are accessed faster or with less information than low-frequency words. They are successfully recognized in noisier environments than low frequency words, or when only parts of the words are presented (Howes, 1957; Grosjean, 1980; Tyler, 1984, inter alia). Like ASR models, human lexical access is **parallel**: multiple words are active at the same time (Marslen-Wilson and Welsh, 1978; Salasoo and Pisoni, 1985, inter alia).

PHONEME RESTORATION EFFECT
MCGURK EFFECT

Finally, human speech perception is **cue based**: speech input is interpreted by integrating cues at many different levels. Human phone perception combines acoustic cues, such as formant structure or the exact timing of voicing, (Oden and Massaro, 1978; Miller, 1994) visual cues, such as lip movement (McGurk and Macdonald, 1976; Massaro and Cohen, 1983; Massaro, 1998) and lexical cues such as the identity of the word in which the phone is placed (Warren, 1970; Samuel, 1981; Connine and Clifton, 1987; Connine, 1990). For example, in what is often called the **phoneme restoration effect**, Warren (1970) took a speech sample and replaced one phone (e.g. the [s] in *legislature*) with a cough. Warren found that subjects listening to the resulting tape typically heard the entire word *legislature* including the [s], and perceived the cough as background. In the **McGurk effect**, (McGurk and Macdonald, 1976) showed that visual input can interfere with phone perception, causing us to perceive a completely different phone. They showed subjects a video of someone say-

WORD ASSOCIATION

REPETITION PRIMING

ing the syllable *ga* in which the audio signal was dubbed instead with someone saying the syllable *ba*. Subjects reported hearing something like *da* instead. It is definitely worth trying this out yourself from video demos on the web; see for example `http://www.haskins.yale.edu/featured/heads/mcgurk.html`. Other cues in human speech perception include semantic **word association** (words are accessed more quickly if a semantically related word has been heard recently) and **repetition priming** (words are accessed more quickly if they themselves have just been heard). The intuitions of both these results are incorporated into recent language models discussed in Ch. 4, such as the cache model of Kuhn and De Mori (1990), which models repetition priming, or the trigger model of Rosenfeld (1996) and the LSA models of Coccaro and Jurafsky (1998) and Bellegarda (1999), which model word association. In a fascinating reminder that good ideas are never discovered only once, Cole and Rudnicky (1983) point out that many of these insights about context effects on word and phone processing were actually discovered by William Bagley (1901). Bagley achieved his results, including an early version of the phoneme restoration effect, by recording speech on Edison phonograph cylinders, modifying it, and presenting it to subjects. Bagley's results were forgotten and only rediscovered much later.[2]

ON-LINE

One difference between current ASR models and human speech recognition is the time-course of the model. It is important for the performance of the ASR algorithm that the the decoding search optimizes over the entire utterance. This means that the best sentence hypothesis returned by a decoder at the end of the sentence may be very different than the current-best hypothesis, halfway into the sentence. By contrast, there is extensive evidence that human processing is **on-line**: people incrementally segment and utterance into words and assign it an interpretation as they hear it. For example, Marslen-Wilson (1973) studied **close shadowers**: people who are able to shadow (repeat back) a passage as they hear it with lags as short as 250 ms. Marslen-Wilson found that when these shadowers made errors, they were syntactically and semantically appropriate with the context, indicating that word segmentation, parsing, and interpretation took place within these 250 ms. Cole (1973) and Cole and Jakimik (1980) found similar effects in their work on the detection of mispronunciations. These results have led psychological models of human speech perception (such as the Cohort model (Marslen-Wilson and Welsh, 1978) and the computational TRACE model (McClelland and Elman, 1986)) to focus on the time-course of word selection and segmentation. The TRACE model, for example, is a connectionist interactive-activation model, based on independent computational units organized into three levels: feature, phoneme, and word. Each unit represents a hypothesis about its presence in the input. Units are activated in parallel by the input, and activation flows between units; connections between units on different levels are excitatory, while connections between units on single level are inhibitatory. Thus the activation of a word slightly inhibits all other words.

We have focused on the similarities between human and machine speech recognition; there are also many differences. In particular, many other cues have been shown to play a role in human speech recognition but have yet to be successfully integrated into ASR. The most important class of these missing cues is prosody. To give only one example, Cutler and Norris (1988), Cutler and Carter (1987) note that most mul-

[2] Recall the discussion on page **??** of multiple independent discovery in science.

tisyllabic English word tokens have stress on the initial syllable, suggesting in their metrical segmentation strategy (MSS) that stress should be used as a cue for word segmentation. Another difference is that human lexical access exhibits **neighborhood effects** (the neighborhood of a word is the set of words which closely resemble it). Words with large frequency-weighted neighborhoods are accessed slower than words with less neighbors (Luce et al., 1990). Current models of ASR don't generally focus on this word-level competition.

10.8 Summary

- We introduced two advanced decoding algorithms: The multipass (*N*-best or lattice) decoding algorithm, and **stack** or **A*** decoding.
- Advanced acoustic models are based on context-dependent **triphones** rather than phones. Because the complete set of triphones would be too large, we use a smaller number of automatically clustered triphones instead.
- Acoustic models can be **adapted** to new speakers.
- Pronunciation variation is a source of errors in human-human speech recognition, but one that is not successfully handled by current technology.

Bibliographical and Historical Notes

A* SEARCH

See the previous chapter for most of the relevant speech recognition history. Note that although stack decoding is equivalent to the **A*** **search** developed in artificial intelligence, the stack decoding algorithm was developed independently in the information theory literature and the link with AI best-first search was noticed only later (Jelinek, 1976). Useful references on vocal tract length normalization include (Cohen et al., 1995; Wegmann et al., 1996; Eide and Gish, 1996; Lee and Rose, 1996; Welling et al., 2002; Kim et al., 2004).

FRAME-BASED

SEGMENT-BASED RECOGNIZERS

There are many new directions in current speech recognition research involving alternatives to the HMM model. For example,there are new architectures based on graphical models (dynamic bayes nets, factorial HMMs, etc) (Zweig, 1998; Bilmes, 2003; Livescu et al., 2003; Bilmes and Bartels, 2005; Frankel et al., 2007). There are attempts to replace the **frame-based** HMM acoustic model (that make a decision about each frame) with **segment-based recognizers** that attempt to detect variable-length segments (phones) (Digilakis, 1992; Ostendorf et al., 1996; Glass, 2003). Landmark-based recognizers and articulatory phonology-based recognizers focus on the use of distinctive features, defined acoustically or articulatorily (respectively) (Niyogi et al., 1998; Livescu, 2005; Hasegawa-Johnson and et al, 2005; Juneja and Espy-Wilson, 2003).

See Shriberg (2005) for an overview of metadata research. Shriberg (2002) and Nakatani and Hirschberg (1994) are computationally-focused corpus studies of the acoustic and lexical properties of disfluencies. Early papers on sentence segmentation from speech include Wang and Hirschberg (1992), Ostendorf and Ross (1997) See Shriberg et al. (2000), Liu et al. (2006a) for recent work on sentence segmentation, Kim and Woodland (2001), Hillard et al. (2006) on punctuation detection, Nakatani and Hirschberg (1994), Honal and Schultz (2003, 2005), Lease et al. (2006), and a number of papers that jointly address multiple metadata extraction tasks (Heeman and Allen, 1999; Liu et al., 2005, 2006b).

EXERCISES

10.1 Implement the Stack decoding algorithm of Fig. 10.7 on page 9. Pick a very simple h^* function like an estimate of the number of words remaining in the sentence.

10.2 Modify the forward algorithm of Fig. **??** from Ch. 9 to use the tree-structured lexicon of Fig. 10.10 on page 12.

10.3 Many ASR systems, including the Sonic and HTK systems, use a different algorithm for Viterbi called the **token-passing Viterbi** algorithm (Young et al., 1989). Read this paper and implement this algorithm.

Atal, B. S. (1974). Effectiveness of linear prediction characteristics of the speech wave for automatic speaker identification and verification. *The Journal of the Acoustical Society of America*, *55*(6), 1304–1312.

Aubert, X. and Ney, H. (1995). Large vocabulary continuous speech recognition using word graphs. In *IEEE ICASSP*, Vol. 1, pp. 49–52.

Austin, S., Schwartz, R., and Placeway, P. (1991). The forward-backward search algorithm. In *IEEE ICASSP-91*, Vol. 1, pp. 697–700.

Bagley, W. C. (1900–1901). The apperception of the spoken sentence: A study in the psychology of language. *The American Journal of Psychology*, *12*, 80–130. †.

Bahl, L. R., Brown, P. F., de Souza, P. V., and Mercer, R. L. (1986). Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *IEEE ICASSP-86*, Tokyo, pp. 49–52.

Bahl, L. R., de Souza, P. V., Gopalakrishnan, P. S., Nahamoo, D., and Picheny, M. A. (1992). A fast match for continuous speech recognition using allophonic models. In *IEEE ICASSP-92*, San Francisco, CA, pp. I.17–20.

Bell, A., Jurafsky, D., Fosler-Lussier, E., Girand, C., Gregory, M. L., and Gildea, D. (2003). Effects of disfluencies, predictability, and utterance position on word form variation in English conversation. *Journal of the Acoustical Society of America*, *113*(2), 1001–1024.

Bellegarda, J. R. (1999). Speech recognition experiments using multi-span statistical language models. In *IEEE ICASSP-99*, pp. 717–720.

Bilmes, J. (2003). Buried Markov Models: A graphical-modeling approach to automatic speech recognition. *Computer Speech and Language*, *17*(2-3).

Bilmes, J. and Bartels, C. (2005). Graphical model architectures for speech recognition. *IEEE Signal Processing Magazine*, *22*(5), 89–100.

Bourlard, H. and Morgan, N. (1994). *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Press.

Chou, W., Lee, C. H., and Juang, B. H. (1993). Minimum error rate training based on *n*-best string models. In *IEEE ICASSP-93*, pp. 2.652–655.

Coccaro, N. and Jurafsky, D. (1998). Towards better integration of semantic predictors in statistical language modeling. In *ICSLP-98*, Sydney, Vol. 6, pp. 2403–2406.

Cohen, J., Kamm, T., and Andreou, A. (1995). Vocal tract normalization in speech recognition: compensating for systematic systematic speaker variability. *Journal of the Acoustical Society of America*, *97*(5), 3246–3247.

Cohen, M. H. (1989). *Phonological Structures for Speech Recognition*. Ph.D. thesis, University of California, Berkeley.

Cole, R. A. (1973). Listening for mispronunciations: A measure of what we hear during speech. *Perception and Psychophysics*, *13*, 153–156.

Cole, R. A. and Jakimik, J. (1980). A model of speech perception. In Cole, R. A. (Ed.), *Perception and Production of Fluent Speech*, pp. 133–163. Lawrence Erlbaum.

Cole, R. A. and Rudnicky, A. I. (1983). What's new in speech perception? The research and ideas of William Chandler Bagley. *Psychological Review*, *90*(1), 94–101.

Connine, C. M. (1990). Effects of sentence context and lexical knowledge in speech processing. In Altmann, G. T. M. (Ed.), *Cognitive Models of Speech Processing*, pp. 281–294. MIT Press.

Connine, C. M. and Clifton, C. (1987). Interactive use of lexical information in speech perception. *Journal of Experimental Psychology: Human Perception and Performance*, *13*, 291–299.

Cutler, A. and Carter, D. M. (1987). The predominance of strong initial syllables in the English vocabulary. *Computer Speech and Language*, *2*, 133–142.

Cutler, A. and Norris, D. (1988). The role of strong syllables in segmentation for lexical access. *Journal of Experimental Psychology: Human Perception and Performance*, *14*, 113–121.

Deng, L., Lennig, M., Seitz, F., and Mermelstein, P. (1990). Large vocabulary word recognition using context-dependent allophonic hidden Markov models. *Computer Speech and Language*, *4*, 345–357.

Digilakis, V. (1992). *Segment-based stochastic models of spectral dynamics for continuous speech recognition*. Ph.D. thesis, Boston University.

Doumpiotis, V., Tsakalidis, S., and Byrne, W. (2003a). Discriminative training for segmental minimum bayes-risk decoding. In *IEEE ICASSP-03*.

Doumpiotis, V., Tsakalidis, S., and Byrne, W. (2003b). Lattice segmentation and minimum bayes risk discriminative training. In *EUROSPEECH-03*.

Eide, E. M. and Gish, H. (1996). A parametric approach to vocal tract length normalization. In *IEEE ICASSP-96*, Atlanta, GA, pp. 346–348.

Evermann, G. and Woodland, P. C. (2000). Large vocabulary decoding and confidence estimation using word posterior probabilities. In *IEEE ICASSP-00*, Istanbul, Vol. III, pp. 1655–1658.

Fant, G. (1997). The voice source in connected speech. *Speech Communication*, *22*(2-3), 125–139.

Fosler-Lussier, E. (1999). Multi-level decision trees for static and dynamic pronunciation models. In *EUROSPEECH-99*, Budapest.

Fosler-Lussier, E. and Morgan, N. (1999). Effects of speaking rate and word predictability on conversational pronunciations. *Speech Communication*, *29*(2-4), 137–158.

Frankel, J., Wester, M., and King, S. (2007). Articulatory feature recognition using dynamic bayesian networks. *Computer Speech and Language*, *21*(4), 620–640.

Glass, J. (2003). A probabilistic framework for segment-based speech recognition. *Computer Speech and Language,*, *17*(1–2), 137–152.

Grosjean, F. (1980). Spoken word recognition processes and the gating paradigm. *Perception and Psychophysics*, *28*, 267–283.

Gupta, V., Lennig, M., and Mermelstein, P. (1988). Fast search strategy in a large vocabulary word recognizer. *Journal of the Acoustical Society of America*, *84*(6), 2007–2017.

Hain, T. (2002). Implicit pronunciation modelling in asr. In *Proceedings of ISCA Pronunciation Modeling Workshop*.

Hain, T., Woodland, P. C., Evermann, G., and Povey, D. (2001). New features in the CU-HTK system for transcription of conversational telephone speech. In *IEEE ICASSP-01*, Salt Lake City, Utah.

Hasegawa-Johnson, M. and et al (2005). Landmark-based speech recognition: Report of the 2004 Johns Hopkins Summer Workshop. In *IEEE ICASSP-05*.

Heeman, P. A. and Allen, J. (1999). Speech repairs, intonational phrases and discourse markers: Modeling speakers' utterances in spoken dialog. *Computational Linguistics*, *25*(4).

Hermansky, H. (1990). Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America*, *87*(4), 1738–1752.

Hillard, D., Huang, Z., Ji, H., Grishman, R., Hakkani-Tür, D., Harper, M., Ostendorf, M., and Wang, W. (2006). Impact of automatic comma prediction on pos/name tagging of speech. In *Proceedings of IEEE/ACL 06 Workshop on Spoken Language Technology*, Aruba.

Honal, M. and Schultz, T. (2003). Correction of disfluencies in spontaneous speech using a noisy-channel approach. In *EUROSPEECH-03*.

Honal, M. and Schultz, T. (2005). Automatic disfluency removal on recognized spontaneous speech - rapid adaptation to speaker-dependent disfluencies. In *IEEE ICASSP-05*.

Howes, D. (1957). On the relation between the intelligibility and frequency of occurrence of English words. *Journal of the Acoustical Society of America*, *29*, 296–305.

Huang, C., Chang, E., Zhou, J., and Lee, K.-F. (2000). Accent modeling based on pronunciation dictionary adaptation for large vocabulary mandarin speech recognition. In *ICSLP-00*, Beijing, China.

Jelinek, F. (1969). A fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, *13*, 675–685.

Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, *64*(4), 532–557.

Jelinek, F. (1997). *Statistical Methods for Speech Recognition*. MIT Press.

Jelinek, F., Mercer, R. L., and Bahl, L. R. (1975). Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory*, *IT-21*(3), 250–256.

Jones, D. A., Gibson, E., Shen, W., Granoien, N., Herzog, M., Reynolds, D., and Weinstein, C. (2005). Measuring human readability of machine generated text: Three case studies in speech recognition and machine translation. In *IEEE ICASSP-05*, pp. 18–23.

Jones, D. A., Wolf, F., Gibson, E., Williams, E., Fedorenko, E., Reynolds, D. A., and Zissman, M. (2003). Measuring the readability of automatic speech-to-text transcripts. In *EUROSPEECH-03*, pp. 1585–1588.

Juneja, A. and Espy-Wilson, C. (2003). Speech segmentation using probabilistic phonetic feature hierarchy and support vector machines. In *IJCNN 2003*.

Junqua, J. C. (1993). The Lombard reflex and its role on human listeners and automatic speech recognizers. *Journal of the Acoustical Society of America*, *93*(1), 510–524.

Jurafsky, D., Ward, W., Jianping, Z., Herold, K., Xiuyang, Y., and Sen, Z. (2001). What kind of pronunciation variation is hard for triphones to model?. In *IEEE ICASSP-01*, Salt Lake City, Utah, pp. I.577–580.

Jurafsky, D., Bell, A., Fosler-Lussier, E., Girand, C., and Raymond, W. D. (1998). Reduction of English function words in Switchboard. In *ICSLP-98*, Sydney, Vol. 7, pp. 3111–3114.

Kim, D., Gales, M., Hain, T., and Woodland, P. C. (2004). Using vtln for broadcast news transcription. In *ICSLP-04*, Jeju, South Korea.

Kim, J. and Woodland, P. (2001). The use of prosody in a combined system for punctuation generation and speech recognition. In *EUROSPEECH-01*, pp. 2757–2760.

Klovstad, J. W. and Mondshein, L. F. (1975). The CASPERS linguistic analysis system. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *ASSP-23*(1), 118–123.

Kuhn, R. and De Mori, R. (1990). A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *12*(6), 570–583.

Kumar, S. and Byrne, W. (2002). Risk based lattice cutting for segmental minimum Bayes-risk decoding. In *ICSLP-02*, Denver, CO.

Lease, M., Johnson, M., and Charniak, E. (2006). Recognizing disfluencies in conversational speech. *IEEE Transactions on Audio, Speech and Language Processing*, *14*(5), 1566–1573.

Lee, L. and Rose, R. C. (1996). Speaker normalisation using efficient frequency warping procedures. In *ICASSP96*, pp. 353–356.

Leggetter, C. J. and Woodland, P. C. (1995). Maximum likelihood linear regression for speaker adaptation of *HMM*s. *Computer Speech and Language,*, *9*(2), 171–186.

Levelt, W. J. M. (1983). Monitoring and self-repair in speech. *Cognition*, *14*, 41–104.

Liu, Y., Chawla, N. V., Harper, M. P., Shriberg, E., and Stolcke, A. (2006a). A study in machine learning from imbalanced data for sentence boundary detection in speech. *Computer Speech & Language*, *20*(4), 468–494.

Liu, Y., Shriberg, E., Stolcke, A., Hillard, D., Ostendorf, M., and Harper, M. (2006b). Enriching speech recognition with automatic detection of sentence boundaries and disfluencies. *IEEE Transactions on Audio, Speech, and Language Processing*, *14*(5), 1526–1540.

Liu, Y., Shriberg, E., Stolcke, A., Peskin, B., Ang, J., Hillard, D., Ostendorf, M., Tomalin, M., Woodland, P. C., and Harper, M. P. (2005). Structural metadata research in the ears program. In *IEEE ICASSP-05*.

Liu, Y. (2004). Word fragment identification using acoustic-prosodic features in conversational speech. In *HLT-NAACL-03 student research workshop*, pp. 37–42.

Livescu, K., Glass, J., and Bilmes, J. (2003). Hidden feature modeling for speech recognition using dynamic bayesian networks. In *EUROSPEECH-03*.

Livescu, K. (2005). *Feature-Based Pronuncaition Modeling for Automatic Speech Recognition*. Ph.D. thesis, Massachusetts Institute of Technology.

Livescu, K. and Glass, J. (2004a). Feature-based pronunciation modeling for speech recognition. In *HLT-NAACL-04*, Boston, MA.

Livescu, K. and Glass, J. (2004b). Feature-based pronunciation modeling with trainable asynchrony probabilities. In *ICSLP-04*, Jeju, South Korea.

Luce, P. A., Pisoni, D. B., and Goldfinger, S. D. (1990). Similarity neighborhoods of spoken words. In Altmann, G. T. M. (Ed.), *Cognitive Models of Speech Processing*, pp. 122–147. MIT Press.

Makhoul, J., Baron, A., Bulyko, I., Nguyen, L., Ramshaw, L., Stallard, D., Schwartz, R., and Xiang, B. (2005). The effects of speech recognition and punctuation on information extraction performance. In *INTERSPEECH-05*, Lisbon, Portugal, pp. 57–60.

Mangu, L., Brill, E., and Stolcke, A. (2000). Finding consensus in speech recognition: Word error minimization and other applications of confusion networks. *Computer Speech and Language*, *14*(4), 373–400.

Marslen-Wilson, W. and Welsh, A. (1978). Processing interactions and lexical access during word recognition in continuous speech. *Cognitive Psychology*, *10*, 29–63.

Marslen-Wilson, W. (1973). Linguistic structure and speech shadowing at very short latencies. *Nature*, *244*, 522–523.

Massaro, D. W. (1998). *Perceiving Talking Faces: From Speech Perception to a Behavioral Principle*. MIT Press.

Massaro, D. W. and Cohen, M. M. (1983). Evaluation and integration of visual and auditory information in speech perception. *Journal of Experimental Psychology: Human Perception and Performance*, *9*, 753–771.

McClelland, J. L. and Elman, J. L. (1986). Interactive processes in speech perception: The TRACE model. In McClelland, J. L., Rumelhart, D. E., and the PDP Research Group (Eds.), *Parallel Distributed Processing Volume 2: Psychological and Biological Models*, pp. 58–121. MIT Press.

McDermott, E. and Hazen, T. (2004). Minimum Classification Error training of landmark models for real-time continuous speech recognition. In *IEEE ICASSP-04*.

McGurk, H. and Macdonald, J. (1976). Hearing lips and seeing voices. *Nature*, *264*, 746–748.

Miller, J. L. (1994). On the internal structure of phonetic categories: a progress report. *Cognition*, *50*, 271–275.

Murveit, H., Butzberger, J. W., Digalakis, V. V., and Weintraub, M. (1993). Large-vocabulary dictation using SRI's decipher speech recognition system: Progressive-search techniques. In *IEEE ICASSP-93*, Vol. 2, pp. 319–322. IEEE.

Nadas, A. (1983). A decision theorectic formulation of a training problem in speech recognition and a comparison of training by unconditional versus conditional maximum likelihood. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *31*(4), 814–817.

Nakatani, C. and Hirschberg, J. (1994). A corpus-based study of repair cues in spontaneous speech. *Journal of the Acoustical Society of America*, *95*(3), 1603–1616.

Ney, H., Haeb-Umbach, R., Tran, B.-H., and Oerder, M. (1992). Improvements in beam search for 10000-word continuous speech recognition. In *IEEE ICASSP-92*, San Francisco, CA, pp. I.9–12. IEEE.

Nguyen, L. and Schwartz, R. (1999). Single-tree method for grammar-directed search. In *IEEE ICASSP-99*, pp. 613–616. IEEE.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA.

Niyogi, P., Burges, C., and Ramesh, P. (1998). Distinctive feature detection using support vector machines. In *IEEE ICASSP-98*.

Normandin, Y. (1996). Maximum mutual information estimation of hidden Markov models. In Lee, C. H., Soong, F. K., and Paliwal, K. K. (Eds.), *Automatic Speech and Speaker Recognition*, pp. 57–82. Kluwer.

Odell, J. J. (1995). *The Use of Context in Large Vocabulary Speech Recognition*. Ph.D. thesis, Queen's College, University of Cambridge.

Oden, G. C. and Massaro, D. W. (1978). Integration of featural information in speech perception. *Psychological Review*, *85*, 172–191.

Ortmanns, S., Ney, H., and Aubert, X. (1997). A word graph algorithm for large vocabulary continuous speech recognition. *Computer Speech and Language,*, *11*, 43–72.

Ostendorf, M., Digilakis, V., and Kimball, O. (1996). From HMMs to segment models: A unified view of stochastic modeling for speech recognition. *IEEE Transactions on Speech and Audio*, *4*(5), 360–378.

Ostendorf, M. and Ross, K. (1997). Multi-level recognition of intonation labels. In Sagisaka, Y., Campbell, N., and Higuchi, N. (Eds.), *Computing Prosody: Computational Models for Processing Spontaneous Speech*, chap. 19, pp. 291–308. Springer.

Paul, D. B. (1991). Algorithms for an optimal A* search and linearizing the search in the stack decoder. In *IEEE ICASSP-91*, Vol. 1, pp. 693–696. IEEE.

Pearl, J. (1984). *Heuristics*. Addison-Wesley, Reading, MA.

Ravishankar, M. K. (1996). *Efficient Algorithms for Speech Recognition*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh. Available as CMU CS tech report CMU-CS-96-143.

Rosenberg, A. E. (1971). Effect of Glottal Pulse Shape on the Quality of Natural Vowels. *The Journal of the Acoustical Society of America*, *49*, 583–590.

Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, *10*, 187–228.

Salasoo, A. and Pisoni, D. B. (1985). Interaction of knowledge sources in spoken word identification. *Journal of Memory and Language*, *24*, 210–231.

Samuel, A. G. (1981). Phonemic restoration: Insights from a new methodology. *Journal of Experimental Psychology: General*, *110*, 474–494.

Saraclar, M., Nock, H., and Khudanpur, S. (2000). Pronunciation modeling by sharing gaussian densities across phonetic models. *Computer Speech and Language*, *14*(2), 137–160.

Schwartz, R. and Austin, S. (1991). A comparison of several approximate algorithms for finding multiple (*N*-BEST) sentence hypotheses. In *icassp91*, Toronto, Vol. 1, pp. 701–704. IEEE.

Schwartz, R. and Chow, Y.-L. (1990). The N-best algorithm: An efficient and exact procedure for finding the N most likely sentence hypotheses. In *IEEE ICASSP-90*, Vol. 1, pp. 81–84. IEEE.

Schwartz, R., Chow, Y.-L., Kimball, O., Roukos, S., Krasnwer, M., and Makhoul, J. (1985). Context-dependent modeling for acoustic-phonetic recognition of continuous speech. In *IEEE ICASSP-85*, Vol. 3, pp. 1205–1208. IEEE.

Shriberg, E. (2002). To 'errrr' is human: ecology and acoustics of speech disfluencies. *Journal of the International Phonetic Association*, *31*(1), 153–169.

Shriberg, E. (2005). Spontaneous speech: How people really talk, and why engineers should care. In *INTERSPEECH-05*, Lisbon, Portugal.

Shriberg, E., Stolcke, A., Hakkani-Tür, D., and Tür, G. (2000). Prosody-based automatic segmentation of speech into sentences and topics. *Speech Communication*, *32*(1-2), 127–154.

Shriberg, E. (1994). *Preliminaries to a Theory of Speech Disfluencies*. Ph.D. thesis, University of California, Berkeley, CA. (unpublished).

Soong, F. K. and Huang, E.-F. (1990). A tree-trellis based fast search for finding the n-best sentence hypotheses in continuous speech recognition. In *Proceedings DARPA Speech and Natural Language Processing Workshop*, Hidden Valley, PA, pp. 705–708. Also in Proceedings of IEEE ICASSP-91, 705-708.

Stolcke, A. (2002). Srilm - an extensible language modeling toolkit. In *ICSLP-02*, Denver, CO.

Tomokiyo, L. M. and Waibel, A. (2001). Adaptation methods for non-native speech. In *Proceedings of Multilinguality in Spoken Language Processing*, Aalborg, Denmark.

Tyler, L. K. (1984). The structure of the initial cohort: Evidence from gating. *Perception & Psychophysics*, *36*(5), 417–427.

Wang, M. Q. and Hirschberg, J. (1992). Automatic classification of intonational phrasing boundaries. *Computer Speech and Language*, *6*(2), 175–196.

Wang, Z., Schultz, T., and Waibel, A. (2003). Comparison of acoustic model adaptation techniques on non-native speech. In *IEEE ICASSP*, Vol. 1, pp. 540–543.

Ward, W. (1989). Modelling non-verbal sounds for speech recognition. In *HLT '89: Proceedings of the Workshop on Speech and Natural Language*, Cape Cod, MA, pp. 47–50.

Warren, R. M. (1970). Perceptual restoration of missing speech sounds. *Science*, *167*, 392–393.

Wegmann, S., McAllaster, D., Orloff, J., and Peskin, B. (1996). Speaker normalisation on conversational telephone speech. In *IEEE ICASSP-96*, Atlanta, GA.

Weintraub, M., Taussig, K., Hunicke-Smith, K., and Snodgras, A. (1996). Effect of speaking style on LVCSR performance. In *ICSLP-96*, Philadelphia, PA, pp. 16–19.

Welling, L., Ney, H., and Kanthak, S. (2002). Speaker adaptive modeling by vocal tract normalisation. *IEEE Transactions on Speech and Audio Processing*, *10*, 415–426.

Woodland, P. C., Leggetter, C. J., Odell, J. J., Valtchev, V., and Young, S. J. (1995). The 1994 htk large vocabulary speech recognition system. In *IEEE ICASSP*.

Woodland, P. C. and Povey, D. (2002). Large scale discriminative training of hidden Markov models for speech recognition. *Computer Speech and Language,*, *16*, 25–47.

Woodland, P. C. (2001). Speaker adaptation for continuous density HMMs: A review. In Juncqua, J.-C. and Wellekens, C. (Eds.), *Proceedings of the ITRW 'Adaptation Methods For Speech Recognition'*, Sophia-Antipolis, France.

Young, S. J. (1984). Generating multiple solutions from connected word dp recognition algorithms. *Proceedings of the Institute of Acoustics*, *6*(4), 351–354.

Young, S. J., Odell, J. J., and Woodland, P. C. (1994). Tree-based state tying for high accuracy acoustic modelling. In *Proceedings ARPA Workshop on Human Language Technology*, pp. 307–312.

Young, S. J., Russell, N. H., and Thornton, J. H. S. (1989). Token passing: A simple conceptual model for connected speech recognition systems. Tech. rep. CUED/F-INFENG/TR.38, Cambridge University Engineering Department, Cambridge, England.

Young, S. J. and Woodland, P. C. (1994). State clustering in HMM-based continuous speech recognition. *Computer Speech and Language*, *8*(4), 369–394.

Young, S. J., Evermann, G., Gales, M., Hain, T., Kershaw, D., Moore, G., Odell, J. J., Ollason, D., Povey, D., Valtchev, V., and Woodland, P. C. (2005). *The HTK Book*. Cambridge University Engineering Department.

Zheng, Y., Sproat, R., Gu, L., Shafran, I., Zhou, H., Su, Y., Jurafsky, D., Starr, R., and Yoon, S.-Y. (2005). Accent detection and speech recognition for shanghai-accented mandarin. In *InterSpeech 2005*, Lisbon, Portugal.

Zweig, G. (1998). *Speech Recognition with Dynamic Bayesian Networks*. Ph.D. thesis, University of California, Berkeley.

11 COMPUTATIONAL PHONOLOGY

bidakupadotigolabubidakutupiropadotigolabutupirobidaku...
Word segmentation stimulus (Saffran et al., 1996a)

Recall from Ch. 7 that **phonology** is the area of linguistics that describes the systematic way that sounds are differently realized in different environments, and how this system of sounds is related to the rest of the grammar. This chapter introduces **computational phonology**, the use of computational models in phonological theory.

COMPUTATIONAL PHONOLOGY

One focus of computational phonology is on computational models of phonological representation, and on how to use phonological models to map from surface phonological forms to underlying phonological representation. Models in (non-computational) phonological theory are generative; the goal of the model is to represent how an underlying form can generate a surface phonological form. In computation, we are generally more interested in the alternative problem of **phonological parsing**; going from surface form to underlying structure. One major tool for this task is the finite-state automaton, which is employed in two families of models: **finite-state phonology** and **optimality theory**.

A related kind of phonological parsing task is **syllabification**: the task of assigning syllable structure to sequences of phones. Besides its theoretical interest, syllabification turns out to be a useful practical tool in aspects of speech synthesis such as pronunciation dictionary design. We therefore summarize a few practical algorithms for syllabification.

Finally, we spend the remainder of the chapter on the key problem of how phonological and morphological representations can be learned.

11.1 Finite-State Phonology

Ch. 3 showed that spelling rules can be implemented by transducers. Phonological rules can be implemented as transducers in the same way; indeed the original work by Johnson (1972) and Kaplan and Kay (1981) on finite-state models was based on phonological rules rather than spelling rules. There are a number of different models of **computational phonology** that use finite automata in various ways to realize phonological rules. We will describe the **two-level morphology** of Koskenniemi (1983) first mentioned in Ch. 3. Let's begin with the intuition, by seeing the transducer in Fig. 11.1 which models the simplified flapping rule in (11.1):

(11.1) $$/\text{t}/ \rightarrow [\text{dx}] \;/\; \acute{\text{V}} \;\_\_\_\; \text{V}$$

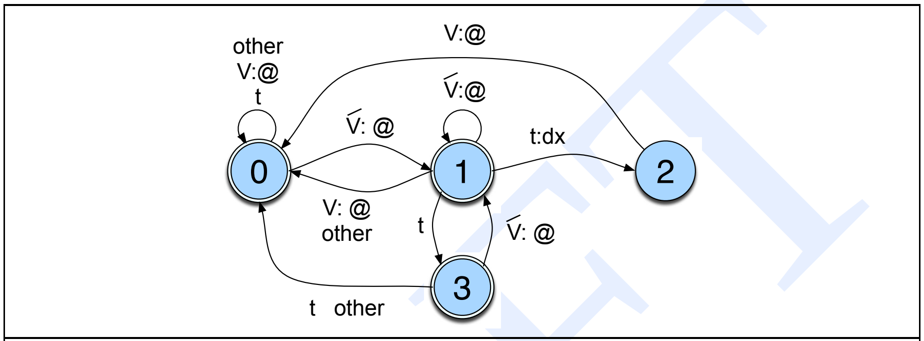


Figure 11.1 Transducer for English Flapping: ARPAbet "dx" indicates a flap, and the "other" symbol means "any feasible pair not used elsewhere in the transducer". "@" means "any symbol not used elsewhere on any arc".

The transducer in Fig. 11.1 accepts any string in which flaps occur in the correct places (after a stressed vowel, before an unstressed vowel), and rejects strings in which flapping doesn't occur, or in which flapping occurs in the wrong environment.[1]

We've seen both transducers and rules before; the intuition of two-level morphology is to augment the rule notation to correspond more naturally to transducers. We motivate his idea by beginning with the notion of **rule ordering**. In a traditional phonological system, many different phonological rules apply between the lexical form and the surface form. Sometimes these rules interact; the output from one rule affects the input to another rule. One way to implement rule-interaction in a transducer system is to run transducers in a *cascade*. Consider, for example, the rules that are needed to deal with the phonological behavior of the English noun plural suffix *-s*. This suffix is pronounced [ix z] after the phones [s], [sh], [z], [zh], [ch], or [jh] (so *peaches* is pronounced [p iy ch ix z], and *faxes* is pronounced [f ae k s ix z]), [z] after voiced sounds (*pigs* is pronounced [p ih g z]), and [s] after unvoiced sounds (*cats* is pronounced [k ae t s]). We model this variation by writing phonological rules for the realization of the morpheme in different contexts. We first need to choose one of these three forms ([s], [z], [ix z]) as the "lexical" pronunciation of the suffix; we chose [z] only because it turns out to simplify rule writing. Next we write two phonological rules. One, similar to the E-insertion spelling rule of page **??**, inserts an [ix] after a morpheme-final sibilant and before the plural morpheme [z]. The other makes sure that the *-s* suffix is properly realized as [s] after unvoiced consonants.

(11.2) $$\epsilon \;\rightarrow\; \text{ix} \;/\; [\text{+sibilant}] \;\hat{}\; \_\_\_ \; \text{z}\;\#$$

[1] For pedagogical purposes, this example assumes (incorrectly) that the factors that influence flapping are purely phonetic and are non-stochastic.

(11.3) z → s / [-voice] ˆ ___ #

These two rules must be *ordered*; rule (11.2) must apply before (11.3). This is because the environment of (11.2) includes z, and the rule (11.3) changes z. Consider running both rules on the lexical form *fox* concatenated with the plural *-s*:

Lexical form:	f aa k ˆ z
(11.2) applies:	f aa k s ˆ ix z
(11.3) doesn't apply:	f aa k sˆ ix z

BLEEDING

If the devoicing rule (11.3) was ordered first, we would get the wrong result. This situation, in which one rule destroys the environment for another, is called **bleeding**:[2]

Lexical form:	f aa k s ˆ z
(11.3) applies:	f aa k s ˆ s
(11.2) doesn't apply:	f aa k s ˆ s

As was suggested in Ch. 3, each of these rules can be represented by a transducer. Since the rules are ordered, the transducers would also need to be ordered. For example if they are placed in a **cascade**, the output of the first transducer would feed the input of the second transducer.

Many rules can be cascaded together this way. As Ch. 3 discussed, running a cascade, particularly one with many levels, can be unwieldy, and so transducer cascades are usually replaced with a single more complex transducer by **composing** the individual transducers.

Koskenniemi's method of **two-level morphology** that was sketchily introduced in Ch. 3 is another way to solve the problem of rule ordering. Koskenniemi (1983) observed that most phonological rules in a grammar are independent of one another; that feeding and bleeding relations between rules are not the norm.[3] Since this is the case, Koskenniemi proposed that phonological rules be run in parallel rather than in series. The cases where there is rule interaction (feeding or bleeding) we deal with by slightly modifying some rules. Koskenniemi's two-level rules can be thought of as a way of expressing **declarative constraints** on the well-formedness of the lexical-surface mapping.

Two-level rules also differ from traditional phonological rules by explicitly coding when they are obligatory or optional, by using four differing **rule operators**; the ⇔ rule corresponds to traditional **obligatory** phonological rules, while the ⇒ rule implements **optional rules**:

Rule type	Interpretation
a:b ⇐ c ___ d	*a* is **always** realized as *b* in the context *c* ___ *d*
a:b ⇒ c ___ d	*a* may be realized as *b* **only** in the context *c* ___ *d*
a:b ⇔ c ___ d	*a* must be realized as *b* in context *c* ___ *d* and nowhere else
a:b /⇐ c ___ d	*a* is **never** realized as *b* in the context *c* ___ *d*

[2] If we had chosen to represent the lexical pronunciation of *-s* as [s] rather than [z], we would have written the rule inversely to voice the *-s* after voiced sounds, but the rules would still need to be ordered; the ordering would simply flip.

[3] Feeding is a situation in which one rule creates the environment for another rule and so must be run beforehand.

The most important intuition of the two-level rules, and the mechanism that lets them avoid feeding and bleeding, is their ability to represent constraints on *two levels*. This is based on the use of the colon (":"), which was touched on very briefly in Ch. 3. The symbol *a:b* means a lexical *a* that maps to a surface *b*. Thus $a{:}b \Leftrightarrow {:}c\ \underline{\quad}$ means *a* is realized as *b* after a **surface** c. By contrast $a{:}b \Leftrightarrow c{:}\ \underline{\quad}$ means that *a* is realized as *b* after a **lexical** c. As discussed in Ch. 3, the symbol *c* with no colon is equivalent to *c:c* that means a lexical *c* which maps to a surface *c*.

Fig. 11.2 shows an intuition for how the two-level approach avoids ordering for the ix-insertion and z-devoicing rules. The idea is that the z-devoicing rule maps a *lexical* z-insertion to a *surface* s and the ix rule refers to the *lexical* z.

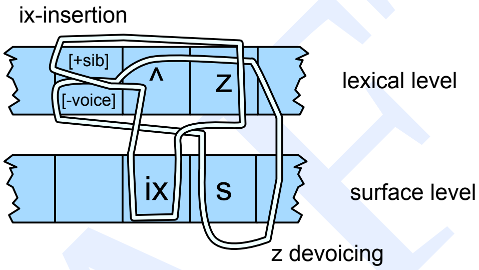


Figure 11.2 The constraints for the i-insertion and z-devoicing rules both refer to a *lexical* z, not a *surface* z.

The two-level rules that model this constraint are shown in (11.4) and (11.5):

(11.4) $$\epsilon : \text{ix} \;\Leftrightarrow\; [+\text{sibilant}]{:}\ \hat{}\ \underline{\quad}\ \text{z}{:}\ \#$$

(11.5) $$\text{z} : \text{s} \;\Leftrightarrow\; [-\text{voice}]{:}\ \hat{}\ \underline{\quad}\ \#$$

As Ch. 3 discussed, there are compilation algorithms for creating automata from rules. Kaplan and Kay (1994) give the general derivation of these algorithms, and Antworth (1990) gives one that is specific to two-level rules. The automata corresponding to the two rules are shown in Fig. 11.3 and Fig. 11.4. Fig. 11.3 is based on Figure 3.14 of Ch. 3; see page 78 for a reminder of how this automaton works. Note in Fig. 11.3 that the plural morpheme is represented by z:, indicating that the constraint is expressed about a lexical rather than surface z.

Fig. 11.5 shows the two automata run in parallel on the input [f aa k s ˆ z]. Note that both the automata assumes the default mapping $\hat{}{:}\epsilon$ to remove the morpheme boundary, and that both automata end in an accepting state.

11.2 Advanced Finite-State Phonology

11.2.1 Harmony

Finite-state models of phonology have also been applied to more sophisticated phonological and morphological phenomena. Let's consider a finite-state model of a well-

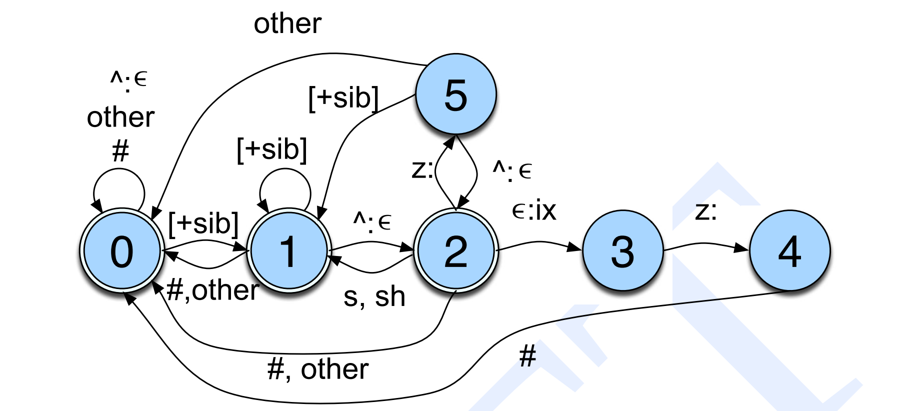


Figure 11.3 The transducer for the ix-insertion rule 11.2. The rule can be read *whenever a morpheme ends in a sibilant, and the following morpheme is word-final* z*, insert [ix].*

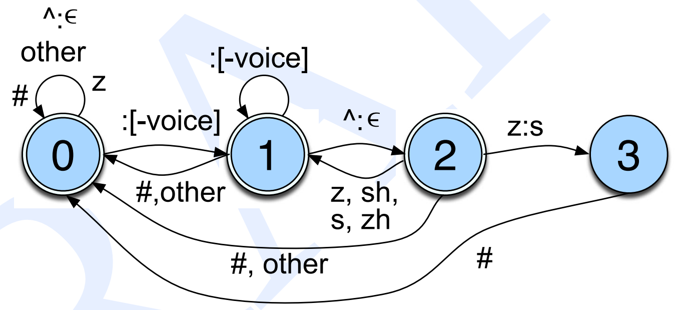


Figure 11.4 The transducer for the z-devoicing rule 11.3. This rule might be summarized *Devoice the morpheme z if it follows a morpheme-final voiceless consonant.*

known complex interaction of three phonological rules in the Yawelmani dialect of Yokuts, a Native American language spoken in California.[4]

VOWEL HARMONY

First, Yokuts (like many other languages including for example Turkish and Hungarian) has **vowel harmony**. Vowel harmony is a process in which a vowel changes its form to look like a neighboring vowel. In Yokuts, a suffix vowel changes its form to agree in backness and roundness with the preceding stem vowel. That is, a front vowel like /i/ will appear as a back vowel [u] if the stem vowel is /u/. This **Harmony** rule applies if the suffix and stem vowels are of the same height (e.g., /u/ and /i/ both high, /o/ and /a/ both low): [5]

[4] These rules were first drawn up in the traditional Chomsky and Halle (1968) format by Kisseberth (1969) following the field work of Newman (1944).

[5] Examples from Cole and Kisseberth (1995). Some parts of system such as vowel underspecification have been removed for pedagogical simplification (Archangeli, 1984).

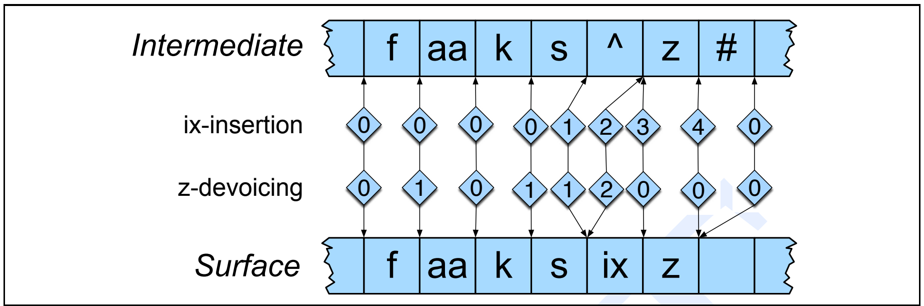


Figure 11.5 The transducer for the ix-insertion rule 11.2 and the z-devoicing rule 11.3 run in parallel.

	High Stem				Low Stem			
	Lexical		Surface	Gloss	Lexical		Surface	Gloss
Harmony	dub+hin	→	dubhun	“tangles”	bok’+al	→	bok’ol	“might eat”
No Harmony	xil+hin	→	xilhin	“leads by the hand”	xat’+al	→	xat’al	“might find”

The second relevant rule, **Lowering**, causes long high vowels to become low; /uː/ becomes [oː] and /iː/ becomes [eː], while the third rule, **Shortening**, shortens long vowels in closed syllables:

Lowering				Shortening		
ʔuːt’+it	→	ʔoːt’ut	“steal, passive aorist”	sːap+hin	→	saphin
miːk’+it	→	meːk’+it	“swallow, passive aorist”	suduːk+hin	→	sudokhun

The three Yokuts rules must be ordered, just as the ix-insertion and z-devoicing rules had to be ordered. Harmony must be ordered before Lowering because the /uː/ in the lexical form /ʔuːt’+it/ causes the /i/ to become [u] before it lowers in the surface form [ʔoːt’ut]. Lowering must be ordered before Shortening because the /uː/ in /suduːk+hin/ lowers to [o]; if it was ordered after shortening it would appear on the surface as [u].

The Yokuts data can be modeled either as a cascade of three rules in series, or in the two-level formalism as three rules in parallel; Fig. 11.6 shows the two architectures (Goldsmith, 1993; Lakoff, 1993; Karttunen, 1998). Just as in the two-level examples presented earlier, the rules work by referring sometimes to the lexical context, sometimes to the surface context; writing the rules is left as Exercise 11.4 for the reader.

11.2.2 Templatic Morphology

Finite-state models of phonology and morphology have also been proposed for the templatic (non-concatenative) morphology (discussed on page **??**) common in Semitic languages like Arabic, Hebrew, and Syriac.

There are a number of computational finite-state implementations of non-concatenative morphology. Many of them draw on the the CV approach of McCarthy (1981), in which a word like /katab/ is represented by three separate morphemes; a **root morpheme** consisting of consonants (*ktb*), a **vocalic morpheme** consisting of vowels (*a*), and a CV

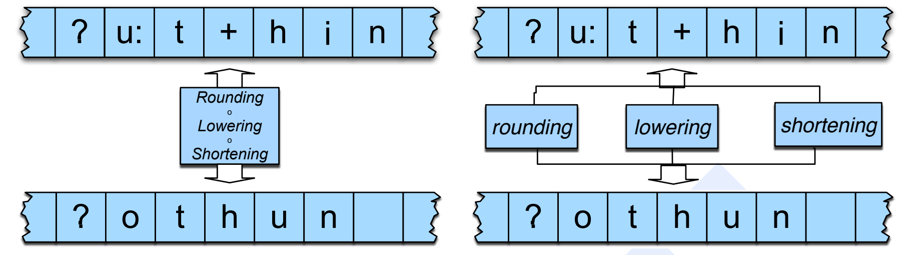


Figure 11.6 Combining the rounding, lowering, and shortening rules for Yawelmani Yokuts.

pattern morpheme (sometimes called a **binyan** or a **CV skeleton**) (*CVCVC*). McCarthy represented this morphemes on three separate morphological **tiers** (Goldsmith, 1976).

TIERS

An influential model by Kay (1987), for example, uses separate tapes for each of McCarthy's tiers. A high-level intuition of Kay's model is shown in Fig. 11.7, which shows his special transducer that reads four tapes instead of two.

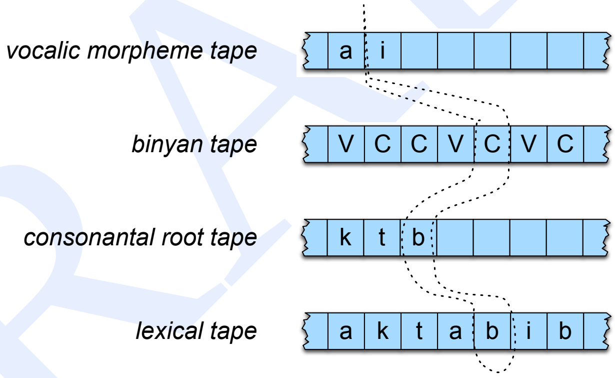


Figure 11.7 A finite-state model of templatic ("non-concatenative") morphology. Adapted from Kay (1987) and Sproat (1993).

The complication with such a multi-tape model is designing a machine which aligns the various strings on the tapes in the correct way; Kay proposed that the binyan tape could act as a sort of guide for alignment. Kay's intuition has led to a number of more fully worked out finite-state models of Semitic morphology; see the end of the chapter for details of these models, as well as alternatives based on new finite-state operations.

11.3 Computational Optimality Theory

In a traditional phonological derivation, we are given an underlying lexical form and a surface form. The phonological system then consists of a sequence of rules which

OPTIMALITY THEORY
OT

map the underlying form to the surface form. **Optimality Theory** (**OT**) (Prince and Smolensky, 1993) offers an alternative way of viewing phonological derivation, based on the metaphor of filtering rather than transforming. An OT model includes two functions (GEN and EVAL) and a set of ranked violable constraints (CON). Given an underlying form, the GEN function produces all imaginable surface forms, even those which couldn't possibly be a legal surface form for the input. The EVAL function then applies each constraint in CON to these surface forms in order of constraint rank. The surface form which best meets the constraints is chosen.

Let's briefly introduce OT, using some Yawlemani data, and then turn to the computational ramifications.[6] In addition to the interesting vowel harmony phenomena discussed above, Yawelmani has phonotactic constraints that rule out sequences of consonants; three consonants in a row (CCC) are not allowed to occur in a surface word. Sometimes, however, a word contains two consecutive morphemes such that the first one ends in two consonants and the second one starts with one consonant (or vice versa). What does the language do to solve this problem? It turns out that Yawelmani either deletes one of the consonants or inserts a vowel in between.

If a stem ends in a C, and its suffix starts with CC, the first C of the suffix is deleted ("+" here means a morpheme boundary):

(11.6) **C-deletion:** $C \rightarrow \epsilon$ / C + ___ C

For example, simplifying somewhat, the CCVC "passive consequent adjunctive" morpheme hneːl drops the initial C if the previous morpheme ends in a consonant. Thus after diyel "guard", we would get the form diyel-neːl-aw, "guard - passive consequent adjunctive - locative".

If a stem ends in CC and the suffix starts with C, the language instead inserts a vowel to break up the first two consonants:

(11.7) **V-insertion:** $\epsilon \rightarrow$ V / C ___ C + C

For example in i is inserted into the root ʔilk- "sing" when it is followed by the C-initial suffix -hin, "past", producing ʔilik-hin, "sang", but not when followed by a V-initial suffix like -en, "future" in ʔilken "will sing".

Kisseberth (1970) proposed that these two rules have the same function: avoiding three consonants in a row. Let's restate this in terms of syllable structure. It happens that Yawelmani syllables can only be of the form CVC or CV; complex onsets or complex codas i.e., with multiple consonants, aren't allowed. Since CVCC syllables aren't allowed on the surface, CVCC roots must be **resyllabified** when they appear on the surface. From the point of view of syllabification, then, these insertions and deletions all happen so as to allow Yawelmani words to be properly syllabified. Here's examples of resyllabifications with no change, with an insertion, and with a deletion:

RESYLLABIFIED

[6] The following explication of OT via the Yawelmani example draws heavily from Archangeli (1997) and a lecture by Jennifer Cole at the 1999 LSA Linguistic Institute.

underlying morphemes	surface syllabification	gloss
ʔilk-en	ʔil.ken	“will sing”
ʔilk-hin	ʔi.lik.hin	“sang”
diyel-hnil-aw	di.yel.neː.law	“guard - pass. cons. adjunct. - locative”

The intuition of Optimality Theory is to try to directly represent these kind of constraints on syllable structure directly, rather than using idiosyncratic insertion and deletion rules. One such constraint, *COMPLEX, says “No complex onsets or codas”. Another class of constraints requires the surface form to be identical to (faithful to) the underlying form. Thus FAITHV says “Don’t delete or insert vowels” and FAITHC says “Don’t delete or insert consonants”. Given an underlying form, the GEN function produces all possible surface forms (i.e., every possible insertion and deletion of segments with every possible syllabification) and they are ranked by the EVAL function using these (violable) constraints. The idea is that while in general insertion and deletion are dispreferred, in some languages and situations they are preferred over violating other constraints, such as those of syllable structure. Fig. 11.8 shows the architecture.

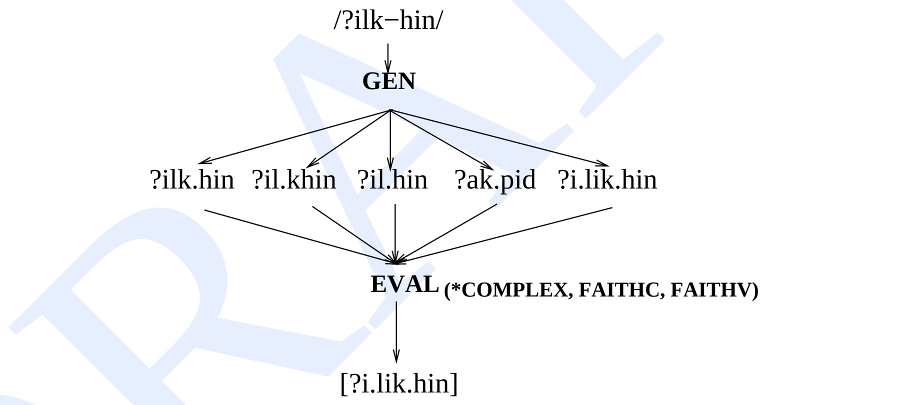


Figure 11.8 The architecture of a derivation in Optimality Theory (after Archangeli (1997)).

The EVAL function works by applying each constraint in ranked order to each candidate. Starting with the highest-ranked constraints, if one candidate either does not violate no constraints or violates less of them than all the other candidates, that candidate is declared optimal. If two candidates tie (have the same highest ranked violation), then the next-highest ranked violation is considered. This evaluation is usually shown on a **tableau** (plural **tableaux**). The top left-hand cell shows the input, the constraints are listed in order of rank across the top row, and the possible outputs along the left-most column.[7] If a form violates a constraint, the relevant cell contains *; a ***!** indicates the fatal violation which causes a candidate to be eliminated. Cells for

TABLEAU

*

*!

[7] Although there are an infinite number of candidates, it is traditional to show only the ones which are ‘close’; in the tableau below we have shown the output ʔak.pid just to make it clear that even very different surface forms are to be included.

constraints which are irrelevant (since a higher-level constraint is already violated) are shaded.

/ʔilk-hin/	*COMPLEX	FAITHC	FAITHV
ʔilk.hin	*!		
ʔil.khin	*!		
ʔil.hin		*!	
☞ ʔi.lik.hin			*
ʔak.pid		*!	

One appeal of Optimality Theoretic derivations is that the constraints are presumed to be cross-linguistic generalizations. That is all languages are presumed to have some version of faithfulness, some preference for simple syllables, and so on. Languages differ in how they rank the constraints; thus English, presumably, ranks FAITHC higher than *COMPLEX. (How do we know this?)

11.3.1 Finite-State Transducer Models of Optimality Theory

Now that we've sketched the linguistic motivations for Optimality Theory, let's turn to the computational implications. We'll explore two: implementation of OT via finite-state models, and stochastic versions of OT.

Can a derivation in Optimality Theory be implemented by finite-state transducers? Frank and Satta (1998), following the foundational work of Ellison (1994), showed that (1) if GEN is a regular relation (for example assuming the input doesn't contain context-free trees of some sort), and (2) if the number of allowed violations of any constraint has some finite bound, then an OT derivation can be computed by finite-state means. This second constraint is relevant because of a property of OT that we haven't mentioned: if two candidates violate exactly the same number of constraints, the winning candidate is the one which has the smallest number of violations of the relevant constraint.

One way to implement OT as a finite-state system was worked out by Karttunen (1998), following the above-mentioned work and that of Hammond (1997). In Karttunen's model, GEN is implemented as a finite-state transducer which is given an underlying form and produces a set of candidate forms. For example for the syllabification example above, GEN would generate all strings that are variants of the input with consonant deletions or vowel insertions, and their syllabifications.

Each constraint is implemented as a filter transducer that lets pass only strings which meet the constraint. For legal strings, the transducer thus acts as the identity mapping. For example, *COMPLEX would be implemented via a transducer that mapped any input string to itself, unless the input string had two consonants in the onset or coda, in which case it would be mapped to null.

The constraints can then be placed in a cascade, in which higher-ranked constraints are simply run first, as suggested in Fig. 11.9.

There is one crucial flaw with the cascade model in Fig. 11.9. Recall that the constraints-transducers filter out any candidate which violates a constraint. But in many derivations, including the proper derivation of ʔi.lik.hin, even the optimal form still violates a constraint. The cascade in Fig. 11.8 would incorrectly filter it out, leaving

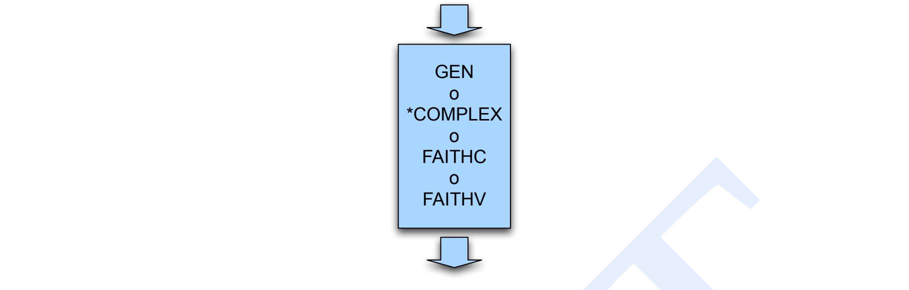


Figure 11.9 Version #1 (“merciless cascade”) of Karttunen’s finite-state cascade implementation of OT.

LENIENT COMPOSITION

no surface form at all! Frank and Satta (1998) and Hammond (1997) both point out that it is essential to only enforce a constraint if it does not reduce the candidate set to zero. Karttunen (1998) formalizes this intuition with the **lenient composition** operator. Lenient composition is a combination of regular composition and an operation called **priority union**. The basic idea is that if any candidates meet the constraint these candidates will be passed through the filter as usual. If no output meets the constraint, lenient composition retains *all* of the candidates. Fig. 11.10 shows the general idea; the interested reader should see Karttunen (1998) for the details.

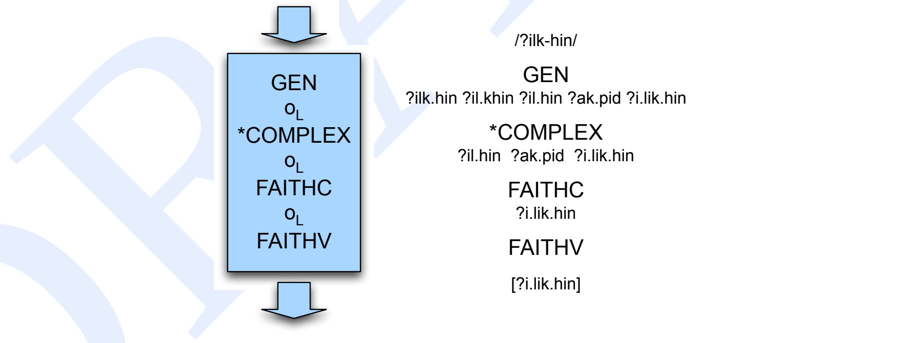


Figure 11.10 Version #2 (“lenient cascade”) of Karttunen’s finite-state cascade implementation of OT, showing a visualization of the candidate populations that would be passed through each FST constraint.

11.3.2 Stochastic Models of Optimality Theory

Classic OT was not designed to handle variation of the kind we saw in Sec. **??**, since it assigns a single most-harmonic output for each input. Dealing with variation requires a more dynamic concept of constraint ranking. We mentioned in that section the variationist model in sociolinguistics, in which logistic regression is used to combine phonetic, contextual, and social factors to predict a probability of a particular phonetic variant. Part of this variationist intuition can be absorbed into an Optimality Theory framework through probabilistic augmentations.

STOCHASTIC OT

One such augmentation is **Stochastic OT** (Boersma and Hayes, 2001). In Stochastic OT, instead of the constraints being rank-ordered, each constraint is associated with a value on a continuous scale. The continuous scale offers one thing a ranking cannot: the relative importance or weight of two constraints can be proportional to the distance between them. Fig. 11.11 shows a sketch of such a continuous scale.

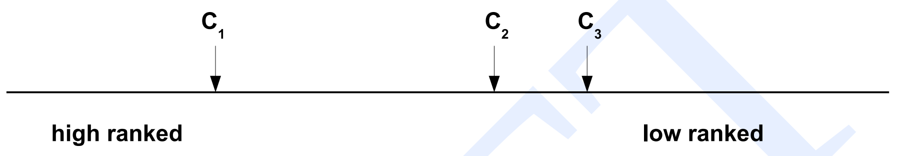


Figure 11.11 Continuous scale in Stochastic OT. After Boersma and Hayes (2001).

How can the distance between constraints play a role in evaluation? Stochastic OT makes a further assumption about the values of constraints. Instead of each constraint having a fixed value as shown in Fig. 11.11. it has a Gaussian distribution of values centered on a fixed value, as shown in Fig. 11.12. At evaluation time, a value for the constraint is drawn (a **selection point**) with a probability defined by the mean and variance of the Gaussian associated with each constraint.

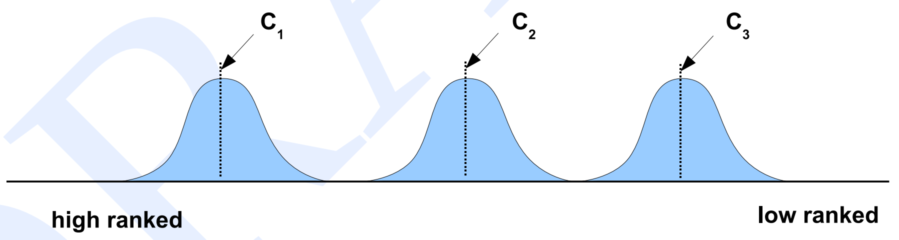


Figure 11.12 Three constraints in Stochastic OT which are strictly ranked; thus non-stochastic OT is a special case of Stochastic OT. After Boersma and Hayes (2001).

If the distribution for two constraints is far enough apart, as shown in Fig. 11.12 there will be little or no probability of the lower ranked constraint outranking the higher-ranked one. Thus Stochastic OT includes non-stochastic OT as a special case.

The interesting cases arise when two constraints in Stochastic OT overlap in their distribution, when there is some probability that a lower-ranked constraint will override a higher-ranked constraint. In Fig. 11.13, for example, constraint C_2 will generally outrank C_3 but occasionally outrank C_2. This allows Stochastic OT to model variation, since for the same underlying form differing selection points can cause different surface variants to be most highly ranked.

In addition to the advantage of modeling variation, Stochastic OT differs from non-stochastic OT in having a stochastic learning theory, which we will return to in

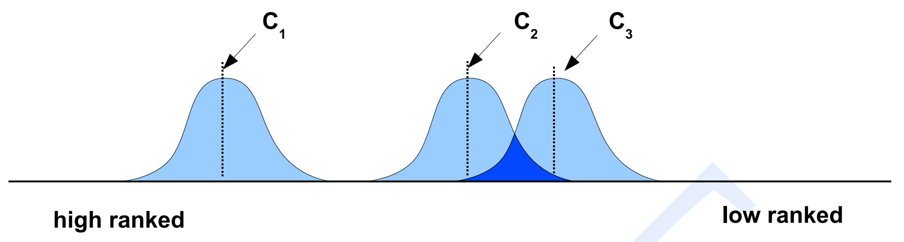


Figure 11.13 Three constraints in Stochastic OT in which C_3 will sometimes outrank C_2. . After Boersma and Hayes (2001).

Sec. 11.5.3.

We can see stochastic OT itself as a special case of the general linear models of Ch. 6.

11.4 Syllabification

SYLLABIFICATION

Syllabification, the task of segmenting a sequence of phones into syllables, is important in a variety of speech applications. In speech synthesis, syllables are important in predicting prosodic factors like accent; the realization of a phone is also dependent on its position in the syllable (onset [l] is pronounced differently than coda [l]). In speech recognition syllabification has been used to build recognizers which represent pronunciations in terms of syllables rather than phones. Syllabification can help find errors in pronunciation dictionaries, by finding words that can't be syllabified, and can help annotate corpora with syllable boundaries for corpus linguistics research. Syllabification also plays an important role in theoretical generative phonology.

One reason syllabification is a difficult computational task is that there is no completely agreed-upon definition of syllable boundaries. Different on-line syllabified dictionaries (such as the CMU and the CELEX lexicons) sometimes choose different syllabifications. Indeed, as Ladefoged (1993) points out, sometimes it isn't even clear how many syllables a word has; some words (*meal, teal, seal, hire, fire, hour*) can be viewed either as having one syllable or two.

MAXIMUM ONSET

Like much work in speech and language processing, syllabifiers can be based on hand-written rules, or on machine learning from hand-labeled training sets. What kinds of knowledge can we use in designing either kind of syllabifier? One possible constraint is the **Maximum Onset** principle, which says that when a series of consonants occur word-medially before a vowel (VCCV), as many as possible (given the other constraints of the language) should be syllabified into the onset of the second syllable rather than the coda of the first syllable. Thus the Maximum Onset principle favors the syllabification V.CCV over the syllabifications VC.CV or VCC.V.

SONORITY

SONORITY HIERARCHY

Another principle is to use the **sonority** of a sound, which is a measure of how perceptually salient, loud or vowel-like it is. There are various attempts to define a **sonority hierarchy**; in general, all things being equal, vowels are more sonorous than glides (w, y), which are more sonorous than liquids (l, r), followed by nasals (n, m, ng), fricatives (z, s, sh, zh, v, f th, dh), and stops. The sonority constraint on syllable structure says that the nucleus of the syllable must be the most sonorous phone in a sequence (the **sonority peak**), and that sonority decreases monotonically out from the nucleus (toward the coda and toward the onset). Thus in a syllable $C_1C_2VC_3C_4$, the nucleus V will be the most sonorous element, consonant C_2 will be more sonorous than C_1 and consonant C_3 will be more sonorant than consonant C_4.

Goldwater and Johnson (2005) implement a simple rule-based language-independent classifier based only on maximum onset and sonority sequencing. Given a cluster of consonants between two syllable nuclei, sonority constrains the syllable boundary to be either just before or just after the consonant with the lowest sonority. Combining sonority with maximum onset, their parser predicts a syllable boundary just before the consonant with the lowest sonority. They show that this simple syllabifier correctly syllabifies 86-87% of multisyllabic words in English and German.

While this error rate is not unreasonable, and there is further linguistic and some psychological evidence that these principles play a role in syllable structure, both Maximum Onset and sonority sequencing seem to have exceptions. For example in the English syllable-initial clusters /sp st sk/ in words like *spell*, the less sonorous /p/ occurs between the more sonorous /s/ and the vowel, violating sonority sequencing (Blevins, 1995). Without some way to rule out onset clusters that are disallowed language-specifically like /kn/ in English, the combination of sonority sequencing plus maximum onset incorrectly predicts the syllabification of words like *weakness* to be *wea.kness* rather than *weak.ness*. Furthermore, other constraints seem to be important, including whether a syllable is stressed (stressed syllables tend to have more complex codas), the presence or absence of morphological boundaries, and even the spelling of the word (Titone and Connine, 1997; Treiman et al., 2002).

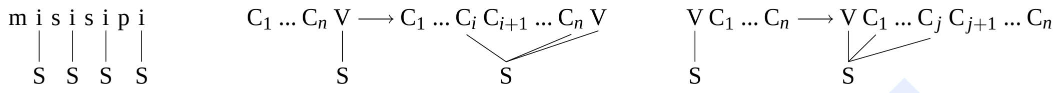
Achieving higher performance thus requires the use of these sorts of language-specific knowledge. The most commonly used rule-based syllabifier is based on the dissertation of Kahn (1976), available in an implementation by Fisher (1996). The Kahn algorithm makes use of language-specific information in the form of lists of allowable English initial initial clusters, allowable English final clusters, and 'universally bad' clusters. The algorithm takes strings of phones, together with other information like word boundaries and stress if they are available, and assigns syllable boundaries between the phones. Syllables are built up incrementally based on three rules, as sketched out in Fig. 11.14. Rule 1 forms nuclei at each syllabic segment, Rule 2a attaches onset consonants to the nucleus, and Rule 2b attaches coda consonants.[8] Rule 2a and 2b make use of lists of legal onset consonant sequences (including e.g. [b], [b l], [b r], [b y], [ch], [d], [d r], [d w], [d y], [dh], [f], [f l], [f r], [f y], [g], [g l], [g r], [g w], etc). and legal coda clusters. There are a very large number of coda consonant clusters in English; some of the longer (4-consonant) clusters include:

k s t s
k s th s
l f th s
l k t s
l t s t
m f s t
m p f t
m p s t
n d th s
n t s t
n t th s
n k s t
n k t s
n k th s
r k t s
r l d z
r m p th
r p t s
r s t s
r t s t

The algorithm also takes a parameter indicating how fast or casual the speech is;

[8] Note that the fact that Rule 2a precedes Rule 2b can be seen as an implementation of Maximum Onset.

the faster or more informal the speech, the more resyllabification happens, based on further rules we haven't shown.


m i s i s i p i
S S S S
$C_1 ... C_n V \longrightarrow C_1 ... C_i C_{i+1} ... C_n V$
S
S
$V C_1 ... C_n \longrightarrow V C_1 ... C_j C_{j+1} ... C_n$
S
S

Rule 1: Form Nuclei: link S with each [+syllabic] segment

Rule 2a: Add Onsets: where $C_{i+1}...C_n$ is a permissible initial cluster but $C_iC_{i+1}...C_n$ is not

Rule 2b: Add Codas: where $C_1...C_j$ is a permissible coda cluster but $C_1...C_jC_j+1$ is not

Figure 11.14 First three syllabification rules of Kahn (1976). Rule 2b may not apply across word boundaries.

Instead of hand-written rules, we can apply a machine learning approach, using a hand-syllabified dictionary as a supervised training set. For example the CELEX syllabified lexicon discussed in Sec. **??** is often used this way, selecting some words as a training set, and reserving others as a dev-test and test set. Statistical classifiers can be used to predict syllabifications, including decision trees (van den Bosch, 1997), weighted finite-state transducers (Kiraz and Möbius, 1998), and probabilistic context-free grammars (Seneff et al., 1996; Müller, 2002, 2001; Goldwater and Johnson, 2005).

WEIGHTED FST

For example the Kiraz and Möbius (1998) algorithm is a weighted finite-state transducer which inserts a syllable boundary in a sequence of phones (akin to the morpheme-boundaries we saw in Ch. 3). A **weighted FST** (Pereira et al., 1994) is a simple augmentation of the finite transducer in which each arc is associated with a probability as well as a pair of symbols. The probability indicates how likely that path is to be taken; the probability on all the arcs leaving a node must sum to 1.

The syllabification automaton of Kiraz and Möbius (1998) is composed of three separate weighted transducers, one for onsets, one for nuclei, and one for codas, concatenated together into an FST that inserts a syllable marker after the end of the coda. Kiraz and Möbius (1998) compute path weights from frequencies in the training set; each path (for example the nucleus [iy]) of frequency f is assigned a weight of $1/f$. Another way to convert frequencies to costs is to use log probabilities. Fig. 11.15 shows a sample automaton, simplified from Kiraz and Möbius (1998). We have shown the weights only for some of the nuclei. The arcs for each possible onset, nucleus, and coda, are drawn from a language-dependent list like the one used in the Kahn algorithm above.

The automaton shown in Fig. 11.15 can be used to map from an input sequence like the phonetic representation of *weakness* [w iy k n eh s] into an output sequence that includes the syllabification marker like "-": [w iy k - n eh s]. If there are multiple possible legal syllabifications of a word, the Viterbi algorithm is used to choose the most likely path through the FST, and hence the most probable segmentation. For example, the German word *Fenster*, "window", has three possible syllabifications: [fɛns-tɐ] <74>, [fɛn-stɐ] <75>, and [fɛnst-ɐ] <87> (with costs shown in angle brackets). Their syllabifier correctly chooses the lowest cost syllabification fɛns-tɐ, based on the frequencies of onsets and codas from the training set. Note that since morphological boundaries also are important for syllabification, the Kiraz and Möbius (1998) syllabification transducer can be placed after a morphological parsing transducer, so that

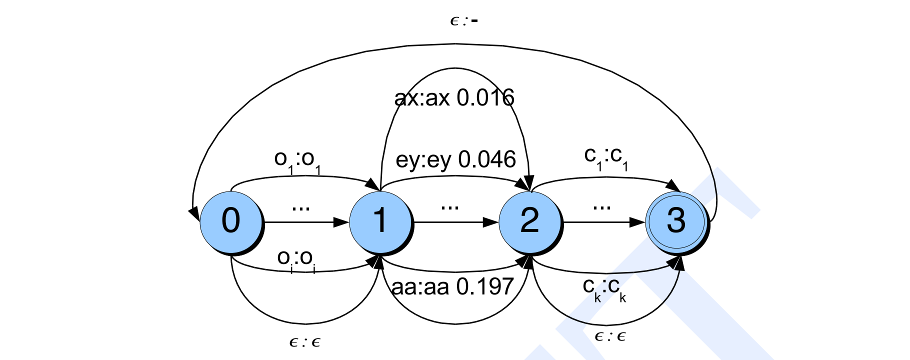


Figure 11.15 Syllabifier automaton, showing onset (o), coda (c), and nucleus arcs. Costs on each arc shown only for some sample nucleus arcs. The syllable boundary marker '-' is inserted after every non-final syllable. eps stands for ϵ. Simplified from Kiraz and Möbius (1998).

syllabification can be influenced by morphological structure.

More recent syllabifiers based on probabilistic context-free grammars (PCFGs) can model more complex hierarchical probabilistic dependencies between syllables (Seneff et al., 1996; Müller, 2002, 2001; Goldwater and Johnson, 2005). Together with other machine learning approaches like van den Bosch (1997), modern statistical syllabification approaches have a word accuracy of around 97–98% correct, and probabilistic model of syllable structure have also been shown to predict human judgments of the acceptability of nonsense words (Coleman and Pierrehumbert, 1997).

There are a number of other directions in syllabification. One is the use of unsupervised machine learning algorithms (Ellison, 1992; Müller et al., 2000; Goldwater and Johnson, 2005) Another is the use of other cues for syllabification such as allophonic details from a narrow phonetic transcription (Church, 1983).

11.5 Learning Phonology & Morphology

Machine learning of phonological structures is an active research area in computational phonology above and beyond the induction of syllable structure discussed in the previous section. Supervised learning work is based on a training set that is explicitly labeled for the phonological (or morphological) structure to be induced. Unsupervised work attempts to induce phonological or morphological structure without labeled training data. Let's look at three representative areas of learning: learning of phonological rules, learning of morphological rules, and learning of OT constraint rankings

11.5.1 Learning Phonological Rules

In this section we briefly summarize some early literature in learning phonological rules, generally couched either in terms of finite state models of two-level phonology or classic Chomsky-Halle rules.

Johnson (1984) gives one of the first computational algorithms for phonological rule induction. His algorithm works for rules of the form

$$a \rightarrow b/C \tag{11.8}$$

where C is the feature matrix of the segments around a. Johnson's algorithm sets up a system of constraint equations which C must satisfy, by considering both the positive contexts, i.e., all the contexts C_i in which a b occurs on the surface, as well as all the negative contexts C_j in which an a occurs on the surface. Touretzky et al. (1990) extended Johnsons work in various ways, including dealing with epenthesis and deletion rules.

The algorithm of Gildea and Jurafsky (1996) was designed to induce transducers representing two-level rules of the type we have discussed earlier. Gildea and Jurafsky's supervised algorithm was trained on pairs of underlying and surface forms. For example, they attempted to learn the rule of English flapping, (focusing only on the phonetic context and ignoring social and other factors). The training set thus consisted of underlying/surface pairs, either with an underlying /t/ and surface flap [dx], or an underlying /t/ and surface [t], as follows:

flapping				non-flapping			
butter	/b ah t axr/	→	[b ah dx axr]	*stop*	/s t aa p/	→	[s t aa p]
meter	/m iy t axr/	→	[m iy dx axr]	*cat*	/k ae t/	→	[k ae t]

The algorithm was based on OSTIA (Oncina et al., 1993), a general learning algorithm for the **subsequential transducers** defined on page **??**. Gildea and Jurafsky showed that by itself, the OSTIA algorithm was too general to learn phonological transducers, even given a large corpus of underlying-form/surface-form pairs. For example, given 25,000 underlying/surface pairs like the examples above, the algorithm ended up with the huge and incorrect automaton in Fig. 11.16(a). Gildea and Jurafsky then augmented the domain-independent OSTIA system with learning biases which are specific to natural language phonology. For example they added a **Faithfulness** bias that underlying segments tend to be realized similarly on the surface (i.e. that all things being equal, an underlying /p/ was likely to emerge as a surface [p]). They did this by starting OSTIA with the underlying and surface strings aligned using Levenshtein distance. They also added knowledge about phonetic features (vowel versus consonant, reduced versus non-reduced vowel, etc). Together, adding these biases enabled OSTIA to learn the automaton in Fig. 11.16(b), as well as correct automatons for other phonological rules like German consonant devoicing.

This phonological learning experiment illustrates that successful learning requires two components: a model which fits some empirical data and some prior knowledge or biases about the structure of the model.

Recent work on learning has focused either on morphological learning, or on ranking of OT constraints rather than on the induction of rules and constraints, and will be discussed in the next two sections.

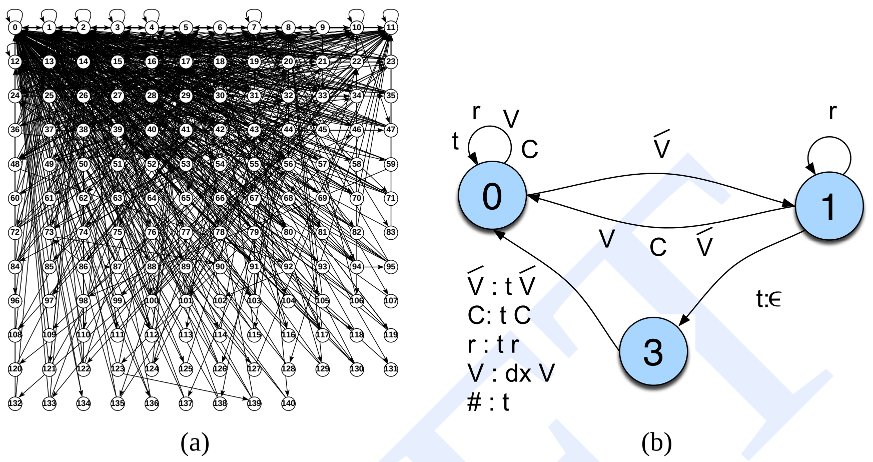


Figure 11.16 Induction of a flapping rule transducer (after Gildea and Jurafsky (1996)). The transducer in (a) is the initial attempt at learning. The transducer in (b) is the correct transducer induced after a faithfulness bias.

11.5.2 Learning Morphology

We discussed in Ch. 3 the use of finite-state transducers for morphological parsing. In general, these morphological parsers are built by hand and have relatively high accuracy, although there has also been some work on supervised machine learning of morphological parsers (van den Bosch, 1997). Recent work, however, has focused on unsupervised ways to automatically bootstrap morphological structure. The unsupervised (or weakly supervised) learning problem has practical applications, since there are many languages for which a hand-built morphological parser, or a morphological segmented training corpus, does not yet exist. In addition, the learnability of linguistic structure is a much-discussed scientific topic in linguistics; unsupervised morphological learning may help us understand what makes language learning possible.

Approaches to unsupervised morphology induction have employed a wide variety of heuristics or cues to a proper morphological parse. Early approaches were all essentially segmentation-based; given a corpus of words they attempted to segment each word into a stem and an affix using various unsupervised heuristics. For example the earliest work hypothesized morpheme boundaries at the point in a word where there is large uncertainty about the following letters (Harris, 1954, 1988; Hafer and Weiss, 1974). For example, Fig. 11.17 shows a **trie**[9] which stores the words *car*, *care*, *cars*, *cares*, *cared*, etc. Note that there there are certain nodes in the tree in Fig. 11.17 that have a wide branching factor (after *car* and after *care*). If we think of the task of predicting the next letter giving the path in the trie so far, we can say that these points have

TRIE

[9] A **trie** is a tree structure used for storing strings, in which a string is represented as a path from the root to a leaf. Each non-terminal node in the tree thus stores a prefix of a string; every common prefix is thus represented by a node. The word **trie** comes from *re***trie***val* and is pronounced either [t r iy] or [t r ay].

a high conditional entropy; there are many possible continuations.[10] While this is a useful heuristic, it is not sufficient; in this example we would need a way to rule out the morpheme *car* as well as *care* being part of the word *careful*; this requires a complex set of thresholds.

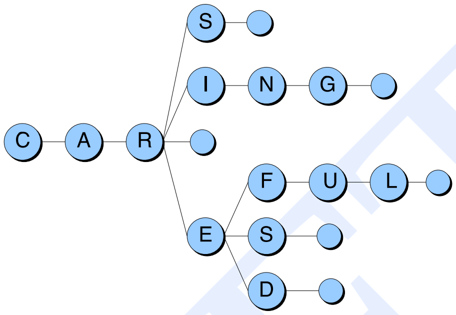


Figure 11.17 Example of a letter trie. A Harris style algorithm would insert morpheme boundaries after *car* and *care*. After Schone and Jurafsky (2000).

MINIMUM DESCRIPTION LENGTH
MDL

Another class of segmentation-based approaches to morphology induction focuses on globally optimizing a single criterion for the whole grammar, the criterion of **minimum description length**, or **MDL**. The MDL principle is widely used in language learning, and we will see it again in grammar induction in Ch. 14. The idea is that we are trying to learn the optimal probabilistic model of some data. Given any proposed model, we can assign a likelihood to the entire data set. We can also use the proposed model to assign a compressed length to this data (with probabilistic models we can use the intuition that the compressed length of the data is related to the entropy, which we can estimate from the log probability). We can also assign a length to the proposed model itself. The MDL principle says to choose the model for which the sum of the data length and the model length is the smallest. The principle is often viewed from a Bayesian perspective; If we are attempting to learn the best model $\hat{M}$ out of all models M for some data D which has the maximum a posteriori probability $P(M|D)$, we can use Bayes Rule to express the best model $\hat{M}$ as:

$$\hat{M} = argmax_M P(M|D) = argmax_M \frac{P(D|M)P(M)}{P(D)} = argmax_M P(D|M)P(M) \tag{11.9}$$

Thus the best model is the one which maximizes two terms: the likelihood of the data $P(D|M)$ and the prior of the model $P(M)$. The MDL principle can be viewed as saying that the prior term on the model should be related to the length of the model.

MDL approaches to segmentation induction were first proposed by de Marcken (1996) and Brent (1999), as well as Kazakov (1997); let's summarize from a more recent instantiation by Goldsmith (2001). The MDL intuition can be seen from the schematic example in Fig. 11.18 inspired by Goldsmith.

[10] Interestingly, this idea of placing boundaries at regions of low predictability has been shown to be used by infants for word segmentation (Saffran et al., 1996b).

$$\left\{\begin{array}{l}\text{cooked cooks cooking}\\ \text{played plays playing}\\ \text{boiled boils boiling}\end{array}\right\}$$

(a) Word list with no structure
Total letter count: 54

$$\left\{\begin{array}{l}\text{cook}\\ \text{play}\\ \text{boil}\end{array}\right\}\left\{\begin{array}{l}\text{ed}\\ \text{s}\\ \text{ing}\end{array}\right\}$$

(b) Word list with morphological structure
Total letter count: 18 letters

Figure 11.18 Naive version of MDL, showing the reduction in the description length of a lexicon with morphological structure; adapted from Goldsmith (2001).

As we see in Fig. 11.18, using morphological structure makes it possible to represent a lexicon with far fewer letters. Of course this example doesn't represent the true complexity of morphological representations, since in reality not every word is combinable with every affix. One way to represent slightly more complexity is to use **signatures**. A signature is a list of suffixes that can appear with a particular stem. Here are some sample signatures from Goldsmith (2001):

SIGNATURES

Signature	Example
NULL.ed.ing.s	remain remained remaining remains
NULL.s	cow cows
e.ed.es.ing	notice noticed notices noticing

The Goldsmith (2001) version of MDL considers all possible segmentations of every word into a stem and a suffix. It then chooses the set of segmentations for the whole corpus that jointly minimize the compressed length of the corpus and the length of the model. The length of the model is the sum of the lengths of the affixes, the stems, and the signatures. The length of the corpus is computed by using the model to assign a probability to the corpus and using this probably to compute the cross-entropy of the corpus given the model.

While approaches based solely on stem and affix statistics like MDL have been quite successful in morphological learning, they do have a number of limitations. For example Schone and Jurafsky (2000, 2001) noted in an error analysis that MDL sometimes segments valid affixes inappropriately (such as segmenting the word *ally* to *all+y*), or fails to segment valid but non-productive affixes (missing the relationship between *dirt* and *dirty*). They argued that such problems stemmed from a lack of semantic or syntactic knowledge, and showed how to use relatively simple semantic features to address them. The Schone and Jurafsky (2000) algorithm uses a trie to come up with “pairs of potential morphological variants”, (PPMVs) words which differ only in potential affixes. For each pair, they compute the semantic similarity between the words, using the Latent Semantic Analysis (LSA) algorithm of Ch. 23. LSA is an unsupervised model of word similarity which is induced directly from the distributions of word in context. Schone and Jurafsky (2000) showed that using the semantic similarity alone was at least as good a predictor of morphological structure as MDL. The table below shows the LSA-based similarity between PPMVs; in this example the similarity is high only for words that are morphologically related.

PPMVs	Score	PPMV	Score	PPMV	Score	PPMV	Score
ally/allies	6.5	dirty/dirt	2.4	car/cares	-0.14	car/cared	-.096
car/cars	5.6	rating/rate	0.97	car/caring	-0.71	ally/all	-1.3

Schone and Jurafsky (2001) extended the algorithm to learn prefixes and circumfixes, and incorporated other useful features, including syntactic and other effects of neighboring word context (Jacquemin, 1997), and the Levenshtein distance between the PPMVs (Gaussier, 1999).

The algorithms we have mentioned so far have focused on the problem of learning regular morphology. Yarowsky and Wicentowski (2000) focused on the more complex problem of learning irregular morphology. Their idea was to probabilistically align an inflected form (such as English *took* or Spanish *juegan*) with each potential stem (such as English *take* or Spanish *jugar*). The result of their alignment-based algorithm was a inflection-root mapping, with both an optional stem change and a suffix, as shown in the following table:

English				Spanish			
root	inflection	stem change	suffix	root	inflection	stem change	suffix
take	took	ake→ook	+ϵ	jugar	juega	gar→eg	+a
take	taking	e→ ϵ	+ing	jugar	jugamos	ar→ ϵ	+amos
skip	skipped	ϵ →p	+ed	tener	tienen	ener→ien	+en

The Yarowsky and Wicentowski (2000) algorithm requires somewhat more information than the algorithms for inducing regular morphology. In particular it assumes knowledge of the regular inflectional affixes of the language and a list of open class stems; both are things that might be induced by the MDL or other algorithms mentioned above. Given an inflected form, the Yarowsky and Wicentowski (2000) algorithm uses various knowledge sources to weight the potential stem, including the relative frequency of the inflected form and potential stem, the similarity in lexical context, and the Levenshtein distance between them.

11.5.3 Learning in Optimality Theory

Let's conclude with a brief sketch of work which addresses the learning problem in Optimality Theory. Most work on OT learning has assumed that the constraints are already given, and the task is to learn the ranking. Two algorithms for learning rankings have been worked out in some detail; the **constraint demotion** algorithm of Tesar and Smolensky (2000) and the **Gradual Learning Algorithm** of Boersma and Hayes (2001).

CONSTRAINT DEMOTION

The **Constraint Demotion** algorithm makes two assumptions: that we know all the possible OT constraints of the language, and that each surface form is annotated with its complete parse and underlying form. The intuition of the algorithm is that each of these surface observations gives us implicit evidence about the constraint ranking.

Given the underlying form, we can use the GEN algorithm to implicitly form the set of competitors. Now we can construct a set of pairs consisting of the correct observed grammatical form and each competitor. The learner must find a constraint ranking that prefers the observed learning *winner* over each (non-observed) competitor *loser*.

Because the set of constraints is given, we can use the standard OT parsing architecture to determine for each winner or loser exactly which constraints they violate.

For example, consider the learning algorithm that has observed Candidate 1, but whose current constraint ranking prefers Candidate 2, as follows (this example and the following tables are modified from Boersma and Hayes (2001)):

	/underlying form/	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
	Candidate 1 (learning observation)	*!	**	*		*			*
☞	Candidate 2 (learner's output)		*	*	*		*		*

Given a set of such *winner/loser* pairs, the Constraint Demotion algorithm needs to demote each constraint that is violated by the winner Candidate 2, until the observed form (Candidate 1) is preferred. The algorithm first cancels any marks due to violations that are identical between the two candidates:

	/underlying form/	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
	Candidate 1 (learning observation)	*!	*~~*~~	~~*~~		*			~~*~~
☞	Candidate 2 (learner's output)		~~*~~	~~*~~	*		*		~~*~~

These constraints are pushed down in the hierarchy until they are dominated by the constraints violated by the loser. The algorithm divides constraints into **strata**, and tries to find a lower strata to move the constraints into. Here's shows a simplification of this intuition, as C_1 and C_2 get moved below C_8.

	/underlying form/	C_3	C_4	C_5	C_6	C_7	C_8	C_1	C_2
☞	Candidate 1 (learning observation)				*			*	*
	Candidate 2 (learner's output)		*!		*				

GRADUAL LEARNING ALGORITHM

The **Gradual Learning Algorithm** (GLA) of (Boersma and Hayes, 2001) is a generalization of Constraint Demotion that learns constraint rankings in Stochastic Optimality Theory. Since OT is a special case of Stochastic OT, the algorithm also implicitly learns OT rankings. It generalizes Constraint Demotion by being able to learn from cases of free variation. Recall from Sec. 11.3 that in Stochastic OT each constraint is associated with a **ranking value** on a continuous scale. The ranking value is defined as the mean of the Gaussian distribution that constitutes the constraint. The goal of the GLA is to assign a ranking value for each constraint. The algorithm is a simple extension to the Constraint Demotion algorithm, and follows exactly the same steps until the final step. Inside of demoting constraints to a lower strata, the ranking value of each constraint violated by the learning observation (Candidate 1) is decreased slightly, and the ranking value of each constraint violated by the learner's output (Candidate 2) is increased slightly, as shown below:

	/underlying form/	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
	Candidate 1 (learning observation)	*!→	* →			* →			
☞	Candidate 2 (learner's output)				← *		← *		

11.6 SUMMARY

This chapter has introduced many of the important concepts of phonetics and computational phonology.

- **Transducers** can be used to model phonological rules just as they were used in Ch. 3 to model spelling rules. **Two-level morphology** is a theory of morphology/phonology which models phonological rules as finite-state **well-formedness constraints** on the mapping between lexical and surface form.
- **Optimality theory** is a theory of phonological well-formedness; there are computational implementations, and relationships to transducers.
- Computational models exist for **syllabification**, inserting syllable boundaries in phone strings.
- There are numerous algorithms for learning phonological and morphological rules, both supervised and unsupervised.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

Computational phonology is a fairly recent field. The idea that phonological rules could be modeled as regular relations dates to Johnson (1972), who showed that any phonological system that didn't allow rules to apply to their own output (i.e., systems that did not have recursive rules) could be modeled with regular relations (or finite-state transducers). Virtually all phonological rules that had been formulated at the time had this property (except some rules with integral-valued features, like early stress and tone rules). Johnson's insight unfortunately did not attract the attention of the community, and was independently discovered by Ronald Kaplan and Martin Kay; see Ch. 3 for the rest of the history of two-level morphology. Karttunen (1993) gives a tutorial introduction to two-level morphology that includes more of the advanced details than we were able to present here, and the definitive text on finite-state morphology is Beesley and Karttunen (2003). Other FSA models of phonology include Bird and Ellison (1994).

Earlier computational finite-state models that deal with templatic morphology in languages like Arabic include Kataja and Koskenniemi (1988), Kornai (1991), Bird and Ellison (1994), and Beesley (1996). Extensions of the Kay (1987) model include Kiraz (1997, 2000, 2001). Recent models based on extensions to the finite-state calculus include Beesley and Karttunen (2000).

Optimality theory was developed by Prince and Smolensky and circulated as a technical report (Prince and Smolensky, 1993) until its publication more than a decade later (Prince and Smolensky, 2004). A selection from the extensive finite-state literature in OT includes Eisner (1997, 2000, 2002), Gerdemann and van Noord (2000), and Riggle (2005).

Recent work on phonological learning has focused on some new areas. One is learning **phonotactic constraints** on the allowable word-internal sequences in the language, including probabilistic (Coleman and Pierrehumbert, 1997; Frisch et al., 2000;

Bailey and Hahn, 2001; Hayes and Wilson, 2007; Albright, 2007) as well as non-probabilistic phonotactic constraints (Hayes, 2004; Prince and Tesar, 2004; Tesar and Prince, 2007). A related task is the learning of **underlying forms** and phonological alternations given the observed surface forms and the set of constraints. Many of the unsupervised algorithms for learning underlying forms are based on a constraint satisfaction approach, in which sets of possible underlying forms are proposed by examining alternating surface forms, and then iteratively ruling out possible underlying forms (Tesar and Prince, 2007; Alderete et al., 2005; Tesar, 2006a, 2006b). The recent unsupervised Maximum Likelihood Learning of Lexicons and Grammars (MLG) model of Jarosz (2006, 2008) learns underlying forms and constraint rankings given surface forms in a probabilistic version of OT using the Expectation-Maximization (EM) algorithm described in Ch. 6.

HARMONIC GRAMMAR

HARMONY

Indeed, in addition to this probabilistic model of Jarosz (2008), as well as the Stochastic OT described earlier in the chapter, much recent work in computational phonology has focused on models with weighted constraints, including **Harmonic Grammar** and **Maximum Entropy Models**. For example **Harmonic Grammar** is an extension to Optimality Theory (indeed is the theory that Optimality Theory originally grew out of) in which optimality for a form is defined as maximal **harmony**. Harmony is defined by the sum of weighted constraints (Smolensky and Legendre, 2006). In using sums of weight rather than OT-style rankings, Harmony Theory resembles the log-linear models of Ch. 6. Recent computational work include the application to OT of Maximum Entropy Models (Goldwater and Johnson, 2003) and the Harmonic Grammar related models of Pater et al. (2007) and Pater (2007).

Word segmentation is one of the earliest problems in computational linguistics, and models date back to Harris (1954). Among the many modern models are Bayesian ones like Brent (1999) and Goldwater et al. (2006). The word segmentation problem is important also in computational developmental psycholinguistics; for representative recent work see Christiansen et al. (1998), Kuhl et al. (2003), Thiessen and Saffran (2004) and Thiessen et al. (2005). Recent work on morphology induction includes Baroni et al. (2002), Clark (2002), and Albright and Hayes (2003).

Readers with further interest in phonology should consult phonology textbooks like Odden (2005) and Kager (2000).

Exercises

11.1 Build an automaton for rule (11.3).

CANADIAN RAISING

11.2 One difference between one dialect of Canadian English and most dialects of American English is called **Canadian raising**. Bromberger and Halle (1989) note that some Canadian dialects of English raise /aɪ/ to [ʌɪ] and /aʊ/ to [ʌʊ] in stressed position before a voiceless consonant. A simplified version of the rule dealing only with /aɪ/ can be stated as:

$$/\text{aɪ}/ \longrightarrow [\text{ʌɪ}] \;/\; \underline{\quad\;} \begin{bmatrix} C \\ -voice \end{bmatrix} \tag{11.10}$$

This rule has an interesting interaction with the flapping rule. In some Canadian dialects the word *rider* and *writer* are pronounced differently: *rider* is pronounced [raɪɾɚ] while *writer* is pronounced [rʌɪɾɚ]. Write a two-level rule and an automaton for both the raising rule and the flapping rule which correctly models this distinction. You may make simplifying assumptions as needed.

11.3 Write the lexical entry for the pronunciation of the English past tense (preterite) suffix *-d*, and the two level-rules that express the difference in its pronunciation depending on the previous context. Don't worry about the spelling rules. (Hint: make sure you correctly handle the pronunciation of the past tenses of the words *add*, *pat*, *bake*, and *bag*.)

11.4 Write two-level rules for the Yawelmani Yokuts phenomena of Harmony, Shortening, and Lowering introduced on page 5. Make sure your rules are capable of running in parallel.

Albright, A. (2007). How many grammars am i holding up? Discovering phonological differences between word classes. In *WCCFL 26*, pp. 34–42.

Albright, A. and Hayes, B. (2003). Rules vs. analogy in english past tenses: A computational/experimental study. *Cognition*, *90*, 119–161.

Alderete, J., Brasoveanu, A., Merchant, N., Prince, A., and Tesar, B. (2005). Contrast analysis aids in the learning of phonological underlying forms. In *WCCFL 24*, pp. 34–42.

Antworth, E. L. (1990). *PC-KIMMO: A Two-level Processor for Morphological Analysis*. Summer Institute of Linguistics, Dallas, TX.

Archangeli, D. (1984). *Underspecification in Yawelmani Phonology and Morphology*. Ph.D. thesis, MIT.

Archangeli, D. (1997). Optimality theory: An introduction to linguistics in the 1990s. In Archangeli, D. and Langendoen, D. T. (Eds.), *Optimality Theory: An Overview*. Blackwell, Oxford.

Bailey, T. and Hahn, U. (2001). Perception of wordlikeness: Effects of segment probability and length on the processing of nonwords. *Journal of Memory and Language*, *44*, 568–591.

Baroni, M., Matiasek, J., and Trost, H. (2002). Unsupervised discovery of morphologically related words based on orthographic and semantic similarity. In *Proceedings of ACL SIGPHON*, Philadelphia, PA.

Beesley, K. R. (1996). Arabic finite-state morphological analysis and generation. In *COLING-96*, Copenhagen, pp. 89–94.

Beesley, K. R. and Karttunen, L. (2000). Finite-state non-concatenative morphotactics. In *Proceedings of ACL SIGPHON*, Luxembourg, pp. 50–59.

Beesley, K. R. and Karttunen, L. (2003). *Finite-State Morphology*. CSLI Publications, Stanford University.

Bird, S. and Ellison, T. M. (1994). One-level phonology: Autosegmental representations and rules as finite automata. *Computational Linguistics*, *20*(1).

Blevins, J. (1995). The handbook of phonological theory. In Goldsmith, J. (Ed.), *The syllable in phonological theory*. Blackwell, Oxford.

Boersma, P. and Hayes, B. (2001). Empirical tests of the gradual learning algorithm. *Linguistic Inquiry*, *32*, 45–86.

Brent, M. R. (1999). An efficient, probabilistically sound algorithm for segmentation and word discovery. *Machine Learning*, *34*(1–3), 71–105.

Bromberger, S. and Halle, M. (1989). Why phonology is different. *Linguistic Inquiry*, *20*, 51–70.

Chomsky, N. and Halle, M. (1968). *The Sound Pattern of English*. Harper and Row.

Christiansen, M. H., Allen, J., and Seidenberg, M. S. (1998). Learning to segment speech using multiple cues: A connectionist model. *Language and Cognitive Processes*, *13*(2), 221–268.

Church, K. W. (1983). *Phrase-Structure Parsing: A Method for Taking Advantage of Allophonic Constraints*. Ph.D. thesis, MIT.

Clark, A. (2002). Memory-based learning of morphology with stochastic transducers. In *ACL-02*, Philadelphia, PA, pp. 513–520.

Cole, J. S. and Kisseberth, C. W. (1995). Restricting multi-level constraint evaluation. Rutgers Optimality Archive ROA-98.

Coleman, J. and Pierrehumbert, J. B. (1997). Stochastic phonological grammars and acceptability. In *Proceedings of ACL SIGPHON*.

de Marcken, C. (1996). *Unsupervised Language Acquisition*. Ph.D. thesis, MIT.

Eisner, J. (1997). Efficient generation in primitive optimality theory. In *ACL/EACL-97*, Madrid, Spain, pp. 313–320.

Eisner, J. (2000). Directional constraint evaluation in Optimality Theory. In *COLING-00*, Saarbrücken, Germany, pp. 257–263.

Eisner, J. (2002). Comprehension and compilation in Optimality Theory. In *ACL-02*, Philadelphia, pp. 56–63.

Ellison, T. M. (1992). *The Machine Learning of Phonological Structure*. Ph.D. thesis, University of Western Australia.

Ellison, T. M. (1994). Phonological derivation in optimality theory. In *COLING-94*, Kyoto, pp. 1007–1013.

Fisher, W. (1996). tsylb2 software and documentation. http://.

Frank, R. and Satta, G. (1998). Optimality theory and the generative complexity of constraint violability. *Computational Linguistics*, *24*(2), 307–315.

Frisch, S. A., Large, N. R., and Pisoni, D. B. (2000). Perception of wordlikeness: Effects of segment probability and length on the processing of nonwords. *Journal of Memory and Language*, *42*, 481–496.

Gaussier, E. (1999). Unsupervised learning of derivational morphology from inflectional lexicons. In *ACL-99*.

Gerdemann, D. and van Noord, G. (2000). Approximation and exactness in finite state optimality theory. In *Proceedings of ACL SIGPHON*.

Gildea, D. and Jurafsky, D. (1996). Learning bias and phonological rule induction. *Computational Linguistics*, *22*(4), 497–530.

Goldsmith, J. (1976). *Autosegmental Phonology*. Ph.D. thesis, MIT.

Goldsmith, J. (1993). Harmonic phonology. In Goldsmith, J. (Ed.), *The Last Phonological Rule*, pp. 21–60. University of Chicago Press, Chicago.

Goldsmith, J. (2001). Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, *27*, 153–198.

Goldwater, S., Griffiths, T. L., and Johnson, M. (2006). Contextual dependencies in unsupervised word segmentation. In *COLING/ACL 2006*, Sydney, Australia.

Goldwater, S. and Johnson, M. (2003). Learning OT constraint rankings using a maximum entropy model. In *Stockholm Workshop on Variation within Optimality Theory*, pp. 111–120. Stockholm University Press.

Goldwater, S. and Johnson, M. (2005). Representational bias in unsupervised learning of syllable structure. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL-2005)*.

Hafer, M. A. and Weiss, S. F. (1974). Word segmentation by letter successor varieties. *Information Storage and Retrieval*, *10*(11-12), 371–385.

Hammond, M. (1997). Parsing in OT. Alternative title "Parsing syllables: Modeling OT computationally". Rutgers Optimality Archive ROA-222-1097.

Harris, Z. S. (1954). Distributional structure. *Word*, *10*, 146–162. Reprinted in J. Fodor and J. Katz, The structure of language: Readings in the philosophy of language, Prentice-hall, 1964 and in Z. S. Harris, Papers in structural and transformational linguistics, Reidel, Dordrecht, 1970, 775–794.

Harris, Z. S. (1988). *Language and Information*. Columbia University Press.

Hayes, B. and Wilson, C. (2007). A maximum entropy model of phonotactics and phonotactic learning. *Linguistic Inquiry*. To appear.

Hayes, B. (2004). Phonological acquisition in optimality theory: the early stages. In Kager, R., Pater, J., and Zonneveld, W. (Eds.), *Constraints in Phonological Acquisition*. Cambridge University Press.

Jacquemin, C. (1997). Guessing morphology from terms and corpora. In *SIGIR 1997*, Philadelphia, PA, pp. 156–165.

Jarosz, G. (2006). Richness of the base and probabilistic unsupervised learning in optimality theory. In *Proceedings of ACL SIGPHON*, New York, NY, pp. 50–59.

Jarosz, G. (2008). Restrictiveness and phonological grammar and lexicon learning. In *CLS 43*. In press.

Johnson, C. D. (1972). *Formal Aspects of Phonological Description*. Mouton, The Hague. Monographs on Linguistic Analysis No. 3.

Johnson, M. (1984). A discovery procedure for certain phonological rules. In *COLING-84*, Stanford, CA, pp. 344–347.

Kager, R. (2000). *Optimality Theory*. Cambridge University Press.

Kahn, D. (1976). *Syllable-based Generalizations in English Phonology*. Ph.D. thesis, MIT.

Kaplan, R. M. and Kay, M. (1981). Phonological rules and finite-state transducers. Paper presented at the Annual meeting of the Linguistics Society of America. New York.

Kaplan, R. M. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, *20*(3), 331–378.

Karttunen, L. (1993). Finite-state constraints. In Goldsmith, J. (Ed.), *The Last Phonological Rule*, pp. 173–194. University of Chicago Press.

Karttunen, L. (1998). The proper treatment of optimality in computational phonology. In *Proceedings of FSMNLP'98: International Workshop on Finite-State Methods in Natural Language Processing*, Bilkent University. Ankara, Turkey, pp. 1–12.

Kataja, L. and Koskenniemi, K. (1988). Finite state description of Semitic morphology. In *COLING-88*, Budapest, pp. 313–315.

Kay, M. (1987). Nonconcatenative finite-state morphology. In *EACL-87*, Copenhagen, Denmark, pp. 2–10.

Kazakov, D. (1997). Unsupervised learning of naïve morphology with genetic algorithms. In *ECML/Mlnet Workshop on Empirical Learning of Natural Language Processing Tasks*, Prague, pp. 105–111.

Kiraz, G. A. (1997). Compiling regular formalisms with rule features into finite-state automata. In *ACL/EACL-97*, Madrid, Spain, pp. 329–336.

Kiraz, G. A. (2000). Multitiered nonlinear morphology using multitape finite automata: A case study on syriac and arabic. *Computational Linguistics*, *26*(1), 77–105.

Kiraz, G. A. (2001). *Computational Nonlinear Morphology with Emphasis on Semitic Languages*. Cambridge University Press.

Kiraz, G. A. and Möbius, B. (1998). Multilingual syllabification using weighted finite-state transducers. In *Proceedings of 3rd ESCA Workshop on Speech Synthesis*, Jenolan Caves, pp. 59–64.

Kisseberth, C. W. (1969). On the abstractness of phonology: The evidence from Yawelmani. *Papers in Linguistics*, *1*, 248–282.

Kisseberth, C. W. (1970). On the functional unity of phonological rules. *Linguistic Inquiry*, *1*(3), 291–306.

Kornai, A. (1991). *Formal Phonology*. Ph.D. thesis, Stanford University, Stanford, CA†.

Koskenniemi, K. (1983). Two-level morphology: A general computational model of word-form recognition and production. Tech. rep. Publication No. 11, Department of General Linguistics, University of Helsinki.

Kuhl, P. K., F.-M., T., and Liu, H.-M. (2003). Foreign-language experience in infancy: Effects of short-term exposure and social interaction on phonetic learning. *Proceedings of the National Academy of Sciences*, *100*, 9096–9101.

Ladefoged, P. (1993). *A Course in Phonetics*. Harcourt Brace Jovanovich. Third Edition.

Lakoff, G. (1993). Cognitive phonology. In Goldsmith, J. (Ed.), *The Last Phonological Rule*, pp. 117–145. University of Chicago Press, Chicago.

McCarthy, J. J. (1981). A prosodic theory of non-concatenative morphology. *Linguistic Inquiry*, *12*, 373–418.

Müller, K. (2001). Automatic detection of syllable boundaries combining the advantages of treebank and bracketed corpora training. In *ACL-01*, Toulouse, France. ACL.

Müller, K. (2002). Probabilistic context-free grammars for phonology. In *Proceedings of ACL SIGPHON*, Philadelphia, PA, pp. 70–80.

Müller, K., Möbius, B., and Prescher, D. (2000). Inducing probabilistic syllable classes using multivariate clustering. In *ACL-00*, pp. 225–232.

Newman, S. (1944). *Yokuts Language of California*. Viking Fund Publications in Anthropology 2, New York.

Odden, D. (2005). *Introducing Phonology*. Cambridge University Press.

Oncina, J., García, P., and Vidal, E. (1993). Learning subsequential transducers for pattern recognition tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *15*, 448–458.

Pater, J. (2007). Gradual learning and convergence. *Linguistic Inquiry*. In press.

Pater, J., Potts, C., and Bhatt, R. (2007). Harmonic grammar with linear programming. unpublished manuscript.

Pereira, F. C. N., Riley, M. D., and Sproat, R. (1994). Weighted rational transductions and their applications to human language processing. In *ARPA Human Language Technology Workshop*, Plainsboro, NJ, pp. 262–267. Morgan Kaufmann.

Prince, A. and Smolensky, P. (1993). Optimality theory: Constraint interaction in generative grammar.. Appeared as Tech. rep. CU-CS-696-93, Department of Computer Science, University of Colorado at Boulder, and TEch. rep. TR-2, Rutgers Center for Cognitive Science, Rutgers University, New Brunswick, NJ, April 1993.

Prince, A. and Smolensky, P. (2004). *Optimality Theory: Constraint interaction in generative grammar*. Blackwell.

Prince, A. and Tesar, B. (2004). Learning phonotactic distributions. In Kager, R., Pater, J., and Zonneveld, W. (Eds.), *Constraints in Phonological Acquisition*, pp. 245–291. Cambridge University Press.

Riggle, J. (2005). Contenders and learning. In *WCCFL 23*, pp. 101–114.

Saffran, J. R., Newport, E. L., and Aslin, R. N. (1996a). Statistical learning by 8-month old infants. *Science*, *274*, 1926–1928.

Saffran, J. R., Newport, E. L., and Aslin, R. N. (1996b). Word segmentation: The role of distributional cues. *Journal of Memory and Language*, *35*, 606–621.

Schone, P. and Jurafsky, D. (2000). Knowlege-free induction of morphology using latent semantic analysis. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL-2000)*.

Schone, P. and Jurafsky, D. (2001). Knowledge-free induction of inflectional morphologies. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-2001)*.

Seneff, S., Lau, R., and Meng, H. (1996). ANGIE: A new framework for speech analysis based on morpho-phonological modelling. In *ICSLP-96*.

Smolensky, P. and Legendre, G. (2006). *The Harmonic Mind*. MIT Press.

Sproat, R. (1993). *Morphology and Computation*. MIT Press.

Tesar, B. (2006a). Faithful contrastive features in learning. *Cognitive Science*, *30*(5), 863–903.

Tesar, B. (2006b). Learning from paradigmatic information..

Tesar, B. and Prince, A. (2007). Using phonotactics to learn phonological alternations. In *CLS 39*, pp. 200–213.

Tesar, B. and Smolensky, P. (2000). *Learning in Optimality Theory*. MIT Press.

Thiessen, E. D., Hill, E. A., and Saffran, J. R. (2005). Infant-directed speech facilitates word segmentation. *Infancy*, *7*, 53–71.

Thiessen, E. D. and Saffran, J. R. (2004). Spectral tilt as a cue to word segmentation in infancy and adulthood. *Perception and Psychophysics*, *66*(2), 779–791.

Titone, D. and Connine, C. M. (1997). Syllabification strategies in spoken word processing: Evidence from phonological priming. *Psychological Research*, *60*(4), 251–263.

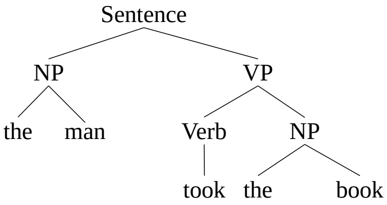
Touretzky, D. S., Elvgren III, G., and Wheeler, D. W. (1990). Phonological rule induction: An architectural solution. In *COGSCI-90*, pp. 348–355.

Treiman, R., Bowey, J., and Bourassa, D. (2002). Segmentation of spoken words into syllables by english-speaking children as compared to adults. *Journal of Experimental Child Psychology*, *83*, 213–238.

van den Bosch, A. (1997). *Learning to Pronounce Written Words: A Study in Inductive Language Learning*. Ph.D. thesis, University of Maastricht, Maastricht, The Netherlands.

Yarowsky, D. and Wicentowski, R. (2000). Minimally supervised morphological analysis by multimodal alignment. In *ACL-00*, Hong Kong, pp. 207–216.

12 FORMAL GRAMMARS OF ENGLISH



The first context-free grammar parse tree (Chomsky, 1956)

If on a winter's night a traveler by Italo Calvino
Nuclear and Radiochemistry by Gerhart Friedlander et al.
The Fire Next Time by James Baldwin
A Tad Overweight, but Violet Eyes to Die For by G. B. Trudeau
Sometimes a Great Notion by Ken Kesey
Dancer from the Dance by Andrew Holleran

Six books in English whose titles are not constituents, from Pullum (1991, p. 195)

The study of grammar has an ancient pedigree; Panini's grammar of Sanskrit was written over two thousand years ago, and is still referenced today in teaching Sanskrit. By contrast, Geoff Pullum noted in a recent talk that "almost everything most educated Americans believe about English grammar is wrong". In this chapter we make a preliminary stab at addressing some of these gaps in our knowledge of grammar and syntax, as well as introducing some of the formal mechanisms that are available for capturing this knowledge.

SYNTAX

The word **syntax** comes from the Greek *sýntaxis*, meaning "setting out together or arrangement", and refers to the way words are arranged together. We have seen various syntactic notions in previous chapters. The regular languages introduced in Ch. 2 offered a simple way to represent the ordering of strings of words, and Ch. 4 showed how to compute probabilities for these word sequences. Ch. 5 showed that part-of-speech categories could act a kind of equivalence class for words. This chapter and the following ones introduce sophisticated notions of syntax and grammar that go well beyond these simpler notions. In this chapter, we introduce three main new ideas: **constituency**, **grammatical relations**, and **subcategorization and dependency**.

The fundamental idea of constituency is that groups of words may behave as a single unit or phrase, called a constituent. For example we will see that a group of words called a **noun phrase** often acts as a unit; noun phrases include single words like *she* or *Michael* and phrases like *the house*, *Russian Hill*, and *a well-weathered three-story structure*. This chapter will introduce the use of **context-free grammars**, a formalism that will allow us to model these constituency facts.

Grammatical relations are a formalization of ideas from traditional grammar such as SUBJECTS and OBJECTS, and other related notions. In the following sentence the noun phrase *She* is the SUBJECT and *a mammoth breakfast* is the OBJECT:

(12.1) She ate a mammoth breakfast.

Subcategorization and **dependency relations** refer to certain kinds of relations between words and phrases. For example the verb *want* can be followed by an infinitive, as in *I want to fly to Detroit*, or a noun phrase, as in *I want a flight to Detroit*. But the verb *find* cannot be followed by an infinitive (**I found to fly to Dallas*). These are called facts about the *subcategorization* of the verb.

As we'll see, none of the syntactic mechanisms that we've discussed up until now can easily capture such phenomena. They can be modeled much more naturally by grammars that are based on context-free grammars. Context-free grammars are thus the backbone of many formal models of the syntax of natural language (and, for that matter, of computer languages). As such they are integral to many computational applications including grammar checking, semantic interpretation, dialogue understanding and machine translation. They are powerful enough to express sophisticated relations among the words in a sentence, yet computationally tractable enough that efficient algorithms exist for parsing sentences with them (as we will see in Ch. 13). Later in Ch. 14 we'll show that adding probability to context-free grammars gives us a model of disambiguation, and also helps model certain aspects of human parsing.

In addition to an introduction to the grammar formalism, this chapter also provides an brief overview of the grammar of English. We have chosen a domain which has relatively simple sentences, the Air Traffic Information System (ATIS) domain (Hemphill et al., 1990). ATIS systems are an early example of spoken language systems for helping book airline reservations. Users try to book flights by conversing with the system, specifying constraints like *I'd like to fly from Atlanta to Denver*. The U.S. government funded a number of different research sites to collect data and build ATIS systems in the early 1990s. The sentences we will be modeling in this chapter are drawn from the corpus of user queries to the system.

12.1 CONSTITUENCY

NOUN PHRASE

How do words group together in English? Consider the **noun phrase**, a sequence of words surrounding at least one noun. Here are some examples of noun phrases (thanks to Damon Runyon):

Harry the Horse
the Broadway coppers
they
a high-class spot such as Mindy's
the reason he comes into the Hot Box
three parties from Brooklyn

How do we know that these words group together (or "form constituents")? One piece of evidence is that they can all appear in similar syntactic environments, for example before a verb.

three parties from Brooklyn *arrive*. . .
a high-class spot such as Mindy's *attracts*. . .
the Broadway coppers *love*. . .
they *sit*

But while the whole noun phrase can occur before a verb, this is not true of each of the individual words that make up a noun phrase. The following are not grammatical sentences of English (recall that we use an asterisk (*) to mark fragments that are not grammatical English sentences):

*from *arrive*. . .   *as *attracts*. . .
*the *is*. . .   *spot *is*. . .

Thus to correctly describe facts about the ordering of these words in English, we must be able to say things like "*Noun Phrases can occur before verbs*".

PREPOSED
POSTPOSED

Other kinds of evidence for constituency come from what are called **preposed** or **postposed** constructions. For example, the prepositional phrase *on September seventeenth* can be placed in a number of different locations in the following examples, including preposed at the beginning, and postposed at the end:

On September seventeenth, I'd like to fly from Atlanta to Denver
I'd like to fly *on September seventeenth* from Atlanta to Denver
I'd like to fly from Atlanta to Denver *on September seventeenth*

But again, while the entire phrase can be placed differently, the individual words making up the phrase cannot be:

*On September, I'd like to fly seventeenth from Atlanta to Denver
*On I'd like to fly September seventeenth from Atlanta to Denver
*I'd like to fly on September from Atlanta to Denver seventeenth

Section 12.6 will give other motivations for context-free grammars based on their ability to model recursive structures. See Radford (1988) for further examples of groups of words behaving as a single constituent.

12.2 Context-Free Grammars

CFG

The most commonly used mathematical system for modeling constituent structure in English and other natural languages is the **Context-Free Grammar**, or **CFG**. Context-free grammars are also called **Phrase-Structure Grammars**, and the formalism is equivalent to what is also called **Backus-Naur Form** or **BNF**. The idea of basing a grammar on constituent structure dates back to the psychologist Wilhelm Wundt (1900), but was not formalized until Chomsky (1956) and, independently, Backus (1959).

RULES
LEXICON
NP

A context-free grammar consists of a set of **rules** or **productions**, each of which expresses the ways that symbols of the language can be grouped and ordered together, and a **lexicon** of words and symbols. For example, the following productions express that a **NP** (or **noun phrase**), can be composed of either a *ProperNoun* or a determiner (*Det*) followed by a *Nominal*; a *Nominal* can be one or more *Noun*s.

$$\begin{array}{rcl} NP & \rightarrow & Det\ Nominal \\ NP & \rightarrow & ProperNoun \\ Nominal & \rightarrow & Noun \mid Nominal\ Noun \end{array}$$

Context-free rules can be hierarchically embedded, so we can combine the previous rules with others like the following which express facts about the lexicon:

$$\begin{array}{rcl} Det & \rightarrow & a \\ Det & \rightarrow & the \\ Noun & \rightarrow & flight \end{array}$$

TERMINAL
NON-TERMINAL

The symbols that are used in a CFG are divided into two classes. The symbols that correspond to words in the language ("the", "nightclub") are called **terminal** symbols; the lexicon is the set of rules that introduce these terminal symbols. The symbols that express clusters or generalizations of these are called **non-terminals**. In each context-free rule, the item to the right of the arrow (→) is an ordered list of one or more terminals and non-terminals, while to the left of the arrow is a single non-terminal symbol expressing some cluster or generalization. Notice that in the lexicon, the non-terminal associated with each word is its lexical category, or part-of-speech, which we defined in Ch. 5.

A CFG can be thought of in two ways: as a device for generating sentences, and as a device for assigning a structure to a given sentence. We saw this same dualism in our discussion of finite-state transducers in Ch. 3. As a generator, we can read the → arrow as "rewrite the symbol on the left with the string of symbols on the right".

So starting from the symbol:	*NP*,
we can use rule 12.2 to rewrite *NP* as:	*Det Nominal*
and then rule 12.2:	*Det Noun*
and finally via rules 12.2 and 12.2 as:	*a flight*

DERIVED
DERIVATION
PARSE TREE

We say the string *a flight* can be **derived** from the non-terminal *NP*. Thus a CFG can be used to generate a set of strings. This sequence of rule expansions is called a **derivation** of the string of words. It is common to represent a derivation by a **parse tree** (commonly shown inverted with the root at the top). Fig. 12.1 shows the tree representation of this derivation.

IMMEDIATELY DOMINATES
DOMINATES

In the parse tree shown in Fig. 12.1 we say that the node *NP* **immediately dominates** the node *Det* and the node *Nom*. We say that the node *NP* **dominates** all the nodes in the tree (*Det*, *Nom*, *Noun*, *a*, *flight*).

START SYMBOL

The formal language defined by a CFG is the set of strings that are derivable from the designated **start symbol**. Each grammar must have one designated start symbol,

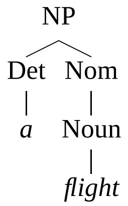


Figure 12.1 A parse tree for “a flight”.

which is often called *S*. Since context-free grammars are often used to define sentences, *S* is usually interpreted as the “sentence” node, and the set of strings that are derivable from *S* is the set of sentences in some simplified version of English.

VERB PHRASE

Let’s add to our list of rules a few higher-level rules that expand *S*, and a couple of others. One will express the fact that a sentence can consist of a noun phrase followed by a **verb phrase**:

$$S \rightarrow NP\ \ VP \qquad \text{I prefer a morning flight}$$

A verb phrase in English consists of a verb followed by assorted other things; for example, one kind of verb phrase consists of a verb followed by a noun phrase:

$$VP \rightarrow Verb\ \ NP \qquad \text{prefer a morning flight}$$

Or the verb phrase may have a verb followed by a noun phrase and a prepositional phrase:

$$VP \rightarrow Verb\ \ NP\ \ PP \qquad \text{leave Boston in the morning}$$

Or the verb may be followed by a prepositional phrase alone:

$$VP \rightarrow Verb\ \ PP \qquad \text{leaving on Thursday}$$

A prepositional phrase generally has a preposition followed by a noun phrase. For example, a very common type of prepositional phrase in the ATIS corpus is used to indicate location or direction:

$$PP \rightarrow Preposition\ \ NP \qquad \text{from Los Angeles}$$

The *NP* inside a *PP* need not be a location; *PPs* are often used with times and dates, and with other nouns as well; they can be arbitrarily complex. Here are ten examples from the ATIS corpus:

to Seattle
in Minneapolis
on Wednesday
in the evening
on the ninth of July

on these flights
about the ground transportation in Chicago
of the round trip flight on United Airlines
of the AP fifty seven flight
with a stopover in Nashville

Noun	→	*flights* | *breeze* | *trip* | *morning* | ...
Verb	→	*is* | *prefer* | *like* | *need* | *want* | *fly*
Adjective	→	*cheapest* | *non – stop* | *first* | *latest*
		| *other* | *direct* | ...
Pronoun	→	*me* | *I* | *you* | *it* | ...
Proper-Noun	→	*Alaska* | *Baltimore* | *Los Angeles*
		| *Chicago* | *United* | *American* | ...
Determiner	→	*the* | *a* | *an* | *this* | *these* | *that* | ...
Preposition	→	*from* | *to* | *on* | *near* | ...
Conjunction	→	*and* | *or* | *but* | ...

Figure 12.2 The lexicon for $\mathscr{L}_0$.

S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

Figure 12.3 The grammar for $\mathscr{L}_0$, with example phrases for each rule.

Fig. 12.2 gives a sample lexicon and Fig. 12.3 summarizes the grammar rules we've seen so far, which we'll call $\mathscr{L}_0$. Note that we can use the or-symbol | to indicate that a non-terminal has alternate possible expansions.

We can use this grammar to generate sentences of this "ATIS-language". We start with *S*, expand it to *NP VP*, then choose a random expansion of *NP* (let's say to *I*), and a random expansion of *VP* (let's say to *Verb NP*), and so on until we generate the string *I prefer a morning flight*. Fig. 12.4 shows a parse tree that represents a complete derivation of *I prefer a morning flight*.

BRACKETED NOTATION

It is sometimes convenient to represent a parse tree in a more compact format called **bracketed notation**, essentially the same as LISP tree representations; here is the bracketed representation of the parse tree of Fig. 12.4:

(12.2) $[_{S}\ [_{NP}\ [_{Pro}\ \text{I}]]\ [_{VP}\ [_{V}\ \text{prefer}]\ [_{NP}\ [_{Det}\ \text{a}]\ [_{Nom}\ [_{N}\ \text{morning}]\ [_{Nom}\ [_{N}\ \text{flight}]]]]]]$

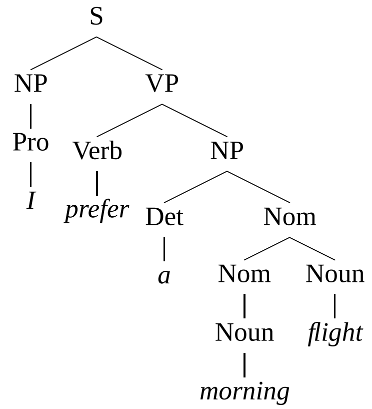


Figure 12.4 The parse tree for "I prefer a morning flight" according to grammar $\mathscr{L}_0$.

GRAMMATICAL

UNGRAMMATICAL

GENERATIVE GRAMMAR

A CFG like that of $\mathscr{L}_0$ defines a formal language. We saw in Ch. 2 that a formal language is a set of strings. Sentences (strings of words) that can be derived by a grammar are in the formal language defined by that grammar, and are called **grammatical** sentences. Sentences that cannot be derived by a given formal grammar are not in the language defined by that grammar, and are referred to as **ungrammatical**. This hard line between "in" and "out" characterizes all formal languages but is only a very simplified model of how natural languages really work. This is because determining whether a given sentence is part of a given natural language (say English) often depends on the context. In linguistics, the use of formal languages to model natural languages is called **generative grammar**, since the language is defined by the set of possible sentences "generated" by the grammar.

12.2.1 Formal definition of context-free grammar

We conclude this section by way of summary with a quick formal description of a context-free grammar and the language it generates. A context-free grammar G is defined by four parameters N, Σ, P, S (technically "is a 4-tuple"):

N a set of **non-terminal symbols** (or **variables**)

Σ a set of **terminal symbols** (disjoint from N)

R a set of **rules** or productions, each of the form $A \rightarrow \beta$, where A is a non-terminal, β is a string of symbols from the infinite set of strings $(\Sigma \cup N)*$

S a designated **start symbol**

For the remainder of the book we'll adhere to the following conventions when discussing the formal properties (as opposed to explaining particular facts about English or other languages) of context-free grammars.

Capital letters like A, B, and S	Non-terminals
S	The start symbol
Lower-case Greek letters like α, β, and γ	Strings drawn from $(\Sigma \cup N)*$
Lower-case Roman letters like u, v, and w	Strings of terminals

A language is defined via the concept of **derivation**. One string **derives** another one if it can be rewritten as the second one via some series of rule applications. More formally, following Hopcroft and Ullman (1979),

DIRECTLY DERIVES

> if $A \rightarrow \beta$ is a production of P and α and γ are any strings in the set $(\Sigma \cup N)*$, then we say that $\alpha A \gamma$ **directly derives** $\alpha\beta\gamma$, or $\alpha A \gamma \Rightarrow \alpha\beta\gamma$.

Derivation is then a generalization of direct derivation:

> Let $\alpha_1, \alpha_2, \ldots, \alpha_m$ be strings in $(\Sigma \cup N)*, m \geq 1$, such that

(12.3)
$$\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{m-1} \Rightarrow \alpha_m$$

DERIVES

> We say that α_1 **derives** α_m, or $\alpha_1 \overset{*}{\Rightarrow} \alpha_m$.

We can then formally define the language $\mathscr{L}_G$ generated by a grammar G as the set of strings composed of terminal symbols which can be derived from the designated start symbol S.

(12.4)
$$\mathscr{L}_G = \{w | w \text{ is in } \Sigma* \text{ and } S \overset{*}{\Rightarrow} w\}$$

PARSING

The problem of mapping from a string of words to its parse tree is called **parsing**; we will define algorithms for parsing in Ch. 13 and in Ch. 14.

12.3 Some Grammar Rules for English

In this section we introduce a few more aspects of the phrase structure of English; for consistency we will continue to focus on sentences from the ATIS domain. Because of space limitations, our discussion will necessarily be limited to highlights. Readers are strongly advised to consult a good reference grammar of English, such as Huddleston and Pullum (2002).

12.3.1 Sentence-Level Constructions

In the small grammar $\mathscr{L}_0$, we provided only one sentence-level construction for declarative sentences like *I prefer a morning flight*. There are a large number of constructions for English sentences, but four are particularly common and important: declarative structure, imperative structure, yes-no-question structure, and wh-question structure.

DECLARATIVE

Sentences with **declarative** structure have a subject noun phrase followed by a verb phrase, like “I prefer a morning flight”. Sentences with this structure have a great number of different uses that we will follow up on in Ch. 23. Here are a number of examples from the ATIS domain:

The flight should be eleven a.m. tomorrow
The return flight should leave at around seven p.m.
I'd like to fly the coach discount class
I want a flight from Ontario to Chicago
I plan to leave on July first around six thirty in the evening

IMPERATIVE

Sentences with **imperative** structure often begin with a verb phrase, and have no subject. They are called imperative because they are almost always used for commands and suggestions; in the ATIS domain they are commands to the system.

Show the lowest fare
Show me the cheapest fare that has lunch
Give me Sunday's flights arriving in Las Vegas from New York City
List all flights between five and seven p.m.
Show me all flights that depart before ten a.m. and have first class fares
Please list the flights from Charlotte to Long Beach arriving after lunch time
Show me the last flight to leave

We can model this sentence structure with another rule for the expansion of *S*:

$$S \rightarrow VP$$

YES-NO QUESTION

Sentences with **yes-no question** structure are often (though not always) used to ask questions (hence the name), and begin with an auxiliary verb, followed by a subject *NP*, followed by a *VP*. Here are some examples (note that the third example is not really a question but a command or suggestion; Ch. 23 will discuss the uses of these question forms to perform different **pragmatic** functions such as asking, requesting, or suggesting.)

Do any of these flights have stops?
Does American's flight eighteen twenty five serve dinner?
Can you give me the same information for United?

Here's the rule:

$$S \rightarrow Aux\ NP\ VP$$

WH-PHRASE

WH-WORD

The most complex of the sentence-level structures we will examine are the various **wh-** structures. These are so named because one of their constituents is a **wh-phrase**, that is, one that includes a **wh-word** (*who, whose, when, where, what, which, how, why*). These may be broadly grouped into two classes of sentence-level structures. The **wh-subject-question** structure is identical to the declarative structure, except that the first noun phrase contains some wh-word.

What airlines fly from Burbank to Denver?
Which flights depart Burbank after noon and arrive in Denver by six p.m?
Whose flights serve breakfast?
Which of these flights have the longest layover in Nashville?

Here is a rule. Exercise 12.10 discusses rules for the constituents that make up the *Wh-NP*.

$$S \rightarrow Wh\text{-}NP\ VP$$

WH-NON-SUBJECT QUESTION

In the **wh-non-subject question** structure, the wh-phrase is not the subject of the sentence, and so the sentence includes another subject. In these types of sentences the auxiliary appears before the subject *NP*, just as in the yes-no-question structures. Here is an example followed by a sample rule:

What flights do you have from Burbank to Tacoma Washington?

$$S \rightarrow \textit{Wh-NP Aux NP VP}$$

LONG-DISTANCE DEPENDENCIES

Constructions like the **wh-non-subject-question** contain what are called **long-distance dependencies** because the *Wh-NP what flights* is far away from the predicate that it is semantically related to, the main verb *have* in the *VP*. In some models of parsing and understanding compatible with the grammar rule above, long-distance dependencies like the relation between *flights* and *have* are thought of as a semantic relation. In such models, the job of figuring out that *flights* is the argument of *have* is done during semantic interpretation. In other models of parsing, the relationship between *flights* and *have* is considered to be a syntactic relation, and the grammar is modified to insert a small marker called a **trace** or **empty category** after the verb. We'll return to such empty-category models when we introduce the Penn Treebank on page 21.

There are other sentence-level structures we won't try to model here, like **topicalization** or other fronting constructions. In topicalization (also treated as a long-distance dependency in the Penn Treebank), a phrase is placed at the beginning of the sentence for discourse purposes.

On Tuesday, I'd like to fly from Detroit to Saint Petersburg

12.3.2 Clauses and Sentences

Before we move on, we should clarify the status of the *S* rules in the grammars we just described. *S* rules are intended to account for entire sentences that stand alone as fundamental units of discourse. However, as we'll see, *S* can also occur on the right-hand side of grammar rules and hence can be embedded within larger sentences. Clearly then there's more to being an *S* then just standing alone as a unit of discourse.

CLAUSE

What differentiates sentence constructions (i.e., the *S* rules) from the rest of the grammar is the notion that they are in some sense *complete*. In this way they correspond to the notion of a **clause** in traditional grammars, which are often described as forming a complete thought. One way of making this notion of 'complete thought' more precise is to say an *S* is a node of the parse tree below which the main verb of the *S* has all of its **arguments**. We'll define verbal arguments later, but for now let's just see an illustration from the tree for *I prefer a morning flight* in Fig. 12.4. The verb *prefer* has two arguments: the subject *I* and the object *a morning flight*. One of the arguments appears below the *VP* node, but the other one, the subject *NP*, appears only below the *S* node.

12.3.3 The Noun Phrase

Our $\mathscr{L}_0$ grammar introduced three of the most frequent types of noun phrases that occur in English: pronouns, proper-nouns and the $NP \rightarrow Det\ Nominal$ construction.

While pronouns and proper-nouns can be complex in their own ways, the central focus of this section is on the last type since that is where the bulk of the syntactic complexity resides. We can view these noun phrases consisting of a head, the central noun in the noun phrase, along with various modifiers that can occur before or after the head noun. Let's take a close look at the various parts.

The Determiner

Noun phrases can begin with simple lexical determiners, as in the following examples:

a stop	the flights	this flight
those flights	any flights	some flights

The role of the determiner in English noun phrases can also be filled by more complex expressions, as follows:

United's flight
United's pilot's union
Denver's mayor's mother's canceled flight

In these examples, the role of the determiner is filled by a possessive expression consisting of a noun phrase followed by an *'s* as a possessive marker, as in the following rule.

$$Det \rightarrow NP\ {}'s$$

The fact that this rule is recursive (since an *NP* can start with a *Det*), will help us model the latter two examples above, where a sequence of possessive expressions serves as a determiner.

There are also circumstances under which determiners are optional in English. For example, determiners may be omitted if the noun they modify is plural:

(12.5) Show me *flights* from San Francisco to Denver on weekdays

As we saw in Ch. 5, **mass nouns** also don't require determination. Recall that mass nouns often (not always) involve something that is treated like a substance (including e.g., *water* and *snow*), don't take the indefinite article "*a*", and don't tend to pluralize. Many abstract nouns are mass nouns (*music*, *homework*). Mass nouns in the ATIS domain include *breakfast*, *lunch*, and *dinner*:

(12.6) Does this flight serve dinner?

Exercise 12.4 asks the reader to represent this fact in the CFG formalism.

The Nominal

The nominal construction follows the determiner and contains any pre- and post-head noun modifiers. As indicated in grammar $\mathscr{L}_0$, in its simplest form a nominal can consist of a single noun.

$$Nominal \rightarrow Noun$$

As we'll see, this rule also provides the basis for the bottom of various recursive rules used to capture more complex nominal constructions.

Before the Head Noun

CARDINAL NUMBERS
ORDINAL NUMBERS
QUANTIFIERS

A number of different kinds of word classes can appear before the the head noun (the "postdeterminers") in a nominal. These include **cardinal numbers**, **ordinal numbers**, and **quantifiers**. Examples of cardinal numbers:

two friends   one stop

Ordinal numbers include *first*, *second*, *third*, and so on, but also words like *next*, *last*, *past*, *other*, and *another*:

the first one   the next day   the second leg
the last flight   the other American flight

Some quantifiers (*many*, *(a) few*, *several*) occur only with plural count nouns:

many fares

The quantifiers *much* and *a little* occur only with noncount nouns.
Adjectives occur after quantifiers but before nouns.

a *first-class* fare   a *nonstop* flight
the *longest* layover   the *earliest* lunch flight

ADJECTIVE PHRASE
AP

Adjectives can also be grouped into a phrase called an **adjective phrase** or **AP**. APs can have an adverb before the adjective (see Ch. 5 for definitions of adjectives and adverbs):

the *least expensive* fare

We can combine all the options for prenominal modifiers with one rule as follows:

$$NP \rightarrow (Det)\ (Card)\ (Ord)\ (Quant)\ (AP)\ Nominal$$

This simplified noun phrase rule has a flatter structure and hence is simpler than would be assumed by most modern generative theories of grammar; as we will see in Sec. 12.4, flat structures are often used for simplicity in computational applications (and indeed, there is no universally agreed-upon internal constituency for the noun phrase).

Note the use of parentheses "( )" to mark **optional constituents**. A rule with one set of parentheses is really a shorthand for two rules, one with the parentheses, one without.

After the Head Noun

A head noun can be followed by **postmodifiers**. Three kinds of nominal postmodifiers are very common in English:

prepositional phrases	all flights *from Cleveland*
non-finite clauses	any flights *arriving after eleven a.m.*
relative clauses	a flight *that serves breakfast*

Prepositional phrase postmodifiers are particularly common in the ATIS corpus, since they are used to mark the origin and destination of flights. Here are some examples, with brackets inserted to show the boundaries of each PP; note that more than one PP can be strung together:

any stopovers *[for Delta seven fifty one]*
all flights *[from Cleveland] [to Newark]*
arrival *[in San Jose] [before seven p.m.]*
a reservation *[on flight six oh six] [from Tampa] [to Montreal]*

Here's a new nominal rule to account for postnominal *PP*s:

$$Nominal \;\rightarrow\; Nominal\;\; PP$$

NON-FINITE

The three most common kinds of **non-finite** postmodifiers are the gerundive (*-ing*), *-ed*, and infinitive forms.

GERUNDIVE

Gerundive postmodifiers are so-called because they consist of a verb phrase that begins with the gerundive (*-ing*) form of the verb. In the following examples, the verb phrases happen to all have only prepositional phrases after the verb, but in general this verb phrase can have anything in it (anything, that is, which is semantically and syntactically compatible with the gerund verb).

any of those *[leaving on Thursday]*
any flights *[arriving after eleven a.m.]*
flights *[arriving within thirty minutes of each other]*

We can define the *Nominals* with gerundive modifiers as follows, making use of a new non-terminal *GerundVP*:

$$Nominal \;\rightarrow\; Nominal\; GerundVP$$

We can make rules for *GerundVP* constituents by duplicating all of our VP productions, substituting *GerundV* for *V*.

$$\begin{array}{lcl} GerundVP & \rightarrow & GerundV\; NP \\ & | & GerundV\; PP \mid GerundV \mid GerundV\; NP\; PP \end{array}$$

GerundV can then be defined as:

$$GerundV \;\rightarrow\; being \mid arriving \mid leaving \mid \ldots$$

The phrases in italics below are examples of the two other common kinds of non-finite clauses, infinitives and *-ed* forms:

the last flight *to arrive in Boston*
I need to have dinner *served*
Which is the aircraft *used by this flight*?

RELATIVE PRONOUN

A postnominal relative clause (more correctly a **restrictive relative clause**), is a clause that often begins with a **relative pronoun** (*that* and *who* are the most common). The relative pronoun functions as the subject of the embedded verb (is a **subject relative**) in the following examples:

a flight *that serves breakfast*
flights *that leave in the morning*
the United flight *that arrives in San Jose around ten p.m.*
the one *that leaves at ten thirty five*

We might add rules like the following to deal with these:

$$\begin{aligned} \mathit{Nominal} &\rightarrow \mathit{Nominal\ RelClause} \\ \mathit{RelClause} &\rightarrow (\mathit{who} \mid \mathit{that})\ VP \end{aligned}$$

The relative pronoun may also function as the object of the embedded verb, as in the following example; we leave as an exercise for the reader writing grammar rules for more complex relative clauses of this kind.

the earliest American Airlines flight that I can get

Various postnominal modifiers can be combined, as the following examples show:

a flight *[from Phoenix to Detroit] [leaving Monday evening]*
I need a flight *[to Seattle] [leaving from Baltimore] [making a stop in Minneapolis]*
evening flights *[from Nashville to Houston] [that serve dinner]*
a friend *[living in Denver] [that would like to visit me here in Washington DC]*

Before the Noun Phrase

PREDETERMINERS

Word classes that modify and appear before *NP*s are called **predeterminers**. Many of these have to do with number or amount; a common predeterminer is *all*:

all the flights all flights all non-stop flights

The example noun phrase given in Fig. 12.5 illustrates some of the complexity that arises when these rules are combined.

12.3.4 Agreement

In Ch. 3 we discussed English inflectional morphology. Recall that most verbs in English can appear in two forms in the present tense: the form used for third-person, singular subjects (*the flight does*), and the form used for all other kinds of subjects (*all the flights do*, *I do*). The third-person-singular (*3sg*) form usually has a final *-s* where the non-3sg form does not. Here are some examples, again using the verb *do*, with various subjects:

Do [$_{NP}$ all of these flights] offer first class service?
Do [$_{NP}$ I] get dinner on this flight?
Do [$_{NP}$ you] have a flight from Boston to Forth Worth?
Does [$_{NP}$ this flight] stop in Dallas?

Here are more examples with the verb *leave*:

What flights *leave* in the morning?
What flight *leaves* from Pittsburgh?

This agreement phenomenon occurs whenever there is a verb that has some noun acting as its subject. Note that sentences in which the subject does not agree with the verb are ungrammatical:

*[What flight] *leave* in the morning?
*Does [$_{NP}$ you] have a flight from Boston to Forth Worth?
*Do [$_{NP}$ this flight] stop in Dallas?

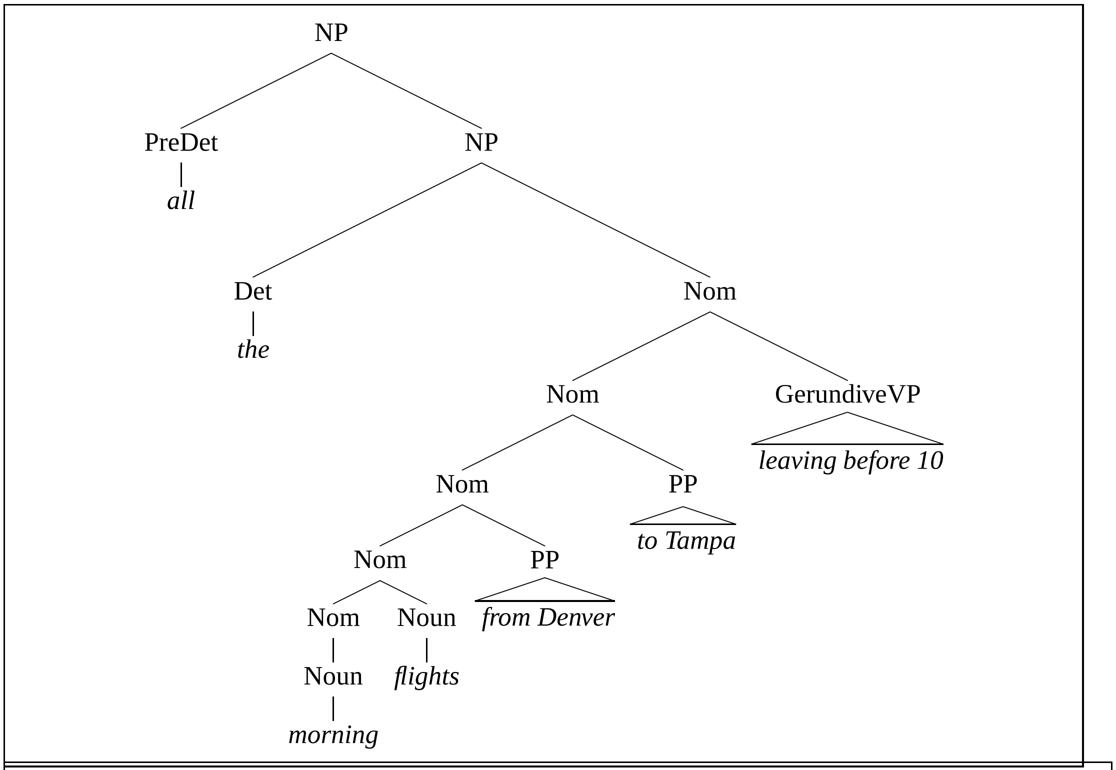


Figure 12.5 A parse tree for “all the morning flights from Denver to Tampa leaving before 10”.

How can we modify our grammar to handle these agreement phenomena? One way is to expand our grammar with multiple sets of rules, one rule set for *3sg* subjects, and one for non-*3sg* subjects. For example, the rule that handled these yes-no-questions used to look like this:

$$S \rightarrow Aux\ NP\ VP$$

We could replace this with two rules of the following form:

$$\begin{aligned} S &\rightarrow 3sgAux\ 3sgNP\ VP \\ S &\rightarrow Non3sgAux\ Non3sgNP\ VP \end{aligned}$$

We could then add rules for the lexicon like these:

$$\begin{aligned} 3sgAux &\rightarrow does \mid has \mid can \mid \ldots \\ Non3sgAux &\rightarrow do \mid have \mid can \mid \ldots \end{aligned}$$

But we would also need to add rules for *3sgNP* and *Non3sgNP*, again by making two copies of each rule for *NP*. While pronouns can be first, second, or third person, full lexical noun phrases can only be third person, so for them we just need to distinguish between singular and plural (dealing with the first and second person pronouns is left as an exercise):

$$\begin{array}{rcl}
\mathit{3SgNP} & \rightarrow & \mathit{Det\ SgNominal} \\
\mathit{Non3SgNP} & \rightarrow & \mathit{Det\ PlNominal} \\
\mathit{SgNominal} & \rightarrow & \mathit{SgNoun} \\
\mathit{PlNominal} & \rightarrow & \mathit{PlNoun} \\
\mathit{SgNoun} & \rightarrow & \mathit{flight} \mid \mathit{fare} \mid \mathit{dollar} \mid \mathit{reservation} \mid \ldots \\
\mathit{PlNoun} & \rightarrow & \mathit{flights} \mid \mathit{fares} \mid \mathit{dollars} \mid \mathit{reservations} \mid \ldots
\end{array}$$

The problem with this method of dealing with number agreement is that it doubles the size of the grammar. Every rule that refers to a noun or a verb needs to have a "singular" version and a "plural" version. Unfortunately, subject-verb agreement is only the tip of the iceberg. We'll also have to introduce copies of rules to capture the fact that head nouns and their determiners have to agree in number as well:

this flight	*this flights
those flights	*those flight

CASE
NOMINATIVE
ACCUSATIVE

Rule proliferation will also have to happen for the noun's **case**; for example English pronouns have **nominative** (*I*, *she*, *he*, *they*) and **accusative** (*me*, *her*, *him*, *them*) versions. We will need new versions of every *NP* and *N* rule for each of these.

GENDER AGREEMENT

These problems are compounded in languages like German or French, which not only have number-agreement as in English, but also have **gender agreement**. We mentioned briefly in Ch. 3 that the gender of a noun must agree with the gender of its modifying adjective and determiner. This adds another multiplier to the rule sets of the language.

Ch. 16 will introduce a way to deal with these agreement problems without exploding the size of the grammar, by effectively **parameterizing** each non-terminal of the grammar with **feature structures** and **unification**. But for many practical computational grammars, we simply rely on CFGs and make do with the large numbers of rules.

12.3.5 The Verb Phrase and Subcategorization

The verb phrase consists of the verb and a number of other constituents. In the simple rules we have built so far, these other constituents include *NP*s and *PP*s and combinations of the two:

$VP \rightarrow Verb$ disappear
$VP \rightarrow Verb\ NP$ prefer a morning flight
$VP \rightarrow Verb\ \ NP\ \ PP$ leave Boston in the morning
$VP \rightarrow Verb\ \ PP$ leaving on Thursday

Verb phrases can be significantly more complicated than this. Many other kinds of constituents can follow the verb, such as an entire embedded sentence. These are called **sentential complements**:

SENTENTIAL COMPLEMENT

You [$_{VP}$ [$_{V}$ said [$_{S}$ there were two flights that were the cheapest]]]
You [$_{VP}$ [$_{V}$ said [$_{S}$ you had a two hundred sixty six dollar fare]]
[$_{VP}$ [$_{V}$ Tell] [$_{NP}$ me] [$_{S}$ how to get from the airport in Philadelphia to downtown]]
I [$_{VP}$ [$_{V}$ think [$_{S}$ I would like to take the nine thirty flight]]

Here's a rule for these:

$$VP \rightarrow Verb\ S$$

Another potential constituent of the VP is another VP. This is often the case for verbs like *want*, *would like*, *try*, *intend*, *need*:

I want [$_{VP}$ to fly from Milwaukee to Orlando]
Hi, I want [$_{VP}$ to arrange three flights]
Hello, I'm trying [$_{VP}$ to find a flight that goes from Pittsburgh to Denver after two p.m.]

Recall from Ch. 5 that verbs can also be followed by *particles*, words that resemble a preposition but that combine with the verb to form a *phrasal verb* like *take off*. These particles are generally considered to be an integral part of the verb in a way that other post-verbal elements are not; phrasal verbs are treated as individual verbs composed of two words.

While a verb phrase can have many possible kinds of constituents, not every verb is compatible with every verb phrase. For example, the verb *want* can either be used with an NP complement (*I want a flight* ...), or with an infinitive VP complement (*I want to fly to* ...). By contrast, a verb like *find* cannot take this sort of VP complement. (* *I found to fly to Dallas*).

TRANSITIVE
INTRANSITIVE

This idea that verbs are compatible with different kinds of complements is a very old one; traditional grammar distinguishes between **transitive** verbs like *find*, which take a direct object NP (*I found a flight*), and **intransitive** verbs like *disappear*, which do not (**I disappeared a flight*).

SUBCATEGORIZE
SUBCATEGORIZES FOR
COMPLEMENTS
SUBCATEGORIZATION FRAME

Where traditional grammars **subcategorize** verbs into these two categories (transitive and intransitive), modern grammars distinguish as many as 100 subcategories. (In fact, tagsets for many such subcategorization frames exist; see Macleod et al. (1998) for the COMLEX tagset, Sanfilippo (1993) for the ACQUILEX tagset, and further discussion in Ch. 16). We say that a verb like *find* **subcategorizes for** an *NP*, while a verb like *want* subcategorizes for either an *NP* or a non-finite *VP*. We also call these constituents the **complements** of the verb (hence our use of the term **sentential complement** above). So we say that *want* can take a *VP* complement. These possible sets of complements are called the **subcategorization frame** for the verb. Another way of talking about the relation between the verb and these other constituents is to think of the verb as a logical predicate and the constituents as logical arguments of the predicate. So we can think of such predicate-argument relations as FIND(I, A FLIGHT), or WANT(I, TO FLY). We will talk more about this view of verbs and arguments in Ch. 17 when we talk about predicate calculus representations of verb semantics.

Subcategorization frames for a set of example verbs are given in Fig. 12.6. Note that a verb can subcategorize for a particular type of verb phrase, such as a verb phrase whose verb is an infinitive (*VPto*), or a verb phrase whose verb is a bare stem (uninflected: *VPbrst*). Note also that a single verb can take different subcategorization

Frame	Verb	Example
$\emptyset$	eat, sleep	I want to eat
NP	prefer, find, leave,	Find $[_{NP}$ the flight from Pittsburgh to Boston]
NP NP	show, give	Show $[_{NP}$ me] $[_{NP}$ airlines with flights from Pittsburgh]
PP_{from} PP_{to}	fly, travel	I would like to fly $[_{PP}$ from Boston] $[_{PP}$ to Philadelphia]
NP PP_{with}	help, load,	Can you help $[_{NP}$ me] $[_{PP}$ with a flight]
VPto	prefer, want, need	I would prefer $[_{VPto}$ to go by United airlines]
VPbrst	can, would, might	I can $[_{VPbrst}$ go from Boston]
S	mean	Does this mean $[_{S}$ AA has a hub in Boston]?

Figure 12.6 Subcategorization frames for a set of example verbs.

frames. The verb *find*, for example, can take an *NP NP* frame (*find me a flight*) as well as an *NP* frame.

How can we represent the relation between verbs and their complements in a context-free grammar? One thing we could do is to do what we did with agreement features: make separate subtypes of the class Verb (*Verb-with-NP-complement*, *Verb-with-Inf-VP-complement*, *Verb-with-S-complement*, and so on):

$$\begin{array}{rcl} \textit{Verb-with-NP-complement} & \rightarrow & \textit{find} \mid \textit{leave} \mid \textit{repeat} \mid \ldots \\ \textit{Verb-with-S-complement} & \rightarrow & \textit{think} \mid \textit{believe} \mid \textit{say} \mid \ldots \\ \textit{Verb-with-Inf-VP-complement} & \rightarrow & \textit{want} \mid \textit{try} \mid \textit{need} \mid \ldots \end{array}$$

Then each *VP* rule could be modified to require the appropriate verb subtype:

$$\begin{array}{lcl} VP & \rightarrow & \textit{Verb-with-no-complement} \quad \text{disappear} \\ VP & \rightarrow & \textit{Verb-with-NP-comp NP} \quad \text{prefer a morning flight} \\ VP & \rightarrow & \textit{Verb-with-S-comp S} \quad \text{said there were two flights} \end{array}$$

The problem with this approach, as with the same solution to the agreement feature problem, is a vast explosion in the number of rules. The standard solution to both of these problems is the **feature structure**, which will be introduced in Ch. 16 where we will also discuss the fact that nouns, adjectives, and prepositions can subcategorize for complements just as verbs can.

12.3.6 Auxiliaries

AUXILIARIES MODAL PERFECT PROGRESSIVE PASSIVE

The subclass of verbs called **auxiliaries** or **helping verbs** have particular syntactic constraints which can be viewed as a kind of subcategorization. Auxiliaries include the **modal** verbs *can, could, may, might, must, will, would, shall*, and *should*, the **perfect** auxiliary *have*, the **progressive** auxiliary *be*, and the **passive** auxiliary *be*. Each of these verbs places a constraint on the form of the following verb, and each of these must also combine in a particular order.

Modal verbs subcategorize for a *VP* whose head verb is a bare stem; for example, *can go in the morning, will try to find a flight*. The perfect verb *have* subcategorizes for a *VP* whose head verb is the past participle form: *have booked 3 flights*. The progressive verb *be* subcategorizes for a *VP* whose head verb is the gerundive participle: *am going from Atlanta*. The passive verb *be* subcategorizes for a *VP* whose head verb is the past participle: *was delayed by inclement weather*.

A sentence can have multiple auxiliary verbs, but they must occur in a particular order: *modal* $<$ *perfect* $<$ *progressive* $<$ *passive*. Here are some examples of multiple auxiliaries:

modal perfect	*could have been* a contender
modal passive	*will be* married
perfect progressive	*have been* feasting
modal perfect passive	*might have been* prevented

SYSTEMIC GRAMMAR
VERB GROUP

Auxiliaries are often treated just like verbs such as *want*, *seem*, or *intend*, which subcategorize for particular kinds of *VP* complements. Thus *can* would be listed in the lexicon as a *verb-with-bare-stem-VP-complement*. One way of capturing the ordering constraints among auxiliaries, commonly used in the **systemic grammar** of Halliday (1985), is to introduce a special constituent called the **verb group**, whose subconstituents include all the auxiliaries as well as the main verb. Some of the ordering constraints can also be captured in a different way. Since modals, for example, do not have a progressive or participle form, they simply will never be allowed to follow progressive or passive *be* or perfect *have*. Exercise 12.8 asks the reader to write grammar rules for auxiliaries.

ACTIVE

The passive construction has a number of properties that make it different than other auxiliaries. One important difference is a semantic one; while the subject of non-passive (**active**) sentence is often the semantic agent of the event described by the verb (*I prevented a catastrophe*) the subject of the passive is often the undergoer or patient of the event (*a catastrophe was prevented*). This will be discussed further in Ch. 18.

12.3.7 Coordination

CONJUNCTIONS
COORDINATE

The major phrase types discussed here can be **conjoined** with **conjunctions** like *and*, *or*, and *but* to form larger constructions of the same type. For example a **coordinate** noun phrase can consist of two other noun phrases separated by a conjunction:

Please repeat [$_{NP}$ [$_{NP}$ the flights] *and* [$_{NP}$ the costs]]
I need to know [$_{NP}$ [$_{NP}$ the aircraft] *and* [$_{NP}$ the flight number]]

Here's a rule that allows these structures:

$$NP \;\rightarrow\; NP \textit{ and } NP$$

Note that the ability to form coordinate phrases via conjunctions is often used as a test for constituency. Consider the following examples which differ from the ones given above in that they lack the second determiner.

Please repeat the [$_{Nom}$ [$_{Nom}$ flights] *and* [$_{Nom}$ costs]]
I need to know the [$_{Nom}$ [$_{Nom}$ aircraft] *and* [$_{Nom}$ flight number]]

The fact that these phrases can be conjoined is evidence for the presence of the underlying *Nominal* constituent we have been making use of. Here's a new rule for this:

$$Nominal \rightarrow Nominal\ and\ Nominal$$

The following examples illustrate conjunctions involving *VP*s and *S*s.

> What flights do you have [$_{VP}$ [$_{VP}$ leaving Denver] *and* [$_{VP}$ arriving in San Francisco]]
> [$_{S}$ [$_{S}$ I'm interested in a flight from Dallas to Washington] *and* [$_{S}$ I'm also interested in going to Baltimore]]

The rules for *VP* and *S* conjunctions mirror the *NP* one given above.

$$\begin{aligned} VP &\rightarrow VP\ and\ VP \\ S &\rightarrow S\ and\ S \end{aligned}$$

Since all the major phrase types can be conjoined in this fashion it is also possible to represent this conjunction fact more generally; a number of grammar formalisms such as (Gazdar et al., 1985) do this via **metarules** such as the following:

METARULES

$$X \rightarrow X\ and\ X$$

This metarule simply states that any non-terminal can be conjoined with the same non-terminal to yield a constituent of the same type. Of course, the variable *X* must be designated as a variable that stands for any non-terminal rather than a non-terminal itself.

12.4 TREEBANKS

Context-free grammar rules of the type that we have explored so far in this chapter can be used, in principle, to assign a parse tree to any sentence. This means that it is possible to build a corpus in which every sentence is syntactically annotated with a parse tree. Such a syntactically annotated corpus is called a **treebank**. Treebanks play an important roles in parsing, as we will see in Ch. 13, and in various empirical investigations of syntactic phenomena.

TREEBANK

A wide variety of treebanks have been created, generally by using parsers (of the sort described in the next two chapters) to automatically parse each sentence, and then using humans (linguists) to hand-correct the parses. The **Penn Treebank** project (whose POS tagset we introduced in Ch. 5) has produced treebanks from the Brown, Switchboard, ATIS, and Wall Street Journal corpora of English, as well as treebanks in Arabic and Chinese. Other treebanks include the Prague Dependency Treebank for Czech, the Negra treebank for German, and the Susanne treebank for English.

PENN TREEBANK

12.4.1 Example: The Penn Treebank Project

Fig. 12.7 shows sentences from the Brown and ATIS portions of the Penn Treebank.[1] Note the formatting differences for the part-of-speech tags; such small differences are common and must be dealt with in processing treebanks. The Penn Treebank part-of-speech tagset was defined in Ch. 5. The use of LISP-style parenthesized notation for trees is extremely common, and resembles the bracketed notation we saw above in (12.2). For those who are not familiar with it we show a standard node-and-line tree representation in Fig. 12.8.

```
((S
   (NP-SBJ (DT That)
     (JJ cold) (, ,)
     (JJ empty) (NN sky) )
   (VP (VBD was)
     (ADJP-PRD (JJ full)
       (PP (IN of)
         (NP (NN fire)
           (CC and)
           (NN light) ))))
   (. .) ))
```

(a)

```
((S
   (NP-SBJ The/DT flight/NN )
   (VP should/MD
     (VP arrive/VB
       (PP-TMP at/IN
         (NP eleven/CD a.m/RB ))
       (NP-TMP tomorrow/NN )))))
```

(b)

Figure 12.7 Parsed sentences from the LDC Treebank3 version of the Brown (a) and ATIS (b) corpora.

TRACES
LONG-DISTANCE DEPENDENCIES
SYNTACTIC MOVEMENT

Fig. 12.9 shows a tree from the Wall Street Journal. This tree shows another feature of the Penn Treebanks: the use of **traces** (`-NONE-` nodes) to mark **long-distance dependencies** or **syntactic movement**. For example, quotations often follow a quotative verb like *say*. But in this example the quotation “We would have to wait until we have collected on those assets” precedes the words *he said*. An empty S containing only the node `-NONE-` is used to mark the position after *said* where the quotation sentence often occurs. This empty node is marked (in Treebanks II and III) with the index 2, as is the quotation S at the beginning of the sentence. Such coindexing may make it easier for some parsers to recover the fact that this fronted or topicalized quotation is the complement of the verb *said*. A similar `-NONE-` node is used mark the fact that there is no syntactic subject right before the verb *to wait*; instead, the subject is the earlier NP *We*. Again, they are both coindexed with the index 1.

The Penn Treebank II and Treebank III releases added further information to make it easier to recover the relationships between predicates and arguments. Certain phrases were marked with tags indicating the grammatical function of the phrase (as surface subject, logical topic, cleft, non-VP predicates) whether it appeared in particular text

[1] The Penn Treebank project released treebanks in multiple languages and in various stages; for example there were Treebank I (Marcus et al., 1993), Treebank II (Marcus et al., 1994), and Treebank III releases of English treebanks. We will use Treebank III for our examples.

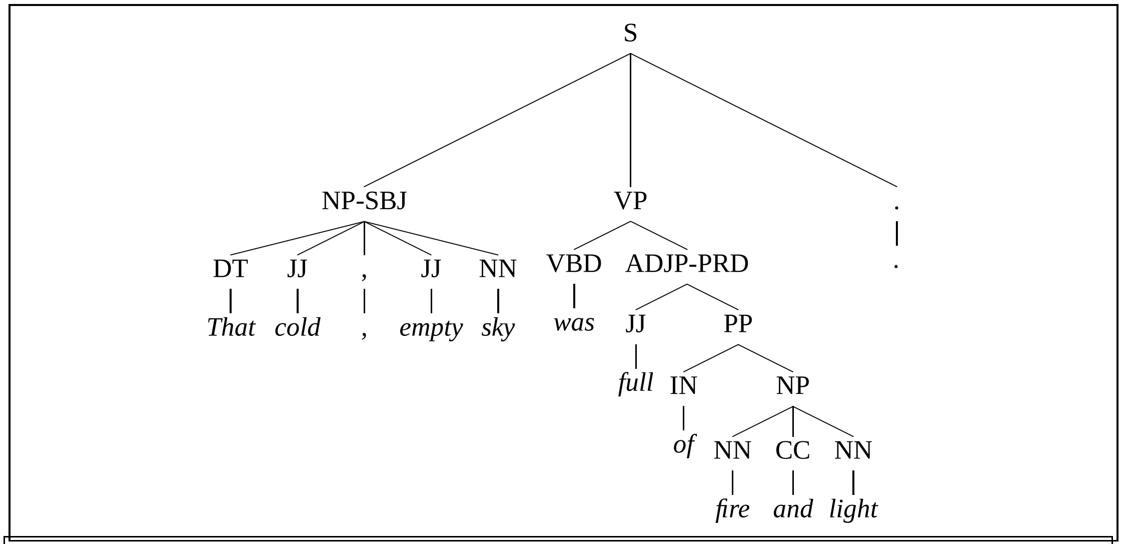


Figure 12.8 The tree corresponding to the Brown corpus sentence in the previous figure.

```
( (S (‘‘ ‘‘)
    (S-TPC-2
      (NP-SBJ-1 (PRP We) )
      (VP (MD would)
        (VP (VB have)
          (S
            (NP-SBJ (-NONE- *-1) )
            (VP (TO to)
              (VP (VB wait)
                (SBAR-TMP (IN until)
                  (S
                    (NP-SBJ (PRP we) )
                    (VP (VBP have)
                      (VP (VBN collected)
                        (PP-CLR (IN on)
                          (NP (DT those) (NNS assets) )))))))))))))
    (, ,) (’’ ’’)
    (NP-SBJ (PRP he) )
    (VP (VBD said)
      (S (-NONE- *T*-2) ))
    (. .) ))
```

Figure 12.9 A sentence from the Wall Street Journal portion of the LDC Penn Treebank. Note the use of the empty `-NONE-` nodes.

S	→	*NP VP .*	*PRP*	→	*we* | *he*
		NP VP	*DT*	→	*the* | *that* | *those*
		" S " , NP VP .	*JJ*	→	*cold* | *empty* | *full*
		-NONE-	*NN*	→	*sky* | *fire* | *light* | *flight*
		DT NN	*NNS*	→	*assets*
		DT NN NNS	*CC*	→	*and*
		NN CC NN	*IN*	→	*of* | *at* | *until* | *on*
		CD RB	*CD*	→	*eleven*
NP	→	*DT JJ , JJ NN*	*RB*	→	*a.m*
		PRP	*VB*	→	*arrive* | *have* | *wait*
		-NONE-	*VBD*	→	*said*
VP	→	*MD VP*	*VBP*	→	*have*
		VBD ADJP	*VBN*	→	*collected*
		VBD S	*MD*	→	*should* | *would*
		VB PP	*TO*	→	*to*
		VB S			
		VB SBAR			
		VBP VP			
		VBN VP			
		TO VP			
SBAR	→	*IN S*			
ADJP	→	*JJ PP*			
PP	→	*IN NP*			

Figure 12.10 A sample of the CFG grammar that would be extracted from the three treebank sentences in Fig. 12.7 and Fig. 12.9.

categories (headlines, titles), and its semantic function (temporal phrases, locations) (Marcus et al., 1994; Bies et al., 1995). Fig. 12.9 shows examples of the `-SBJ` (surface subject) and `-TMP` (temporal phrase) tags. Fig. 12.8 shows in addition the `-PRD` tag, which is used for predicates which are not VPs (the one in Fig. 12.8 is an ADJP). Fig. 12.19 shows the tag `-UNF` in `NP-UNF` meaning 'unfinished or incomplete phrase'.

12.4.2 Using a Treebank as a Grammar

The sentences in a treebank implicitly constitute a grammar of the language. For example, we can take the three parsed sentences in Fig. 12.7 and Fig. 12.9 and extract each of the CFG rules in them. For simplicity, let's strip off the rule suffixes (`-SBJ` and so on). The resulting grammar is shown in Fig. 12.10.

The grammar used to parse the Penn Treebank is relatively flat, resulting in very many and very long rules. For example among the approximately 4,500 different rules for expanding VP are separate rules for PP sequences of any length, and every possible arrangement of verb arguments:

```
VP → VBD PP
VP → VBD PP PP
VP → VBD PP PP PP
VP → VBD PP PP PP PP
VP → VB ADVP PP
VP → VB PP ADVP
VP → ADVP VB PP
```

as well as even longer rules, such a:

```
VP → VBP PP PP PP PP PP ADVP PP
```

which comes from the VP marked in italics:

(12.7) This mostly happens because we *go from football in the fall to lifting in the winter to football again in the spring*.

Some of the many thousands of NP rules include:

```
NP → DT JJ NN
NP → DT JJ NNS
NP → DT JJ NN NN
NP → DT JJ JJ NN
NP → DT JJ CD NNS
NP → RB DT JJ NN NN
NP → RB DT JJ JJ NNS
NP → DT JJ JJ NNP NNS
NP → DT NNP NNP NNP NNP JJ NN
NP → DT JJ NNP CC JJ JJ NN NNS
NP → RB DT JJS NN NN SBAR
NP → DT VBG JJ NNP NNP CC NNP
NP → DT JJ NNS , NNS CC NN NNS NN
NP → DT JJ JJ VBG NN NNP NNP FW NNP
NP → NP JJ , JJ `` SBAR '' NNS
```

The last two of those rules, for example, come from the following two NPs:

(12.8) [$_{\text{DT}}$ The] [$_{\text{JJ}}$ state-owned] [$_{\text{JJ}}$ industrial] [$_{\text{VBG}}$ holding] [$_{\text{NN}}$ company] [$_{\text{NNP}}$ Instituto] [$_{\text{NNP}}$ Nacional] [$_{\text{FW}}$ de] [$_{\text{NNP}}$ Industria]

(12.9) [$_{\text{NP}}$ Shearson's] [$_{\text{JJ}}$ easy-to-film], [$_{\text{JJ}}$ black-and-white] "[$_{\text{SBAR}}$ Where We Stand]" [$_{\text{NNS}}$ commercials]

Viewed as a large grammar in this way, the Penn Treebank III Wall Street Journal corpus, which contains about 1 million words, also has about 1 million non-lexical rule tokens, consisting of about 17,500 distinct rule types.

Various facts about the treebank grammars, such as their large numbers of flat rules, pose problems for probabilistic parsing algorithms. For this reason, it is common to make various modifications to a grammar extracted from a treebank. We will discuss these further in Ch. 14.

12.4.3 Searching Treebanks

It is often important to search through a treebank to find examples of particular grammatical phenomena, either for linguistic research or for answering analytic questions about a computational application. But neither the regular expressions used for text search nor the boolean expressions over words used for web search are a sufficient search tool. What is needed is a language that can specify constraints about nodes and links in a parse tree, so as to search for specific patterns.

Various such tree-searching languages exist in different tools. **Tgrep** (Pito, 1993) and **TGrep2** (Rohde, 2005) are publicly-available tools for searching treebanks that use a similar language for expressing tree constraints. We'll describe the more recent language used by **TGrep2**, drawing from the online manual (Rohde, 2005).

A pattern in **tgrep** or **TGrep2** consists of a specification of a node, possibly followed by links to other nodes. A node specification can then be used to return the subtree rooted at that node. For example, the pattern

```
NP
```

returns all subtrees in a corpus whose root is NP. Nodes can be specified by a name, a regular expression inside slashes, or a disjunction of these. For example, we can specify a singular or plural noun (`NN` or `NNS`) using Penn Treebank notation as either of the following:

```
/NNS?/       NN|NNS
```

A node which either is the word *bush* or else ends in the string *tree* can be expressed as:

```
/tree$/|bush
```

The power of **tgrep/TGrep2** patterns is the ability to specify information about links. The operator `<` means **immediately dominates**; the following pattern thus matches an NP immediately dominating a PP

```
NP < PP
```

The relation `<<` is used to specify dominance; this pattern matches an NP dominating a PP:

```
NP << PP
```

This previous pattern would thus match either of the following trees:

(12.10)
```
(NP (NP (NN reinvestment))
    (PP (IN of)
        (NP (NNS dividends))))
```

(12.11)
```
(NP (NP (DT the) (JJ austere) (NN company) (NN dormitory))
    (VP (VBN run)
        (PP (IN by) (NP (DT a) (JJ prying) (NN caretaker)))))
```

The relation `.` is used to mark linear precedence. The following pattern matches an NP that immediately dominates a JJ and is immediately followed by a PP, for example matching the NP dominating *the austere company dormitory* in (12.11) above:[2]

```
NP < JJ . VP
```

Each of the relations in a **tgrep/TGrep2** expression is interpreted as referring to the first or root node. Thus for example the following expression means an NP which both precedes a PP and dominates an S:

```
NP . PP < S
```

[2] The definition of linear precedence differs slightly between **tgrep** and **TGrep2**. See Rohde (2005) for more details.

If we wanted instead to specify that the PP dominated the S, we could use parentheses as follows:

```
NP . (PP < S)
```

Fig. 12.11 gives the major link operations for **TGrep2**.

A < B	A is the parent of (immediately dominates) B.
A > B	A is the child of B.
A <N B	B is the Nth child of A (the first child is <1).
A >N B	A is the Nth child of B (the first child is >1).
A <, B	Synonymous with A <1 B.
A >, B	Synonymous with A >1 B.
A <-N B	B is the Nth-to-last child of A (the last child is <-1).
A >-N B	A is the Nth-to-last child of B (the last child is >-1).
A <- B	B is the last child of A (synonymous with A <-1 B).
A >- B	A is the last child of B (synonymous with A >-1 B).
A <‘ B	B is the last child of A (also synonymous with A <-1 B).
A >‘ B	A is the last child of B (also synonymous with A >-1 B).
A <: B	B is the only child of A
A >: B	A is the only child of B
A << B	A dominates B (A is an ancestor of B).
A >> B	A is dominated by B (A is a descendant of B).
A <<, B	B is a left-most descendant of A.
A >>, B	A is a left-most descendant of B.
A <<‘ B	B is a right-most descendant of A.
A >>‘ B	A is a right-most descendant of B.
A <<: B	There is a single path of descent from A and B is on it.
A >>: B	There is a single path of descent from B and A is on it.
A . B	A immediately precedes B.
A , B	A immediately follows B.
A .. B	A precedes B.
A ,, B	A follows B.
A $ B	A is a sister of B (and A ≠ B).
A $. B	A is a sister of and immediately precedes B.
A $, B	A is a sister of and immediately follows B.
A $.. B	A is a sister of and precedes B.
A $,, B	A is a sister of and follows B.

Figure 12.11 Links in **TGrep2**, summarized from Rohde (2005).

12.4.4 Heads and Head Finding

We suggested informally earlier that syntactic constituents could be associated with a lexical **head**; *N* is the head of an *NP*, *V* is the head of a *VP*. This idea of a head for each constituent dates back to Bloomfield (1914). It is central to such linguistic formalisms such as Head-Driven Phrase Structure Grammar (Pollard and Sag, 1994), and has become extremely popular in computational linguistics with the rise of lexicalized grammars (Ch. 14).

In one simple model of lexical heads, each context-free rule is associated with a head (Charniak, 1997; Collins, 1999). The head is the word in the phrase which is grammatically the most important. Heads are passed up the parse tree; thus each non-terminal in a parse-tree is annotated with a single word which is its lexical head. Fig. 12.12 shows an example of such a tree from Collins (1999), in which each non-terminal is annotated with its head. "Workers dumped sacks into a bin" is a shortened form of a WSJ sentence.

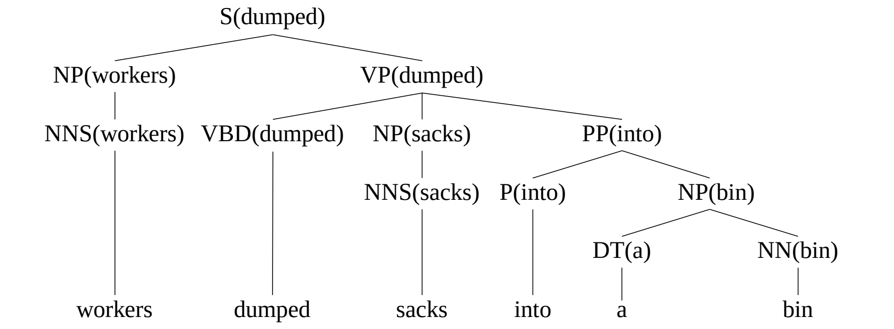


Figure 12.12 A lexicalized tree from Collins (1999).

In order to generate such a tree, each CFG rule must be augmented to identify one right-hand-side constituent to be the head daughter. The headword for a node is then set to the headword of its head daughter. Choosing these head daughters is simple for textbook examples (*NN* is the head of *NP*) but is complicated and indeed controversial for most phrases. (Should the complementizer *to* or the verb be the head of an infinite verb-phrase?) Modern linguistic theories of syntax generally include a component that defines heads (see e.g., Pollard and Sag, 1994).

An alternative approach to head-finding is used in most practical computational systems. Instead of specifying head rules in the grammar itself, heads are identified dynamically in the context of trees for specific sentences. In other words, once a sentence is parsed, the resulting tree is walked to decorate each node with the appropriate head. Most current systems rely on a simple set of hand-written rules, such as a practical one for Penn Treebank grammars given in Collins (1999) but developed originally by Magerman (1995). For example their rule for finding the head of an NP is as follows Collins (1999, 238):

- If the last word is tagged POS, return last-word.
- Else search from right to left for the first child which is an NN, NNP, NNPS, NX, POS, or JJR.
- Else search from left to right for the first child which is an NP.
- Else search from right to left for the first child which is a $, ADJP, or PRN.
- Else search from right to left for the first child which is a CD.
- Else search from right to left for the first child which is a JJ, JJS, RB or QP.
- Else return the last word

Selected other rules from their set are shown in Fig. 12.13. For example, for VP rules of the form $VP \rightarrow Y_1 \cdots Y_n$, the algorithm would start from the left of $Y_1 \cdots Y_n$ looking for the first Y_i of type TO; if no TOs are found it would search for the first Y_i of type VBD; if no VBDs are found it would search for a VBP, and so on. See Collins (1999) for more details.

Parent Non-terminal	Direction	Priority List
ADJP	Left	NNS QP NN $ ADVP JJ VBN VBG ADJP JJR NP JJS DT FW RBR RBS SBAR RB
ADVP	Right	RB RBR RBS FW ADVP TO CD JJR JJ IN NP JJS NN
PRN	Left	
PRT	Right	RP
QP	Left	$ IN NNS NN JJ RB DT CD NCD QP JJR JJS
S	Left	TO IN VP S SBAR ADJP UCP NP
SBAR	Left	WHNP WHPP WHADVP WHADJP IN DT S SQ SINV SBAR FRAG
VP	Left	TO VBD VBN MD VBZ VB VBG VBP VP ADJP NN NNS NP

Figure 12.13 Selected head rules from Collins (1999). The set of head rules is often called a **head percolation table**.

12.5 Grammar Equivalence and Normal Form

A formal language is defined as a (possibly infinite) set of strings of words. This suggests that we could ask if two grammars are equivalent by asking if they generate the same set of strings. In fact it is possible to have two distinct context-free grammars generate the same language.

We usually distinguish two kinds of grammar equivalence: **weak equivalence** and **strong equivalence**. Two grammars are strongly equivalent if they generate the same set of strings *and* if they assign the same phrase structure to each sentence (allowing merely for renaming of the non-terminal symbols). Two grammars are weakly equivalent if they generate the same set of strings but do not assign the same phrase structure to each sentence.

NORMAL FORM

CHOMSKY NORMAL FORM

BINARY BRANCHING

It is sometimes useful to have a **normal form** for grammars, in which each of the productions takes a particular form. For example a context-free grammar is in **Chomsky Normal Form** (CNF) (Chomsky, 1963) if it is ϵ-free and if in addition each production is either of the form $A \rightarrow B\ C$ or $A \rightarrow a$. That is, the right-hand side of each rule either has two non-terminal symbols or one terminal symbol. Chomsky normal form grammars are **binary branching**, i.e. have binary trees (down to the prelexical nodes). We will make use of this binary branching property in the CKY parsing algorithm in Ch. 13.

Any grammar can be converted into a weakly-equivalent Chomsky normal form grammar. For example, a rule of the form

$$A \rightarrow B\ C\ D$$

can be converted into the following two CNF rules (Exercise 12.11 asks the reader to formulate the complete algorithm):

$$\begin{aligned} A &\rightarrow B\ X \\ X &\rightarrow C\ D \end{aligned}$$

Sometimes using binary branching can actually produce smaller grammars. For example the sentences that might be characterized as follows:

```
VP -> VBD NP PP*
```

are represented in the Penn Treebank by this series of rules:

```
VP → VBD PP
VP → VBD PP PP
VP → VBD PP PP PP
VP → VBD PP PP PP PP
...
```

but could also be generated by the following two-rule grammar:

(12.12)
```
VP → VBD PP
VP → VP PP
```

CHOMSKY-ADJUNCTION

To generate a symbol A with a potentially infinite sequence of symbols B by using a rule of the form A → A B is known as **Chomsky-adjunction**.

12.6 Finite-State and Context-Free Grammars

We argued in Sec. 12.1 that adequate models of grammar need to be able to represent complex interrelated facts about constituency, subcategorization, and dependency relations, and we implied that at the least the power of context-free grammars is needed to accomplish this. But why is it that we can't just use finite-state methods to capture these syntactic facts? The answer to this question is critical since, as we'll see in Ch. 13, there is a considerable price to be paid in terms of processing speed when one switches from regular languages to context-free ones.

There are two answers to this question. The first is mathematical; we'll show in Ch. 15 that given certain assumptions, that certain syntactic structures present in English (and other natural languages) make them not regular languages. The second answer is more subjective and has to do with notions of expressiveness; even when finite-state methods are capable of dealing with the syntactic facts in question, they often don't express them in ways that make generalizations obvious, lead to understandable formalisms, or produce structures of immediate use in subsequent semantic processing.

The mathematical objection will be discussed more fully in Ch. 15, but we'll briefly review it here. We mentioned in passing in Ch. 2 that there is a completely equivalent alternative to finite-state machines and regular expressions for describing regular languages, called **regular grammars**. The rules in a regular grammar are a restricted form of the rules in a context-free grammar because they are in right-linear or left-linear form. In a right-linear grammar, for example, the rules are all of the form $A \rightarrow w*$ or $A \rightarrow w * B$, that is the non-terminals either expand to a string of terminals or to a string of terminals followed by a non-terminal. These rules look an awful lot like the rules we've been using throughout this chapter, so what can't they do? What they can't do is express recursive **center-embedding** rules like the following, where a non-terminal is rewritten as itself, surrounded by (non-empty) strings:

(12.13) $$A \overset{*}{\Rightarrow} \alpha A \beta$$

In other words, a language can be generated by a finite-state machine if and only if the grammar that generates L that does not have any **center-embedded** recursions of this form (Chomsky, 1959; Bar-Hillel et al., 1961; Nederhof, 2000). Intuitively, this is because grammar rules in which the non-terminal symbols are always on either the right or left edge of a rule can be processed iteratively rather than recursively. Such center-embedding rules are needed to deal with artificial problems such as the language $a^n b^n$, or for practical problems such as checking for correctly matching delimiters in programming and markup languages. It turns out that there are no slam-dunk examples of this for English, but examples like the following give a flavor of the problem.

(12.14) The luggage arrived.

(12.15) The luggage that the passengers checked arrived.

(12.16) The luggage that the passengers that the storm delayed checked arrived.

At least in theory, this kind of embedding could go on, although it gets increasingly difficult to process such examples and they are luckily fairly rare outside textbooks like this one. Ch. 15 will discuss this and related issues as to whether or not even context-free grammars are up to the task.

So is there no role for finite-state methods in syntactic analysis? A quick review of the rules used for noun-phrases in this chapter, as well as those used in the Penn treebank grammar, reveals that a considerable portion of them can be handled by finite-state methods. Consider the following rule for a **noun group**, the pre-nominal and nominal portions of a noun phrase:

NOUN GROUP

$$Nominal \rightarrow (Det)\ (Card)\ (Ord)\ (Quant)\ (AP)\ Nominal$$

Assuming we convert the pre-nominal elements of this rule into terminals, this rule is effectively right-linear and can be captured by a finite-state machine. Indeed, it is possible to automatically build a regular grammar which is an approximation of a given context-free grammar; see the references at the end of the chapter. Thus for many practical purposes where matching syntactic and semantic rules aren't necessary, finite-state rules are quite sufficient.

12.7 DEPENDENCY GRAMMARS

DEPENDENCY GRAMMARS

We have focused in this chapter on context-free grammars because many available treebanks and parsers produce these kinds of syntactic representation. But in a class of grammar formalisms called **dependency grammars** that are becoming quite important in speech and language processing, constituents and phrase-structure rules do not play any fundamental role. Instead, the syntactic structure of a sentence is described purely in terms of words and binary semantic or syntactic relations between these words. Dependency grammars often draw heavily from the work of Tesnière (1959), and the name **dependency** might have been used first by early computational linguist David Hays. But this lexical dependency notion of grammar is in fact older than the relatively recent phrase-structure or constituency grammars, and has its roots in the ancient Greek and Indian linguistic traditions. Indeed the notion in traditional grammar of “parsing a sentence into subject and predicate” is based on lexical relations rather than constituent relations.

DEPENDENCY

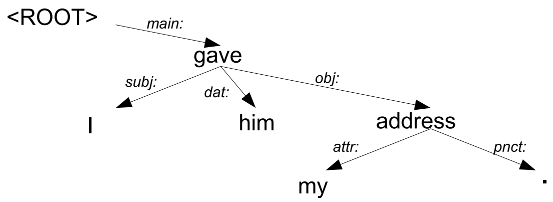


Figure 12.14 A sample dependency grammar parse, using the dependency formalism of Karlsson et al. (1995), after Järvinen and Tapanainen (1997).

Fig. 12.14 shows an example parse of the sentence *I gave him my address*, using the dependency grammar formalism of Järvinen and Tapanainen (1997) and Karlsson et al. (1995). Note that there are no non-terminal or phrasal nodes; each link in the parse tree holds between two lexical nodes (augmented with the special <ROOT> node). The links are drawn from a fixed inventory of around 35 relations, most of which roughly represent grammatical functions or very general semantic relations. Other dependency-based computational grammars, such as **Link Grammar** (Sleator and Temperley, 1993), use different but roughly overlapping links. The following table shows a few of the relations used in Järvinen and Tapanainen (1997):

LINK GRAMMAR

Dependency	Description
subj	syntactic subject
obj	direct object (incl. sentential complements)
dat	indirect object
pcomp	complement of a preposition
comp	predicate nominals (complements of copulas)
tmp	temporal adverbials
loc	location adverbials
attr	premodifying (attributive) nominals (genitives, etc.)
mod	nominal postmodifiers (prepositional phrases, etc.)

As we will see in Ch. 14, one advantage of dependency formalisms is the strong predictive parsing power that words have for their dependents. Knowing the identity of the verb is often a very useful cue for deciding which noun is likely to be the subject or the object. Dependency grammar researchers argue that one of the main advantages of pure dependency grammars is their ability to handle languages with relatively **free word order**. For example the word order in languages like Czech is much more flexible than in English; an *object* might occur before or after a *location adverbial* or a **comp**. A phrase-structure grammar would need a separate rule for each possible place in the parse tree that such an adverbial phrase could occur. A dependency grammar would just have one link-type representing this particular adverbial relation. Thus a dependency grammar abstracts away from word-order variation, representing only the information that is necessary for the parse.

FREE WORD ORDER

There are a number of computational implementations of dependency grammars; Link Grammar (Sleator and Temperley, 1993) and Constraint Grammar (Karlsson et al., 1995) are easily-available broad-coverage dependency grammars and parsers for English. Dependency grammars are also often used for other languages. Hajič (1998), for example, describes the 500,000 word Prague Dependency Treebank for Czech which has been used to train probabilistic dependency parsers (Collins et al., 1999).

12.7.1 The Relationship Between Dependencies and Heads

The reader may have noticed the similarity between dependency graphs like Fig. 12.14 and head structures like Fig. 12.12. In fact an (unlabeled) dependency graph can be automatically derived from a context-free parse by using the head rules; here's an algorithm from Xia and Palmer (2001):

1. Mark the head child of each node in a phrase structure, using the head percolation table.
2. In the dependency structure, make the head of each non- head-child depend on the head of the head-child.

This algorithm applied to the parse tree in Fig. 12.15 would produce the dependency structure in Fig. 12.16.

We will return to the discussion of heads and dependencies when we discuss lexicalized parsing in Ch. 14 and again when we introduce head features and subcategorization in Ch. 16.

12.7.2 Categorial Grammar

CATEGORIAL GRAMMAR

COMBINATORY CATEGORIAL GRAMMAR

CCG

Categorial grammar is an early lexicalized grammar model (Adjukiewicz, 1935; Bar-Hillel, 1953). In this section we will give a simplified overview of one important extension to categorial grammar, **combinatory categorial grammar** or **CCG** (Steedman, 1989, 2000). A categorial grammar has two components. The **categorial lexicon** associates each word with a syntactic and semantic category. The **combinatory rules** allow functions and arguments to be combined. There are two types of categories: functors and arguments. Arguments, like nouns, have simple categories like N. Verbs or determiners act as functors. For example, a determiner can be thought of as a function that applies to an N on its right to produce an NP. Such complex categories are

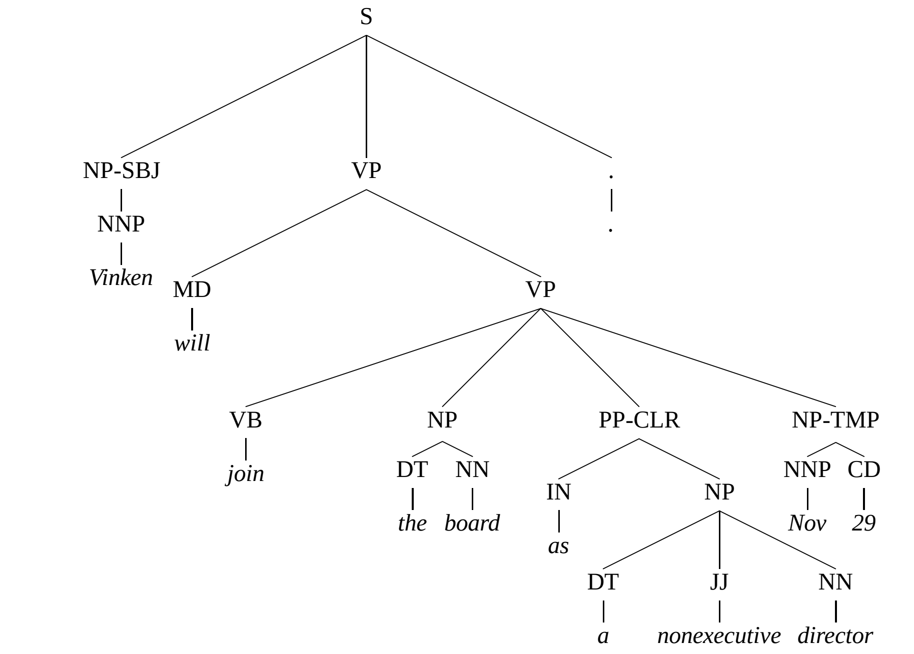


Figure 12.15 A phrase structure tree from the Wall Street Journal component of the Penn Treebank 3

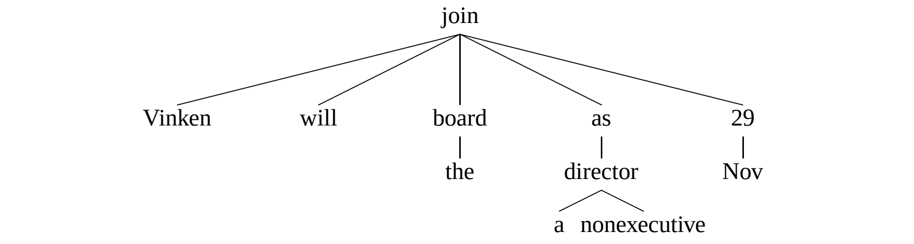


Figure 12.16 The dependency tree produced from Fig. 12.15 by the algorithm given above.

built using the X/Y and X\Y operators. X/Y means a function from Y to X, that is, something which combines with a Y on its right to produce an X. Determiners thus receive the category NP/N: something that combines with an N on its right to produce an NP. Transitive verbs might have the category VP/NP; something that combines with an NP on the right to produce a VP. Ditransitive verbs like *give* might have the category (VP/NP)/NP; something which combines with an NP on its right to yield a transitive verb. The simplest **combination rules** just combine an X/Y with a Y on its right to produce an X or a X\Y with a Y on its left to produce an X.

Consider the simple sentence *Harry eats apples* from Steedman (1989). Instead

the . [exhale] . . . [inhale] . . uh does American airlines . offer any . one way flights . uh one way fares, for one hundred and sixty one dollars
[mm] i'd like to leave i guess between um . [smack] . five o'clock no, five o'clock and uh, seven o'clock . P M
all right, [throat_clear] . . i'd like to know the . give me the flight . times . in the morning . for September twentieth . nineteen ninety one
uh one way
. w- wha- what is the lowest, cost, fare
[click] . i need to fly, betwee- . leaving . Philadelphia . to, Atlanta [exhale]
on United airlines . . give me, the . . time . . from New York . [smack] . to Boise-, to . I'm sorry . on United airlines . [uh] give me the flight, numbers, the flight times from . [uh] Boston . to Dallas

Figure 12.17 Sample spoken utterances from users interacting with an ATIS system.

of using a primitive VP category, let's assume that a finite verb phrase like *eat apples* has the category (S\NP); something which combines with an NP on the left to produce a sentence. *Harry* and *apples* are both NPs. *Eats* is a finite transitive verb which combines with an NP on the right to produce a finite VP: (S\NP)/NP. The derivation of S proceeds as follows:

(12.17)
$$\begin{array}{ccc}
\text{Harry} & \text{eats} & \text{apples} \\ \hline
\text{NP} & (\text{S}\backslash\text{NP})/\text{NP} & \text{NP} \\ \cline{2-3}
 & \text{S}\backslash\text{NP} & \\ \cline{1-2}
\text{S} & &
\end{array}$$

Modern categorial grammars include more complex combinatory rules which are needed for coordination and other complex phenomena, and also include composition of semantic categories as well as syntactic ones. See the end of the chapter for a pointer to useful references.

12.8 Spoken Language Syntax

The grammar of written English and the grammar of conversational spoken English share many features, but also differ in a number of respects. This section gives a quick sketch of a number of the characteristics of the syntax of spoken English.

UTTERANCE

We usually use the term **utterance** rather than **sentence** for the units of spoken language. Fig. 12.17 shows some sample spoken ATIS utterances that exhibit many aspects of spoken language grammar.

This is a standard style of transcription used in transcribing speech corpora for speech recognition. The comma "," marks a short pause, and each period "." marks a long pause. **Fragments** (incomplete words like *wha-* for incomplete *what*) are marked with with a dash, and the square brackets "[smack]" mark non-verbal events (**lipsmacks**, **breaths**, etc.).

There are a number of ways these utterances differ from written English sentences. One is in the lexical statistics; for example spoken English is much higher in pronouns than written English; the subject of a spoken sentence is almost invariably a pronoun. Spoken sentences often consist of short fragments or phrases (*one way* or *around four p.m.*, which are less common in written English. Spoken sentences have phonological, prosodic, and acoustic characteristics that of course written utterances don't have; we will return to these in Ch. 8. Finally, spoken sentences have various kinds of disfluencies (hesitations, repairs, restarts, etc) to be discussed below.

12.8.1 Disfluencies and Repair

DISFLUENCIES
REPAIR

Perhaps the most salient syntactic feature that distinguishes spoken and written language is the class of phenomena known individual as **disfluencies** and collectively as the phenomenon of **repair**.

UH
UM
RESTARTS

Disfluencies include the use of the words **uh** and **um**, word repetitions, **restarts**, and **word fragments**. The ATIS sentence in Fig. 12.18 shows examples of a restart and the use of *uh*. The restart here occurs when the speaker starts by asking for *one-way flights*. and then stops and corrects herself, restarting and asking about *one-way fares*.

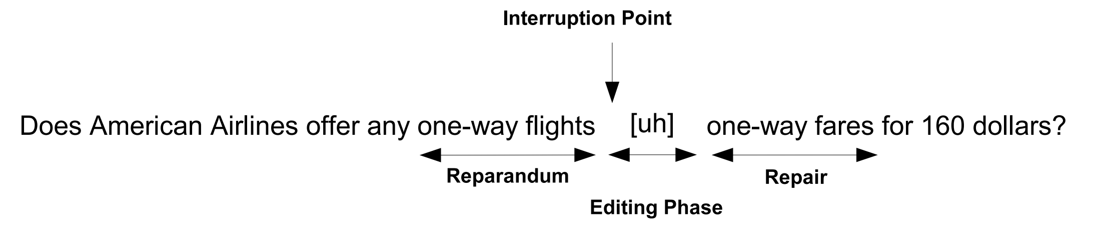

Interruption Point
Does American Airlines offer any one-way flights
[uh]
one-way fares for 160 dollars?
Reparandum
Editing Phase
Repair

Figure 12.18 An example of a disfluency (after Shriberg (1994); terminology is from Levelt (1983)).

REPARANDUM
REPAIR
INTERRUPTION POINT
EDIT TERMS

The segment *one-way flights* is referred to as the **reparandum**, and the replacing sequence *one-way fares* is referred to as the **repair**. The repair is also called the **fluent** region. The **interruption point**, where the speaker breaks off the original word sequence, here occurs right after the word *flights*. In the editing phase we see what are often called **edit terms**, such as *you know*, *I mean*, *uh*, and *um*.

FILLED PAUSES

The words *uh* and *um* (sometimes called **filled pauses** or **fillers**) are generally treated like regular words in speech recognition lexicons and grammars.

FRAGMENTS

Incomplete words like *wha-* and *betwee-* in Fig. 12.17 are known as **fragments**. Fragments are extremely problematic for speech recognition systems, since they are often incorrectly attached to previous or following words, resulting in word missegmentation.

Disfluencies are very common. One count in the Switchboard Treebank corpus found that 37% of the sentences with more than two words were disfluent in some way. Indeed, the word *uh* is one of the most frequent words in Switchboard.

For applications like speech understanding, where our goal is to build a meaning for the input sentence, it may be useful to detect these restarts in order to edit out what the speaker probably considered the "corrected" words. For example in the sentence above, if we could detect that there was a restart, we could just delete the reparandum, and parse the remaining parts of the sentence:

Does American airlines offer any one-way flights uh one-way fares for 160 dollars?

How do disfluencies interact with the constituent structure of the sentence? Hindle (1983) showed that the repair often has the same structure as the constituent just before the interruption point. Thus in the example above, the repair is an NP, as is the reparandum. This means that if it is possible to automatically find the interruption point, it is also often possible to automatically detect the boundaries of the reparandum.

There are other interactions between disfluencies and syntactic structure. For example when there is a disfluency immediately after a subject NP, the repair always repeats the subject but not the preceding discourse marker. If the repair happens after an auxiliary or main verb, the verb and subject are (almost) always recycled together (Fox and Jasperson, 1995).

12.8.2 Treebanks for Spoken Language

Treebanks for spoken corpora like Switchboard use an augmented notation to deal with spoken language phenomena like disfluencies. Fig. 12.19 shows the parse tree for Switchboard sentence (12.18). This sentence shows how the Treebank marks disfluencies; square brackets are used to separate out the entire repair area, including the reparandum, editing phase, and the repair. The plus symbol marks the end of the reparandum.

(12.18) But I don’t have [any, + {F uh, } any] real idea

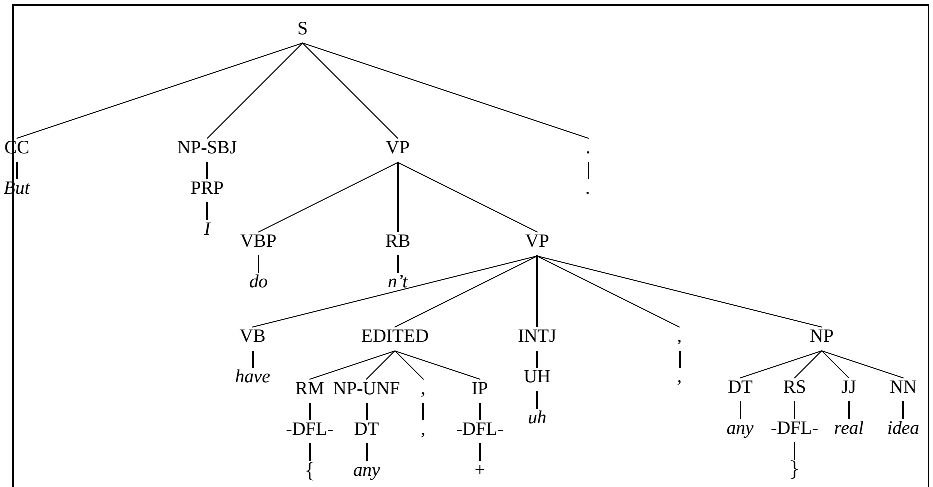


Figure 12.19 Penn Treebank III parse tree for a Switchboard sentence, showing how the disfluency information is represented in the parse tree. Note the .EDITED node, with the .RM and .RS nodes marking the beginning and end of the repair portion, and the use of the filled pause *uh*.

12.9 Grammars and Human Processing

Do people use context-free grammars in their mental processing of language? It has proved very difficult to find clear-cut evidence that they do. For example, some early experiments asked subjects to judge which words in a sentence were more closely connected (Levelt, 1970), finding that their intuitive groupings corresponded to syntactic constituents. Other experimenters examined the role of constituents in auditory comprehension by having subjects listen to sentences while also listening to short "clicks" at different times. Fodor and Bever (1965) found that subjects often mis-heard the clicks as if they occurred at constituent boundaries. They argued that the constituent was thus a "perceptual unit" which resisted interruption. Unfortunately there were severe methodological problems with the click paradigm (see e.g., Clark and Clark (1977) for a discussion).

A broader problem with all these early studies is that they do not control for the fact that constituents are often semantic units as well as syntactic units. Thus, as will be discussed further in Ch. 18, *a single odd block* is a constituent (an *NP*) but also a semantic unit (an object of type BLOCK which has certain properties). Thus experiments which show that people notice the boundaries of constituents could simply be measuring a semantic rather than a syntactic fact.

Thus it is necessary to find evidence for a constituent which is *not* a semantic unit. Furthermore, since there are many non-constituent-based theories of grammar based on lexical dependencies, it is important to find evidence that cannot be interpreted as a *lexical* fact; that is, evidence for constituency that is not based on particular words.

One suggestive series of experiments arguing for constituency has come from Kathryn Bock and her colleagues. Bock and Loebell (1990), for example, avoided all these earlier pitfalls by studying whether a subject who uses a particular syntactic constituent (e.g., a verb-phrase of a particular type, like *V NP PP*), is more likely to use the constituent in following sentences. In other words, they asked whether use of a constituent *primes* its use in subsequent sentences. As we saw in previous chapters, priming is a common way to test for the existence of a mental structure. Bock and Loebell relied on the English **ditransitive alternation**. A ditransitive verb is one like *give* which can take two arguments:

(12.19) The wealthy widow gave [$_{NP}$ the church] [$_{NP}$ her Mercedes].

The verb *give* allows another possible subcategorization frame, called a **prepositional dative** in which the indirect object is expressed as a prepositional phrase:

(12.20) The wealthy widow gave [$_{NP}$ her Mercedes] [$_{PP}$ to the church].

ALTERNATIONS

As we discussed on page 18, many verbs other than *give* have such **alternations** (*send*, *sell*, etc.; see Levin (1993) for a summary of many different alternation patterns). Bock and Loebell relied on these alternations by giving subjects a picture, and asking them to describe it in one sentence. The picture was designed to elicit verbs like *give* or *sell* by showing an event such as a boy handing an apple to a teacher. Since these verbs alternate, subjects might, for example, say *The boy gave the apple to the teacher* or *The boy gave the teacher an apple*.

Before describing the picture, subjects were asked to read an unrelated “priming” sentence out loud; the priming sentences either had *V NP NP* or *V NP PP* structure. Crucially, while these priming sentences had the same *constituent structure* as the dative alternation sentences, they did not have the same *semantics*. For example, the priming sentences might be prepositional *locatives*, rather than *datives*:

(12.21) IBM moved [$_{NP}$ a bigger computer] [$_{PP}$ to the Sears store].

Bock and Loebell found that subjects who had just read a *V NP PP* sentence were more likely to use a *V NP PP* structure in describing the picture. This suggested that the use of a particular constituent *primed* the later use of that constituent, and hence that the constituent must be mentally represented in order to prime and be primed.

In more recent work, Bock and her colleagues have continued to find evidence for this kind of constituency structure.

12.10 Summary

This chapter has introduced a number of fundamental concepts in syntax via the **context-free grammar**.

- In many languages, groups of consecutive words act as a group or a **constituent**, which can be modeled by **context-free grammars** (also known as **phrase-structure grammars**).
- A context-free grammar consists of a set of **rules** or **productions**, expressed over a set of **non-terminal** symbols and a set of **terminal** symbols. Formally, a particular **context-free language** is the set of strings which can be **derived** from a particular **context-free grammar**.
- A **generative grammar** is a traditional name in linguistics for a formal language which is used to model the grammar of a natural language.
- There are many sentence-level grammatical constructions in English; **declarative**, **imperative**, **yes-no-question**, and **wh-question** are four very common types, which can be modeled with context-free rules.
- An English **noun phrase** can have **determiners**, **numbers**, **quantifiers**, and **adjective phrases** preceding the **head noun**, which can be followed by a number of **postmodifiers**; **gerundive** VPs, **infinitives** VPs, and **past participial** VPs are common possibilities.
- **Subjects** in English **agree** with the main verb in person and number.
- Verbs can be **subcategorized** by the types of **complements** they expect. Simple subcategories are **transitive** and **intransitive**; most grammars include many more categories than these.
- The correlate of **sentences** in spoken language are generally called **utterances**. Utterances may be **disfluent**, containing **filled pauses** like *um* and *uh*, **restarts**, and **repairs**.
- **Treebanks** of parsed sentences exist for many genres of English and for many languages. Treebanks can be searched using tree-search tools.

- Any context-free grammar can be converted to **Chomsky normal form**, in which the right-hand-side of each rule has either two non-terminals or a single terminal.
- Context-free grammars are more powerful than finite-state automata, but it is nonetheless possible to **approximate** a context-free grammar with a FSA.
- There is some evidence that constituency plays a role in the human processing of language.

Bibliographical and Historical Notes

“den sprachlichen Ausdruck für die willkürliche Gliederung einer Gesammtvorstellung in ihre in logische Beziehung zueinander gesetzten Bestandteile”
“the linguistic expression for the arbitrary division of a total idea into its constituent parts placed in logical relations to one another”

Wundt’s (1900:240) definition of the sentence; the origin of
the idea of phrasal constituency, cited in Percival (1976).

According to Percival (1976), the idea of breaking up a sentence into a hierarchy of constituents appeared in the *Völkerpsychologie* of the groundbreaking psychologist Wilhelm Wundt (Wundt, 1900). Wundt’s idea of constituency was taken up into linguistics by Leonard Bloomfield in his early book *An Introduction to the Study of Language* (Bloomfield, 1914). By the time of his later book *Language* (Bloomfield, 1933), what was then called “immediate-constituent analysis” was a well-established method of syntactic study in the United States. By contrast, traditional European grammar, dating from the Classical period, defined relations between *words* rather than constituents, and European syntacticians retained this emphasis on such **dependency** grammars.

American Structuralism saw a number of specific definitions of the immediate constituent, couched in terms of their search for a “discovery procedure”; a methodological algorithm for describing the syntax of a language. In general, these attempt to capture the intuition that “The primary criterion of the immediate constituent is the degree in which combinations behave as simple units” (Bazell, 1966, p. 284). The most well-known of the specific definitions is Harris’ idea of distributional similarity to individual units, with the *substitutability* test. Essentially, the method proceeded by breaking up a construction into constituents by attempting to substitute simple structures for possible constituents—if a substitution of a simple form, say *man*, was substitutable in a construction for a more complex set (like *intense young man*), then the form *intense young man* was probably a constituent. Harris’s test was the beginning of the intuition that a constituent is a kind of equivalence class.

The first formalization of this idea of hierarchical constituency was the **phrase-structure grammar** defined in Chomsky (1956), and further expanded upon (and argued against) in Chomsky (1957) and Chomsky (1975). From this time on, most generative linguistic theories were based at least in part on context-free grammars or generalizations of them (such as Head-Driven Phrase Structure Grammar (Pollard and Sag, 1994), Lexical-Functional Grammar (Bresnan, 1982), Government and Binding (Chomsky, 1981), and Construction Grammar (Kay and Fillmore, 1999), inter alia); many of these theories used schematic context-free templates known as **X-bar schemata** which also relied on the notion of syntactic head.

X-BAR SCHEMATA

Shortly after Chomsky's initial work, the context-free grammar was rediscovered by Backus (1959) and independently by Naur et al. (1960) in their descriptions of the ALGOL programming language; Backus (1996) noted that he was influenced by the productions of Emil Post and that Naur's work was independent of his (Backus') own. (Recall the discussion on page **??** of multiple invention in science.) After this early work, a great number of computational models of natural language processing were based on context-free grammars because of the early development of efficient algorithms to parse these grammars (see Ch. 13).

As we have already noted, grammars based on context-free rules are not ubiquitous. Various classes of extensions to CFGs are designed specifically to handle long-distance dependencies. We noted earlier that some grammars treat long-distance-dependent items as being related semantically but not syntactically; the surface syntax does not represent the long-distance link (Kay and Fillmore, 1999; Culicover and Jackendoff, 2005). But there are alternatives. One extended formalism is **Tree Adjoining Grammar** (TAG) (Joshi, 1985). The primary data structure in Tree Adjoining Grammar is the tree, rather than the rule. Trees come in two kinds; **initial trees** and **auxiliary trees**. Initial trees might, for example, represent simple sentential structures, while auxiliary trees are used to add recursion into a tree. Trees are combined by two operations called **substitution** and **adjunction**. The adjunction operation is used to handle long-distance dependencies. See Joshi (1985) for more details. An extension of Tree Adjoining Grammar called Lexicalized Tree Adjoining Grammars will be discussed in Ch. 14. Tree Adjoining Grammar is a member of the family of **mildly context-sensitive languages** to be introduced in Ch. 15.

We mentioned on page 21 another way of handling long-distance dependencies, based on the use of empty categories and co-indexing. The Penn Treebank uses this model, which draws (in various Treebank corpora) from the Extended Standard Theory and Minimalism (Radford, 1997).

Representative examples of grammars that are based on word relations rather than constituency include the dependency grammar of Mel'čuk (1979), the Word Grammar of Hudson (1984), and the Constraint Grammar of Karlsson et al. (1995).

There are a variety of algorithms for building a regular grammar which approximates a CFG (Pereira and Wright, 1997; Johnson, 1998; Langendoen and Langsam, 1987; Nederhof, 2000; Mohri and Nederhof, 2001).

Readers interested in the grammar of English should get one of the three large reference grammars of English: Huddleston and Pullum (2002), Biber et al. (1999), and Quirk et al. (1985), Another useful reference is McCawley (1998).

GENERATIVE

FUNCTIONAL

There are many good introductory textbooks on syntax from different perspectives. Sag et al. (2003) is an introduction to syntax from a **generative** perspective, focusing on the use of phrase-structure, unification, and the type-hierarchy in Head-Driven Phrase Structure Grammar. Van Valin and La Polla (1997) is an introduction from a **functional** perspective, focusing on cross-linguistic data and on the functional motivation for syntactic structures.

See Bach (1988) for an introduction to basic categorial grammar. Various extensions to categorial grammars are presented in Lambek (1958), Dowty (1979), and Ades and Steedman (1982) inter alia; the other papers in Oehrle et al. (1988) give a survey of extensions. Combinatory categorial grammar is presented in Steedman (1989, 2000); see Steedman and Baldridge (2003) for a tutorial introduction. See Ch. 18 for a discussion of semantic composition.

EXERCISES

12.1 Draw tree structures for the following ATIS phrases:

a. Dallas
b. from Denver
c. after five p.m.
d. arriving in Washington
e. early flights
f. all redeye flights
g. on Thursday
h. a one-way fare
i. any delays in Denver

12.2 Draw tree structures for the following ATIS sentences:

a. Does American airlines have a flight between five a.m. and six a.m.
b. I would like to fly on American airlines.
c. Please repeat that.
d. Does American 487 have a first class section?
e. I need to fly between Philadelphia and Atlanta.
f. What is the fare from Atlanta to Denver?
g. Is there an American airlines flight from Philadelphia to Dallas?

12.3 Augment the grammar rules on page 16 to handle pronouns. Deal properly with person and case.

12.4 Modify the noun phrase grammar of Sections 12.3.3–12.3.4 to correctly model mass nouns and their agreement properties

12.5 How many types of *NP*s would the rule on page 12 expand to if we didn't allow parentheses in our grammar formalism?

12.6 Assume a grammar that has many *VP* rules for different subcategorizations, as expressed in Sec. 12.3.5, and differently subcategorized verb rules like *Verb-with-NP-complement*. How would the rule for post-nominal relative clauses (12.7) need to be modified if we wanted to deal properly with examples like *the earliest flight that you have*? Recall that in such examples the pronoun *that* is the object of the verb *get*. Your rules should allow this noun phrase but should correctly rule out the ungrammatical S **I get*.

12.7 Does your solution to the previous problem correctly model the NP *the earliest flight that I can get*? How about *the earliest flight that I think my mother wants me to book for her*? Hint: this phenomenon is called **long-distance dependency**.

12.8 Write rules expressing the verbal subcategory of English auxiliaries; for example you might have a rule *verb-with-bare-stem-VP-complement* → *can*.

POSSESSIVE
GENITIVE

12.9 *NP*s like *Fortune's office* or *my uncle's marks* are called **possessive** or **genitive** noun phrases. A possessive noun phrase can be modeled by treating the sub-NP like *Fortune's* or *my uncle's* as a determiner of the following head noun. Write grammar rules for English possessives. You may treat *'s* as if it were a separate word (i.e., as if there were always a space before *'s*).

12.10 Page 9 discussed the need for a *Wh-NP* constituent. The simplest *Wh-NP* is one of the *Wh-pronouns* (*who, whom, whose, which*). The Wh-words *what* and *which* can be determiners: *which four will you have?*, *what credit do you have with the Duke?* Write rules for the different types of *Wh-NP*s.

12.11 Write an algorithm for converting an arbitrary context-free grammar into Chomsky normal form.

Ades, A. E. and Steedman, M. (1982). On the order of words. *Linguistics and Philosophy*, *4*, 517–558.

Adjukiewicz, K. (1935). Die syntaktische Konnexität. *Studia Philosophica*, *1*, 1–27. English translation “Syntactic Connexion” by H. Weber in McCall, S. (Ed.) *Polish Logic*, pp. 207–231, Oxford University Press, Oxford, 1967.

Bach, E. (1988). Categorial grammars as theories of language. In Oehrle, R. T., Bach, E., and Wheeler, D. (Eds.), *Categorial Grammars and Natural Language Structures*, pp. 17–34. D. Reidel, Dordrecht.

Backus, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zurch ACM-GAMM Conference. In *Information Processing: Proceedings of the International Conference on Information Processing, Paris*, pp. 125–132. UNESCO.

Backus, J. W. (1996). Transcript of question and answer session. In Wexelblat, R. L. (Ed.), *History of Programming Languages*, p. 162. Academic Press.

Bar-Hillel, Y. (1953). A quasi-arithmetical notation for syntactic description. *Language*, *29*, 47–58. Reprinted in Y. Bar-Hillel. (1964). *Language and Information: Selected Essays on their Theory and Application*, Addison-Wesley 1964, 61–74.

Bar-Hillel, Y., Perles, M., and Shamir, E. (1961). On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, *14*, 143–172. Reprinted in Y. Bar-Hillel. (1964). *Language and Information: Selected Essays on their Theory and Application*, Addison-Wesley 1964, 116–150.

Bazell, C. E. (1952/1966). The correspondence fallacy in structural linguistics. In Hamp, E. P., Householder, F. W., and Austerlitz, R. (Eds.), *Studies by Members of the English Department, Istanbul University (3), reprinted in Readings in Linguistics II (1966)*, pp. 271–298. University of Chicago Press, Chicago.

Biber, D., Johansson, S., Leech, G., Conrad, S., and Finegan, E. (1999). *Longman Grammar of Spoken and Written English*. Pearson ESL, Harlow.

Bies, A., Ferguson, M., Katz, K., and MacIntyre, R. (1995). Bracketing guidelines for Treebank II style Penn Treebank Project..

Bloomfield, L. (1914). *An Introduction to the Study of Language*. Henry Holt and Company, New York.

Bloomfield, L. (1933). *Language*. University of Chicago Press, Chicago.

Bock, K. and Loebell, H. (1990). Framing sentences. *Cognition*, *35*, 1–39.

Bresnan, J. (Ed.). (1982). *The Mental Representation of Grammatical Relations*. MIT Press.

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *AAAI-97*, Menlo Park, pp. 598–603. AAAI Press.

Chomsky, N. (1956). Three models for the description of language. *IRI Transactions on Information Theory*, *2*(3), 113–124.

Chomsky, N. (1956/1975). *The Logical Structure of Linguistic Theory*. Plenum.

Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, *2*, 137–167.

Chomsky, N. (1963). Formal properties of grammars. In Luce, R. D., Bush, R., and Galanter, E. (Eds.), *Handbook of Mathematical Psychology*, Vol. 2, pp. 323–418. Wiley.

Chomsky, N. (1981). *Lectures on Government and Binding*. Foris, Dordrecht.

Clark, H. H. and Clark, E. V. (1977). *Psychology and Language*. Harcourt Brace Jovanovich.

Collins, M. (1999). *Head-driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

Collins, M., Hajič, J., Ramshaw, L. A., and Tillmann, C. (1999). A statistical parser for Czech. In *ACL-99*, College Park, MA, pp. 505–512. ACL.

Culicover, P. W. and Jackendoff, R. (2005). *Simpler Syntax*. Oxford University Press.

Dowty, D. R. (1979). *Word Meaning and Montague Grammar*. D. Reidel, Dordrecht.

Fodor, J. A. and Bever, T. G. (1965). The psychological reality of linguistic segments. *Journal of Verbal Learning and Verbal Behavior*, *4*, 414–420.

Fox, B. and Jasperson, R. (1995). A syntactic exploration of repair in English conversation. In Davis, P. (Ed.), *Descriptive and Theoretical Modes in the Alternative Linguistics*, pp. 77–134. John Benjamins, Amsterdam. In press.

Gazdar, G., Klein, E., Pullum, G. K., and Sag, I. A. (1985). *Generalized Phrase Structure Grammar*. Basil Blackwell, Oxford.

Hajič, J. (1998). *Building a Syntactically Annotated Corpus: The Prague Dependency Treebank*, pp. 106–132. Karolinum, Prague/Praha.

Halliday, M. A. K. (1985). *An Introduction to Functional Grammar*. Edward Arnold, London.

Harris, Z. S. (1946). From morpheme to utterance. *Language*, *22*(3), 161–183.

Hemphill, C. T., Godfrey, J., and Doddington, G. (1990). The ATIS spoken language systems pilot corpus. In *Proceedings DARPA Speech and Natural Language Workshop*, Hidden Valley, PA, pp. 96–101. Morgan Kaufmann.

Hindle, D. (1983). Deterministic parsing of syntactic nonfluencies. In *ACL-83*, pp. 123–128. ACL.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

Huddleston, R. and Pullum, G. K. (2002). *The Cambridge grammar of the English language*. Cambridge University Press.

Hudson, R. A. (1984). *Word Grammar*. Basil Blackwell, Oxford.

Järvinen, T. and Tapanainen, P. (1997). A dependency parser for English. Tech. rep. TR-1, Department of General Linguistics, University of Helsinki, Helsinki.

Johnson, M. (1998). Finite-state approximation of constraint-based grammars using left-corner grammar transforms. In *COLING/ACL-98*, Montreal, pp. 619–623.

Joshi, A. K. (1985). Tree adjoining grammars: how much context-sensitivity is required to provide reasonable structural descriptions?. In Dowty, D. R., Karttunen, L., and Zwicky, A. (Eds.), *Natural Language Parsing*, pp. 206–250. Cambridge University Press.

Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (Eds.). (1995). *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, Berlin.

Kay, P. and Fillmore, C. J. (1999). Grammatical constructions and linguistic generalizations: The What's X Doing Y? construction. *Language*, *75*(1), 1–33.

Lambek, J. (1958). The mathematics of sentence structure. *American Mathematical Monthly*, *65*(3), 154–170.

Langendoen, D. T. and Langsam, Y. (1987). On the design of finite transducers for parsing phrase-structure languages. In Manaster-Ramer, A. (Ed.), *Mathematics of Language*, pp. 191–235. John Benjamins, Amsterdam.

Levelt, W. J. M. (1970). A scaling approach to the study of syntactic relations. In d'Arcais, G. B. F. and Levelt, W. J. M. (Eds.), *Advances in psycholinguistics*, pp. 109–121. North-Holland, Amsterdam.

Levelt, W. J. M. (1983). Monitoring and self-repair in speech. *Cognition*, *14*, 41–104.

Levin, B. (1993). *English Verb Classes And Alternations: A Preliminary Investigation*. University of Chicago Press, Chicago.

Macleod, C., Grishman, R., and Meyers, A. (1998). COMLEX Syntax Reference Manual Version 3.0. Linguistic Data Consortium.

Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *ACL-95*.

Marcus, M. P., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The Penn Treebank: Annotating predicate argument structure. In *ARPA Human Language Technology Workshop*, Plainsboro, NJ, pp. 114–119. Morgan Kaufmann.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, *19*(2), 313–330.

McCawley, J. D. (1998). *The Syntactic Phenomena of English*. University of Chicago Press, Chicago.

Mel'čuk, I. A. (1979). *Studies in dependency syntax*. Karoma Publishers, Ann Arbor.

Mohri, M. and Nederhof, M. J. (2001). Regular approximation of context-free grammars through transformation. In Junqua, J.-C. and van Noord, G. (Eds.), *Robustness in Language and Speech Technology*, pp. 153–163. Kluwer.

Naur, P., Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., van Wijnagaarden, A., and Woodger, M. (1960). Report on the algorithmic language ALGOL 60. *Communications of the ACM*, *3*(5), 299–314. Revised in CACM 6:1, 1-17, 1963.

Nederhof, M.-J. (2000). Practical experiments with regular approximation of context-free languages. *Computational Linguistics*, *26*(1), 17–44.

Oehrle, R. T., Bach, E., and Wheeler, D. (Eds.). (1988). *Categorial Grammars and Natural Language Structures*. D. Reidel, Dordrecht.

Percival, W. K. (1976). On the historical source of immediate constituent analysis. In McCawley, J. D. (Ed.), *Syntax and Semantics Volume 7, Notes from the Linguistic Underground*, pp. 229–242. Academic Press.

Pereira, F. C. N. and Wright, R. N. (1997). Finite-state approximation of phrase-structure grammars. In Roche, E. and Schabes, Y. (Eds.), *Finite-State Language Processing*, pp. 149–174. MIT Press.

Pito, R. (1993). Tgrepdoc man page..

Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Pullum, G. K. (1991). *The Great Eskimo Vocabulary Hoax*. University of Chicago, Chicago, IL.

Quirk, R., Greenbaum, S., Leech, G., and Svartvik, J. (1985). *A Comprehensive Grammar of the English Language*. Longman, London.

Radford, A. (1988). *Transformational Grammar: A First Course*. Cambridge University Press.

Radford, A. (1997). *Syntactic Theory and the Structure of English: A Minimalist Approach*. Cambridge University Press.

Rohde, D. L. T. (2005). Tgrep2 user manual..

Sag, I. A., Wasow, T., and Bender, E. M. (Eds.). (2003). *Syntactic Theory: A Formal Introduction*. CSLI Publications, Stanford, CA.

Sanfilippo, A. (1993). LKB encoding of lexical knowledge. In Briscoe, T., de Paiva, V., and Copestake, A. (Eds.), *Inheritance, Defaults, and the Lexicon*, pp. 190–222. Cambridge University Press.

Shriberg, E. (1994). *Preliminaries to a Theory of Speech Disfluencies*. Ph.D. thesis, University of California, Berkeley, CA. (unpublished).

Sleator, D. and Temperley, D. (1993). Parsing English with a link grammar. In *Proceeedings, Third International Workshop on Parsing Technologies*, Tilburg, The Netherlands/Durbuy, Belgium.

Steedman, M. (1989). Constituency and coordination in a combinatory grammar. In Baltin, M. R. and Kroch, A. S. (Eds.), *Alternative Conceptions of Phrase Structure*, pp. 201–231. University of Chicago, Chicago.

Steedman, M. (2000). *The Syntactic Process*. The MIT Press.

Steedman, M. and Baldridge, J. (2003). Combinatory categorial grammar. Unpublished tutorial paper.

Tesnière, L. (1959). *Éléments de Syntaxe Structurale*. Librairie C. Klincksieck, Paris.

Van Valin, Jr., R. D. and La Polla, R. (1997). Syntax: Structure, meaning, and function..

Wundt, W. (1900). *Völkerpsychologie: eine Untersuchung der Entwicklungsgesetze von Sprache, Mythus, und Sitte*. W. Engelmann, Leipzig. Band II: Die Sprache, Zweiter Teil.

Xia, F. and Palmer, M. (2001). Converting dependency structures to phrase structures. In *HLT-01*, San Diego, pp. 1–5.

13 PARSING WITH CONTEXT-FREE GRAMMARS

There are and can exist but two ways of investigating and discovering truth. The one hurries on rapidly from the senses and particulars to the most general axioms, and from them... derives and discovers the intermediate axioms. The other constructs its axioms from the senses and particulars, by ascending continually and gradually, till it finally arrives at the most general axioms.

Francis Bacon, *Novum Organum* Book I.19 (1620)

We defined parsing in Ch. 3 as a combination of recognizing an input string and assigning a structure to it. Syntactic parsing, then, is the task of recognizing a sentence and assigning a syntactic structure to it. This chapter focuses on the kind of structures assigned by context-free grammars of the kind described in Ch. 12. However, since they are a purely declarative formalism, context-free grammars don't specify *how* the parse tree for a given sentence should be computed, therefore we'll need to specify algorithms that employ these grammars to produce trees. This chapter presents three of the most widely used parsing algorithms for automatically assigning a complete context-free (phrase structure) tree to an input sentence.

These kinds of parse trees are directly useful in applications such as **grammar checking** in word-processing systems; a sentence which cannot be parsed may have grammatical errors (or at least be hard to read). More typically, however, parse trees serve as an important intermediate stage of representation for **semantic analysis** (as we will see in Ch. 18), and thus plays an important role in applications like **question answering** and **information extraction**. For example, to answer the question

What books were written by British women authors before 1800?

we'll need to know that the subject of the sentence was *what books* and that the by-adjunct was *British women authors* to help us figure out that the user wants a list of books (and not a list of authors).

Before presenting any parsing algorithms, we begin by describing some of the factors that motivate the standard algorithms. First, we revisit the **search metaphor** for parsing and recognition, which we introduced for finite-state automata in Ch. 2, and talk about the **top-down** and **bottom-up** search strategies. We then discuss how the

S → *NP VP*	*Det* → *that* | *this* | *a*
S → *Aux NP VP*	*Noun* → *book* | *flight* | *meal* | *money*
S → *VP*	*Verb* → *book* | *include* | *prefer*
NP → *Pronoun*	*Pronoun* → *I* | *she* | *me*
NP → *Proper-Noun*	*Proper-Noun* → *Houston* | *TWA*
NP → *Det Nominal*	*Aux* → *does*
Nominal → *Noun*	*Preposition* → *from* | *to* | *on* | *near* | *through*
Nominal → *Nominal Noun*	
Nominal → *Nominal PP*	
VP → *Verb*	
VP → *Verb NP*	
VP → *Verb NP PP*	
VP → *Verb PP*	
VP → *VP PP*	
PP → *Preposition NP*	

Figure 13.1 The $\mathscr{L}_1$ miniature English grammar and lexicon.

ambiguity problem rears its head again in syntactic processing, and how it ultimately makes simplistic approaches based on backtracking infeasible.

The sections that follow then present the Cocke-Kasami-Younger (CKY) algorithm (Kasami, 1965; Younger, 1967), the Earley algorithm (Earley, 1970), and the Chart Parsing approach (Kay, 1986; Kaplan, 1973). These approaches all combine insights from bottom-up and top-down parsing with dynamic programming to efficiently handle complex inputs. Recall that we've already seen several applications of dynamic programming algorithms in earlier chapters — Minimum-Edit-Distance, Viterbi, Forward. Finally, we discuss **partial parsing methods**, for use in situations where a superficial syntactic analysis of an input may be sufficient.

13.1 Parsing as Search

Chs. 2 and 3 showed that finding the right path through a finite-state automaton, or finding the right transduction for an input, can be viewed as a search problem. For finite-state automata, the search is through the space of all possible paths through a machine. In syntactic parsing, the parser can be viewed as searching through the space of possible parse trees to find the correct parse tree for a given sentence. Just as the search space of possible paths was defined by the structure of an automata, so the search space of possible parse trees is defined by a grammar. Consider the following ATIS sentence:

(13.1) Book that flight.

Fig. 13.1 introduces the $\mathscr{L}_1$ grammar, which consists of the $\mathscr{L}_0$ grammar from the last chapter with a few additional rules. Given this grammar, the correct parse tree for this example would be the one shown in Fig. 13.2.

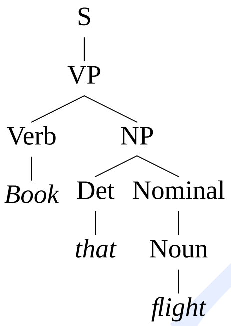


Figure 13.2 The parse tree for the sentence *Book that flight* according to grammar $\mathscr{L}_1$.

How can we use $\mathscr{L}_1$ to assign the parse tree in Fig. 13.2 to this example? The goal of a parsing search is to find all the trees whose root is the start symbol *S* and which cover exactly the words in the input. Regardless of the search algorithm we choose, there are two kinds of constraints that should help guide the search. One set of constraints comes from the data, that is, the input sentence itself. Whatever else is true of the final parse tree, we know that there must be three leaves, and they must be the words *book*, *that*, and *flight*. The second kind of constraint comes from the grammar. We know that whatever else is true of the final parse tree, it must have one root, which must be the start symbol *S*.

RATIONALIST
EMPIRICIST TRADITION

These two constraints, invoked by Bacon at the start of this chapter, give rise to the two search strategies underlying most parsers: **top-down** or **goal-directed search**, and **bottom-up** or **data-directed search**. These constraints are more than just search strategies. They reflect two important insights in the western philosophical tradition: the **rationalist** tradition, which emphasizes the use of prior knowledge, and the **empiricist tradition** tradition, which emphasizes the data in front of us.

13.1.1 Top-Down Parsing

TOP-DOWN
PLY

A **top-down** parser searches for a parse tree by trying to build from the root node *S* down to the leaves. Let's consider the search space that a top-down parser explores, assuming for the moment that it builds all possible trees in parallel. The algorithm starts by assuming the input can be derived by the designated start symbol *S*. The next step is to find the tops of all trees which can start with *S*, by looking for all the grammar rules with *S* on the left-hand side. In the grammar in Fig. 13.1, there are three rules that expand *S*, so the second **ply**, or level, of the search space in Fig. 13.3 has three partial trees.

We next expand the constituents in these three new trees, just as we originally expanded *S*. The first tree tells us to expect an *NP* followed by a *VP*, the second expects an *Aux* followed by an *NP* and a *VP*, and the third a *VP* by itself. To fit the search space on the page, we have shown in the third ply of Fig. 13.3 only a subset of the trees that result from the expansion of the left-most leaves of each tree. At each ply of the search space we use the right-hand sides of the rules to provide new sets of expectations

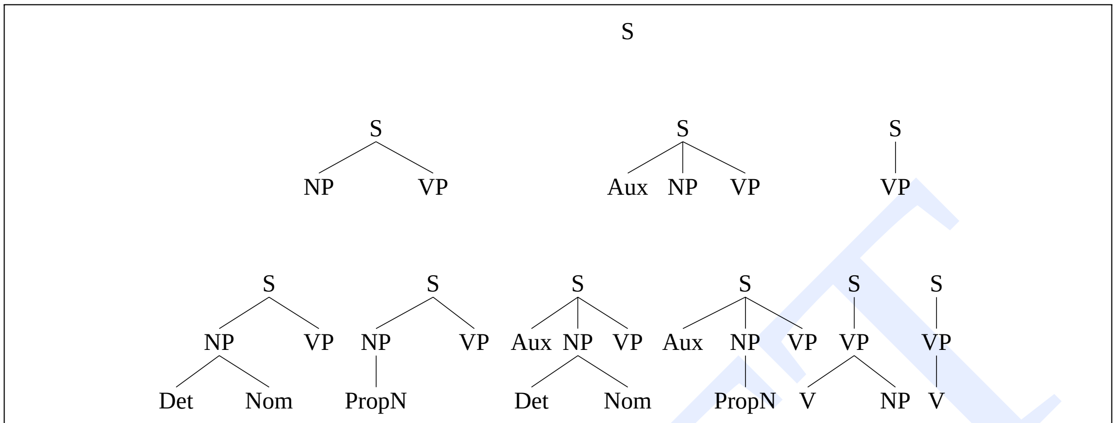


Figure 13.3 An expanding top-down search space. Each ply is created by taking each tree from the previous ply, replacing the leftmost non-terminal with each of its possible expansions, and collecting each of these trees into a new ply.

for the parser, which are then used to recursively generate the rest of the trees. Trees are grown downward until they eventually reach the part-of-speech categories at the bottom of the tree. At this point, trees whose leaves fail to match all the words in the input can be rejected, leaving behind those trees that represent successful parses. In Fig. 13.3, only the fifth parse tree in the third ply (the one which has expanded the rule *VP* → *Verb NP*) will eventually match the input sentence *Book that flight*.

13.1.2 Bottom-Up Parsing

BOTTOM-UP

Bottom-up parsing is the earliest known parsing algorithm (it was first suggested by Yngve (1955)), and is used in the shift-reduce parsers common for computer languages (Aho and Ullman, 1972). In bottom-up parsing, the parser starts with the words of the input, and tries to build trees from the words up, again by applying rules from the grammar one at a time. The parse is successful if the parser succeeds in building a tree rooted in the start symbol *S* that covers all of the input. Fig. 13.4 shows the bottom-up search space, beginning with the sentence *Book that flight*. The parser begins by looking up each input word in the lexicon and building three partial trees with the part-of-speech for each word. But the word *book* is ambiguous; it can be a noun or a verb. Thus the parser must consider two possible sets of trees. The first two plies in Fig. 13.4 show this initial bifurcation of the search space.

Each of the trees in the second ply is then expanded. In the parse on the left (the one in which *book* is incorrectly considered a noun), the *Nominal* → *Noun* rule is applied to both of the nouns (*book* and *flight*). This same rule is also applied to the sole noun (*flight*) on the right, producing the trees on the third ply.

In general, the parser extends one ply to the next by looking for places in the parse-in-progress where the right-hand side of some rule might fit. This contrasts with the earlier top-down parser, which expanded trees by applying rules when their left-hand side matched an unexpanded non-terminal.

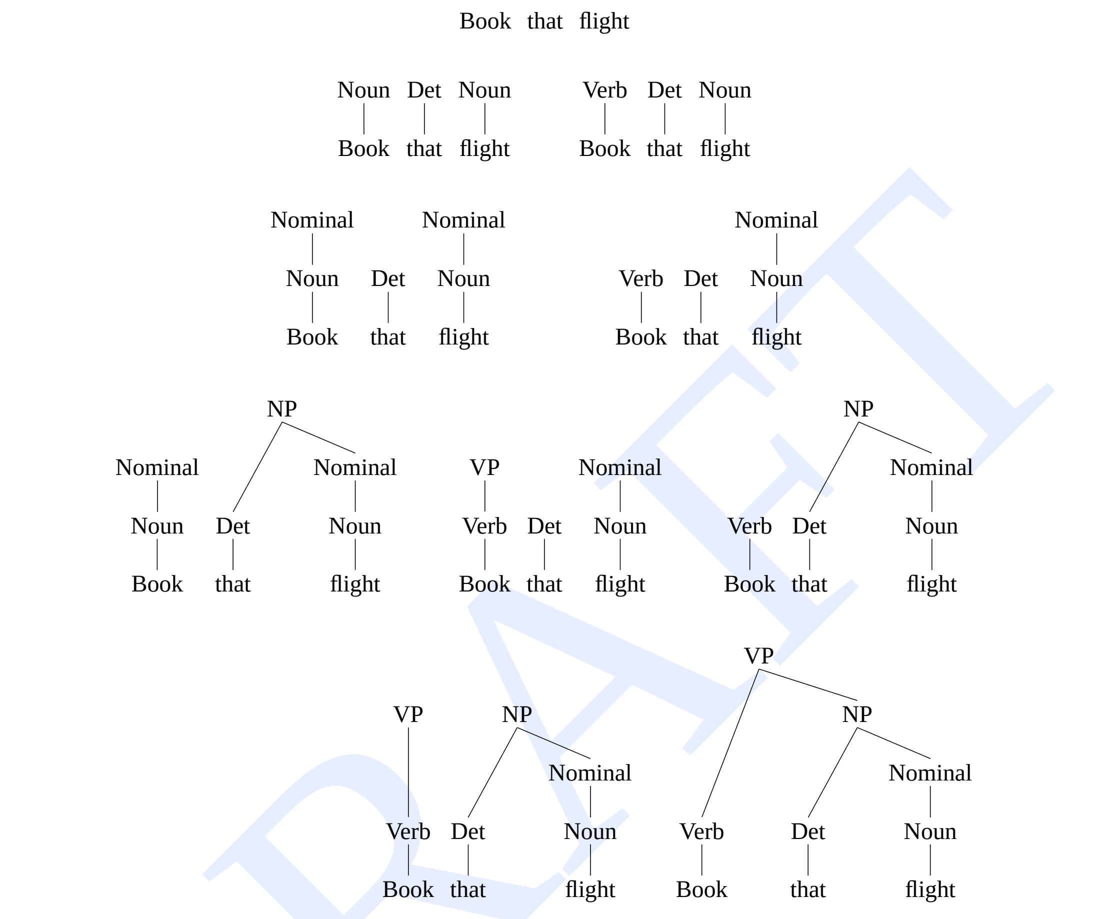


Figure 13.4 An expanding bottom-up search space for the sentence *Book that flight*. This figure does not show the final tier of the search with the correct parse tree (see Fig. 13.2). Make sure you understand how that final parse tree follows from the search space in this figure.

Thus in the fourth ply, in the first and third parse, the sequence *Det Nominal* is recognized as the right-hand side of the *NP* → *Det Nominal* rule.

In the fifth ply, the interpretation of *book* as a noun has been pruned from the search space. This is because this parse cannot be continued: there is no rule in the grammar with the right-hand side *Nominal NP*. The final ply of the search space (not shown in Fig. 13.4) contains the correct parse (see Fig. 13.2).

13.1.3 Comparing Top-Down and Bottom-Up Parsing

Each of these two architectures has its own advantages and disadvantages. The top-down strategy never wastes time exploring trees that cannot result in an *S*, since it

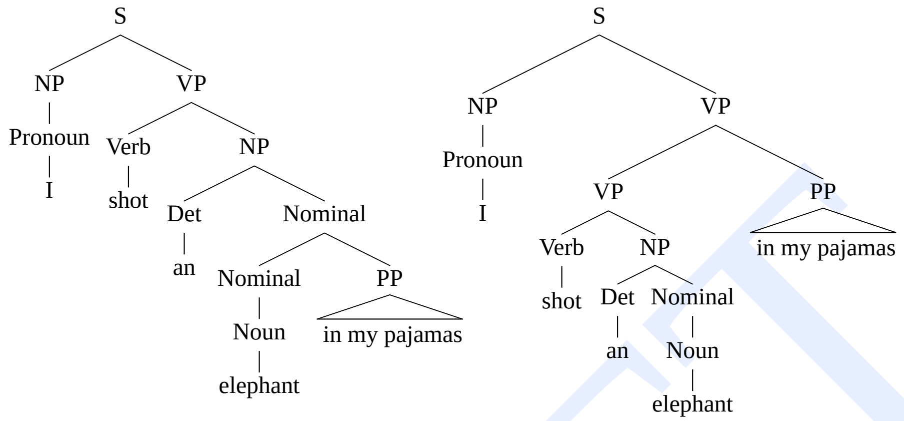


Figure 13.5 Two parse trees for an ambiguous sentence. Parse (a) corresponds to the humorous reading in which the elephant is in the pajamas, parse (b) to the reading in which Captain Spaulding did the shooting in his pajamas.

begins by generating just those trees. This means it also never explores subtrees that cannot find a place in some *S*-rooted tree. In the bottom-up strategy, by contrast, trees that have no hope of leading to an *S*, or fitting in with any of their neighbors, are generated with wild abandon.

The top-down approach has its own inefficiencies. While it does not waste time with trees that do not lead to an *S*, it does spend considerable effort on *S* trees that are not consistent with the input. Note that the first four of the six trees in the third ply in Fig. 13.3 all have left branches that cannot match the word *book*. None of these trees could possibly be used in parsing this sentence. This weakness in top-down parsers arises from the fact that they generate trees before ever examining the input. Bottom-up parsers, on the other hand, never suggest trees that are not at least locally grounded in the input.

13.2 Ambiguity

> *One morning I shot an elephant in my pajamas. How he got into my pajamas I don't know.*
>
> Groucho Marx, *Animal Crackers*, 1930

Ambiguity is perhaps the most serious problem faced by parsers. Ch. 5 introduced the notions of **part-of-speech ambiguity** and **part-of-speech disambiguation**. In this section we introduce a new kind of ambiguity, which arises in the syntactic structures used in parsing, called **structural ambiguity**. Structural ambiguity occurs when the grammar assigns more than one possible parse to a sentence. Groucho Marx's well-known line as Captain Spaulding is ambiguous because the phrase *in my pajamas* can be part of the *NP* headed by *elephant* or the verb-phrase headed by *shot*.

Structural ambiguity, appropriately enough, comes in many forms. Two particularly common kinds of ambiguity are **attachment ambiguity** and **coordination ambiguity**.

A sentence has an **attachment ambiguity** if a particular constituent can be attached to the parse tree at more than one place. The Groucho Marx sentence above is an example of PP-attachment ambiguity. Various kinds of adverbial phrases are also subject to this kind of ambiguity. For example in the following example the gerundive-VP *flying to Paris* can be part of a gerundive sentence whose subject is *the Eiffel Tower* or it can be an adjunct modifying the VP headed by *saw*:

(13.2) We saw the Eiffel Tower flying to Paris.

In **coordination ambiguity** there are different sets of phrases that can be conjoined by a conjunction like *and*. For example, the phrase *old men and women* can be bracketed as *[old [men and women]]*, referring to *old men* and *old women*, or as *[old men] and [women]*, in which case it is only the men who are old.

These ambiguities combine in complex ways in real sentences. A program that summarized the news, for example, would need to be able to parse sentences like the following from the Brown corpus:

(13.3) President Kennedy today pushed aside other White House business to devote all his time and attention to working on the Berlin crisis address he will deliver tomorrow night to the American people over nationwide television and radio.

This sentence has a number of ambiguities, although since they are semantically unreasonable, it requires a careful reading to see them. The last noun phrase could be parsed *[nationwide [television and radio]]* or *[[nationwide television] and radio]*. The direct object of *pushed aside* should be *other White House business* but could also be the bizarre phrase *[other White House business to devote all his time and attention to working]* (i.e., a structure like *Kennedy affirmed [his intention to propose a new budget to address the deficit]*). Then the phrase *on the Berlin crisis address he will deliver tomorrow night to the American people* could be an adjunct modifying the verb *pushed*. The *PP over nationwide television and radio* could be attached to any of the higher *VP*s or *NP*s (e.g., it could modify *people* or *night*).

SYNTACTIC DISAMBIGUATION

The fact that there are many unreasonable parses for naturally occurring sentences is an extremely irksome problem that affects all parsers. Ultimately, most natural language processing systems need to be able to choose the correct parse from the multitude of possible parses via process known as **syntactic disambiguation**. Unfortunately, effective disambiguation algorithms generally require statistical, semantic, and pragmatic knowledge not readily available during syntactic processing (techniques for making use of such knowledge will be introduced later, in Ch. 14 and Ch. 18).

Lacking such knowledge we are left with the choice of simply returning all the possible parse trees for a given input. Unfortunately, generating all the possible parses from robust, highly ambiguous, wide-coverage grammars such as the Penn Treebank grammar described in Ch. 12 is problematic. The reason for this lies in the potentially exponential number of parses that are possible for certain inputs. Consider the following ATIS example:

(13.4) Show me the meal on Flight UA 386 from San Francisco to Denver.

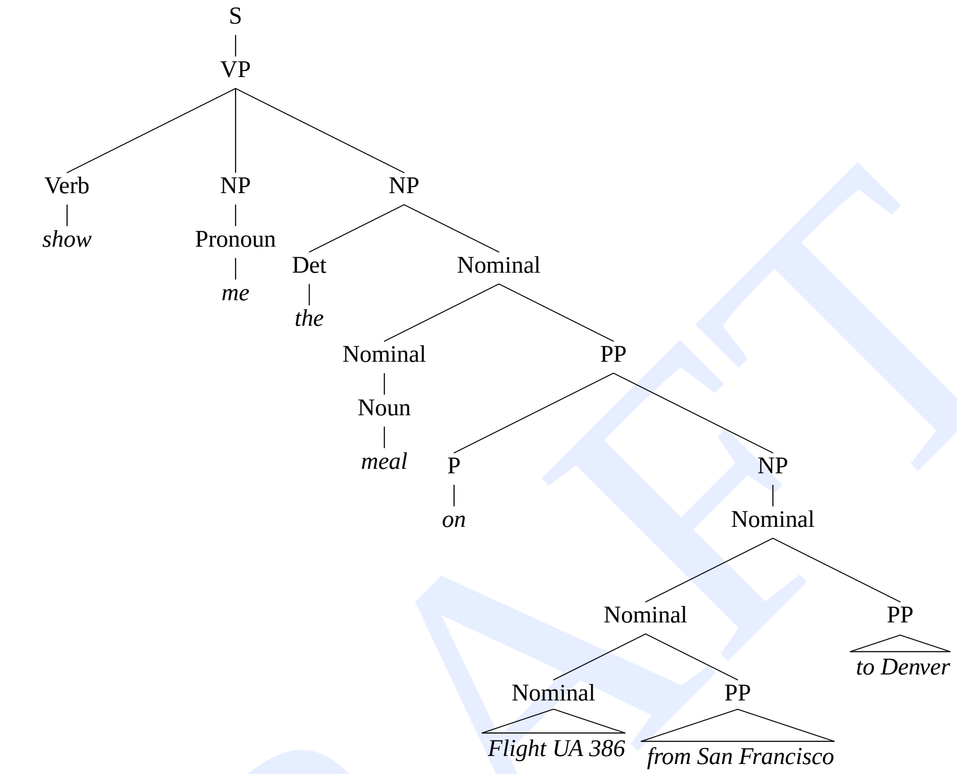


Figure 13.6 A reasonable parse for Ex. 13.4.

The recursive *VP* → *VP PP* and *Nominal* → *Nominal PP* rules conspire with the three prepositional phrases at the end of this sentence to yield a total of 14 parse trees for this sentence. For example *from San Francisco* could be part of the *VP* headed by *show* (which would have the bizarre interpretation that the showing was happening from San Francisco). Church and Patil (1982) showed that the number of parses for sentences of this type grows exponentially at the same rate as the number of parenthesizations of arithmetic expressions.

LOCAL AMBIGUITY

Even if a sentence isn't ambiguous (i.e. it doesn't have more than one parse in the end), it can be inefficient to parse due to **local ambiguity**. Local ambiguity occurs when some part of a sentence is ambiguous, that is, has more than one parse, even if the whole sentence is not ambiguous. For example the sentence *Book that flight* is unambiguous, but when the parser sees the first word *Book*, it cannot know if it is a verb or a noun until later. Thus it must use consider both possible parses.

13.3 Search in the Face of Ambiguity

To fully understand the problem that local and global ambiguity poses for syntactic parsing let's return to our earlier description of top-down and bottom-up parsing. There we made the simplifying assumption that we could explore all possible parse trees in parallel. Thus each ply of the search in Fig. 13.3 and Fig. 13.4 showed parallel expansions of the parse trees on the previous plies. Although it is certainly possible to implement this method directly, it typically entails the use of an unrealistic amount of memory to store the space of trees as they are being constructed. This is especially true since realistic grammars have much more ambiguity than the miniature grammar we've been using.

A common alternative approach to exploring complex search-spaces is to use an agenda-based backtracking **strategy** such as those used to implement the various finite-state machines in Chs. 2 and 3. A backtracking approach expands the search space incrementally by systematically exploring one state at a time. The state chosen for expansion can be based on simple systematic strategies such as depth-first or breadth-first methods, or on more complex methods that make use of probabilistic and semantic considerations. When the given strategy arrives at a tree that is inconsistent with the input, the search continues by returning to an unexplored option already on the agenda. The net effect of this strategy is a parser that single-mindedly pursues trees until they either succeed or fail before returning to work on trees generated earlier in the process.

Unfortunately, the pervasive ambiguity in typical grammars leads to intolerable inefficiencies in any backtracking approach. Backtracking parsers will often build valid trees for portions of the input, and then discard them during backtracking, only to find that they have to be rebuilt again. Consider the top-down backtracking process involved in finding a parse for the *NP* in (13.5):

(13.5) a flight from Indianapolis to Houston on TWA

The preferred complete parse shown as the bottom tree in Fig. 13.7. While there are numerous parses of this phrase, we will focus here on the amount of repeated work expended on the path to retrieving this single preferred parse.

A typical top-down, depth-first, left-to-right backtracking strategy leads to small parse trees that fail because they do not cover all of the input. These successive failures trigger backtracking events which lead to parses that incrementally cover more and more of the input. The sequence of trees attempted on the way to the correct parse by this top-down approach is shown in Fig. 13.7.

This figure clearly illustrates the kind of silly reduplication of work that arises in backtracking approaches. Except for its topmost component, every part of the final tree is derived more than once. The work done on this simple example would, of course, be magnified by any ambiguity introduced by the verb phrase or sentential level. Note that although this example is specific to top-down parsing, similar examples of wasted effort exist for bottom-up parsing as well.

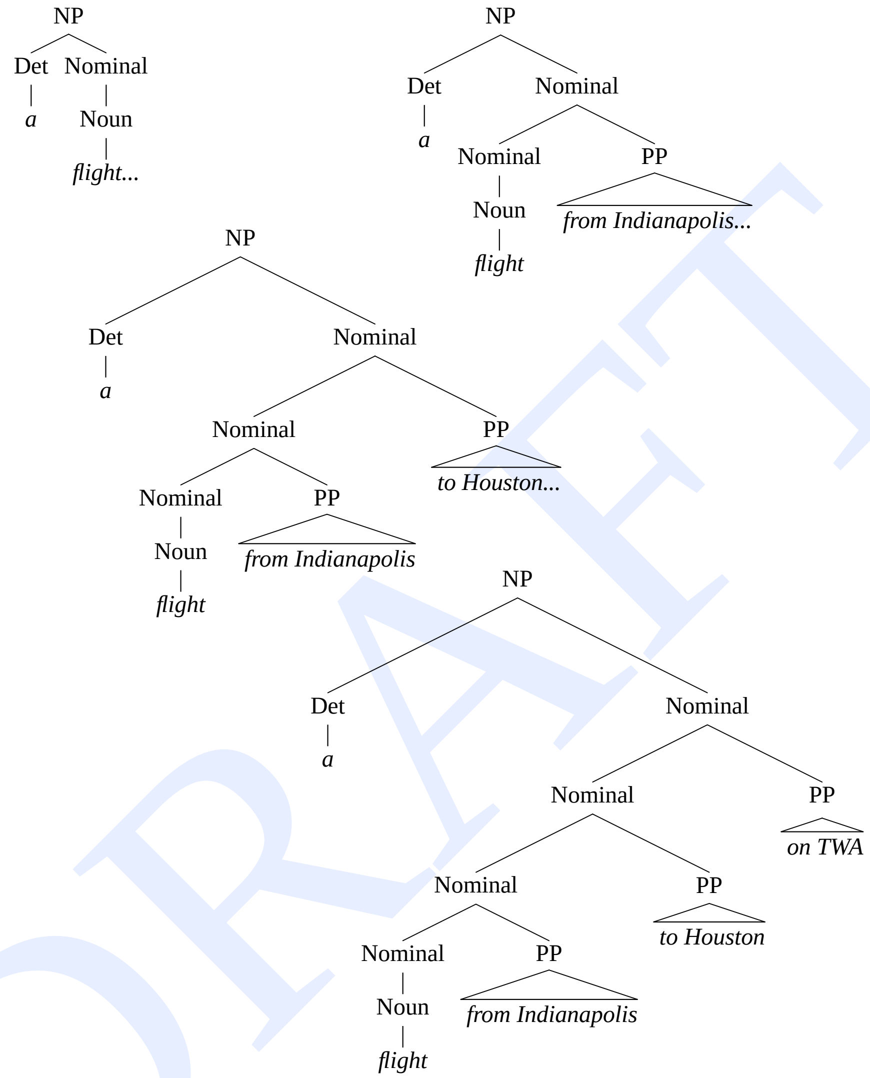


Figure 13.7 Reduplicated effort caused by backtracking in top-down parsing.

13.4 DYNAMIC PROGRAMMING PARSING METHODS

The previous section presented some of the problems that afflict standard bottom-up or top-down parsers due to ambiguity. Luckily, there is a single class of algorithms which can solve these problems. **Dynamic programming** once again provides a framework for solving this problem, just as it helped us with the Minimum Edit Distance, Viterbi, and Forward algorithms. Recall that dynamic programming approaches systematically fill in tables of solutions to sub-problems. When complete, the tables contain the solution to all the sub-problems needed to solve the problem as a whole.

In the case of parsing, such tables are used to store subtrees for each of the various constituents in the input as they are discovered. The efficiency gain arises from the fact that these subtrees are discovered once, stored, and then used in all parses calling for that constituent. This solves the re-parsing problem (subtrees are looked up, not re-parsed) and partially solves the ambiguity problem (the dynamic programming table implicitly stores all possible parses by storing all the constituents with links that enable the parses to be reconstructed). As we mentioned earlier, the three most widely used methods are the Cocke-Kasami-Younger (CKY) algorithm, the Earley algorithm, and Chart Parsing.

13.4.1 CKY Parsing

Let's begin our investigation of CKY algorithm by examining one of its major requirements: the grammars used with it must be in Chomsky Normal Form (CNF). Recall from Ch. 12 that grammars in CNF are restricted to rules of the form $A \rightarrow B\,C$, or $A \rightarrow w$. That is, the right-hand side of each rule must expand to either two non-terminals or to a single terminal. Recall also that restricting a grammar to CNF does not lead to any loss in expressiveness since any context-free grammar can be converted into a corresponding CNF grammar that accepts exactly the same set of strings as the original grammar. This single restriction gives rise to an extremely simple and elegant table-based parsing method.

Conversion to CNF

Let's start with the process of converting a generic CFG into one represented in CNF. Assuming we're dealing with an ε-free grammar, there are three situations we need to address in any generic grammar: rules that mix terminals with non-terminals on the right-hand side, rules that have a single non-terminal on the right, and rules where the right-hand side's length is greater than two.

The remediation for rules that mix terminals and non-terminals is to simply introduce a new dummy non-terminal that covers only the original terminal. For example, a rule for an infinitive verb phrase such as *INF-VP* → *to VP* would be replaced by the two rules *INF-VP* → *TO VP* and *TO* → *to*.

UNIT PRODUCTIONS

Rules with a single non-terminal on the right are called **unit productions**. Unit productions are eliminated by rewriting the right-hand side of the original rules with the right-hand side of all the non-unit production rules that they ultimately lead to. More formally, if $A \overset{*}{\Rightarrow} B$ by a chain of one or more unit productions, and $B \rightarrow \gamma$ is a non-unit production in our grammar, then we add $A \rightarrow \gamma$ for each such rule in the grammar, and discard all the intervening unit productions. As we'll see with our toy grammar, this can lead to a substantial *flattening* of the grammar, and a consequent promotion of terminals to fairly high levels in the resulting trees.

Rules with right-hand sides longer than 2 are remedied through the introduction of new non-terminals that spread the longer sequences over several new productions. Formally, if we have a rule like

$$A \rightarrow B\,C\,\gamma$$

we replace the leftmost pair of non-terminals with a new non-terminal and introduce a new production result in the following new rules.

$$X1 \rightarrow B\ C$$

$$A \rightarrow X1\ \gamma$$

In the case of longer right-hand sides, we simply iterate this process until the offending rule has length 2. The choice of replacing the leftmost pair of non-terminals is purely arbitrary; any systematic scheme that results in binary rules would suffice.

In our current grammar, the rule $S \rightarrow Aux\ NP\ VP$ would be replaced by the two rules $S \rightarrow X1\ VP$ and $X1 \rightarrow Aux\ NP$.

The entire conversion process can be summarized as follows:

1. Copy all conforming rules to the new grammar unchanged,
2. Convert terminals within rules to dummy non-terminals,
3. Convert unit-productions,
4. Binarize all rules and add to new grammar.

Fig. 13.8 shows the results of applying this entire conversion procedure to the $\mathscr{L}_1$ grammar introduced earlier on page 2. Note that this figure doesn't show the original lexical rules; since these original lexical rules are already in CNF, they all carry over unchanged to the new grammar. Fig. 13.8 does, however, show the various places where the process of eliminating unit-productions has, in effect, created new lexical rules. For example, all the original verbs have been promoted to both *VP*s and to *S*s in the converted grammar.

CKY Recognition

With our grammar now in CNF, each non-terminal node above the part-of-speech level in a parse tree will have exactly two daughters. A simple two-dimensional matrix can be used to encode the structure of an entire tree. More specifically, for a sentence of length n, we will be working with the upper-triangular portion of an $(n+1) \times (n+1)$ matrix. Each cell $[i, j]$ in this matrix contains a set of non-terminals that represent all the constituents that span positions i through j of the input. Since our indexing scheme begins with 0, it's natural to think of the indexes as pointing at the gaps between the input words (as in $_0$ *Book* $_1$ *that* $_2$ *flight* $_3$). It follows then that the cell that represents the entire input resides in position $[0, n]$ in the matrix.

Since our grammar is in CNF, the non-terminal entries in the table have exactly two daughters in the parse. Therefore, for each constituent represented by an entry $[i, j]$ in the table there must be a position in the input, k, where it can be split into two parts such that $i < k < j$. Given such a k, the first constituent $[i, k]$ must lie to the left of entry $[i, j]$ somewhere along row i, and the second entry $[k, j]$ must lie beneath it, along column j.

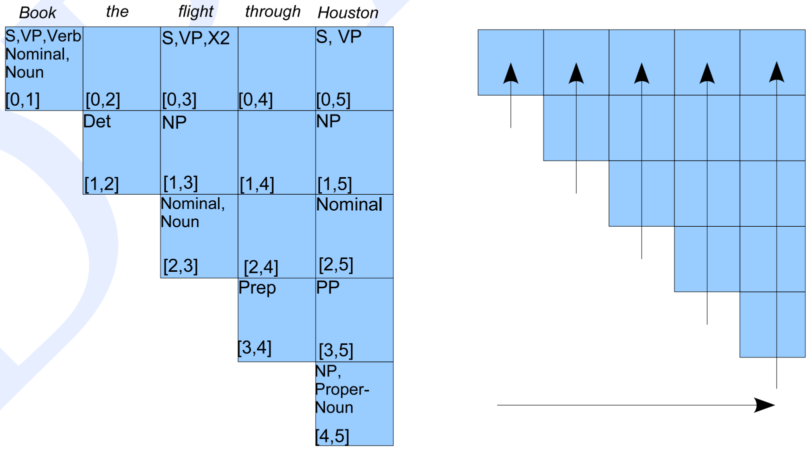
To make this more concrete, consider the following example with its completed parse matrix shown in Fig. 13.9.

(13.6) Book the flight through Houston.

The superdiagonal row in the matrix contains the parts of speech for each input word in the input. The subsequent diagonals above that superdiagonal contain constituents that cover all the spans of increasing length in the input.

S → *NP VP*	*S* → *NP VP*
S → *Aux NP VP*	*S* → *X1 VP*
	X1 → *Aux NP*
S → *VP*	*S* → *book* | *include* | *prefer*
	S → *Verb NP*
	S → *X2 PP*
	S → *Verb PP*
	S → *VP PP*
NP → *Pronoun*	*NP* → *I* | *she* | *me*
NP → *Proper-Noun*	*NP* → *TWA* | *Houston*
NP → *Det Nominal*	*NP* → *Det Nominal*
Nominal → *Noun*	*Nominal* → *book* | *flight* | *meal* | *money*
Nominal → *Nominal Noun*	*Nominal* → *Nominal Noun*
Nominal → *Nominal PP*	*Nominal* → *Nominal PP*
VP → *Verb*	*VP* → *book* | *include* | *prefer*
VP → *Verb NP*	*VP* → *Verb NP*
VP → *Verb NP PP*	*VP* → *X2 PP*
	X2 → *Verb NP*
VP → *Verb PP*	*VP* → *Verb PP*
VP → *VP PP*	*VP* → *VP PP*
PP → *Preposition NP*	*PP* → *Preposition NP*

Figure 13.8 $\mathscr{L}_1$ Grammar and its conversion to CNF. Note that although they aren't shown here all the original lexical entries from $\mathscr{L}_1$ carry over unchanged as well.



Book	*the*	*flight*	*through*	*Houston*
S,VP,Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S, VP [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

Figure 13.9 Completed parse table for *Book the flight through Houston.*

Given all this, CKY recognition is simply a matter of filling the parse table in the right way. To do this, we'll proceed in a bottom-up fashion so that at the point where we are filling any cell $[i, j]$, the cells containing the parts that could contribute to this entry, (i.e. the cells to the left and the cells below) have already been filled. There are several ways to do this; as the right side of Fig. 13.9 illustrates, the algorithm given in Fig. 13.10 fills the upper-triangular matrix a column at a time working from left to right. Each column is then filled from bottom to top. This scheme guarantees that at each point in time we have all the information we need (to the left, since all the columns to the left have already been filled, and below since we're filling bottom to top). It also mirrors on-line parsing since filling the columns from left to right corresponds to processing each word one at a time.

```
function CKY-PARSE(words, grammar) returns table

  for $j \leftarrow$ from 1 to LENGTH(words) do
     $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
     for $i \leftarrow$ from $j-2$ downto 0 do
        for $k \leftarrow i+1$ to $j-1$ do
           $table[i,j] \leftarrow table[i,j] \cup$
                         $\{A \mid A \rightarrow BC \in grammar,$
                            $B \in table[i,k],$
                            $C \in table[k,j]\}$
```

Figure 13.10 The CKY algorithm

The outermost loop of the algorithm given in Fig. 13.10 iterates over the columns, the second loop iterates over the rows, from the bottom up. The purpose of the innermost loop is to range over all the places where a substring spanning i to j in the input might be split in two. As k ranges over the places where the string can be split, the pairs of cells we consider move, in lockstep, to the right along row i and down along column j. Fig. 13.11 illustrates the general case of filling cell $[i, j]$. At each such split, the algorithm considers whether the contents of the two cells can be combined in a way that is sanctioned by a rule in the grammar. If such a rule exists, the non-terminal on its left-hand side is entered into the table.

Fig. 13.12 shows how the five cells of column 5 of the table are filled after the word *Houston* is read. The arrows point out the two spans that are being used to add an entry to the table. Note that the action in cell $[0,5]$ indicates the presence of three alternative parses for this input, one where the *PP* modifies the *flight*, one where it modifies the *booking*, and one that captures the second argument in the original *VP* → *Verb NP PP* rule, now captured indirectly with the *VP* → *X2 PP* rule.

In fact, since our current algorithm manipulates *sets* of non-terminals as cell entries, it won't include multiple copies of the same non-terminal in the table; the second *S* and *VP* discovered while processing $[0,5]$ would have no effect. We'll revisit this behavior in the next section.

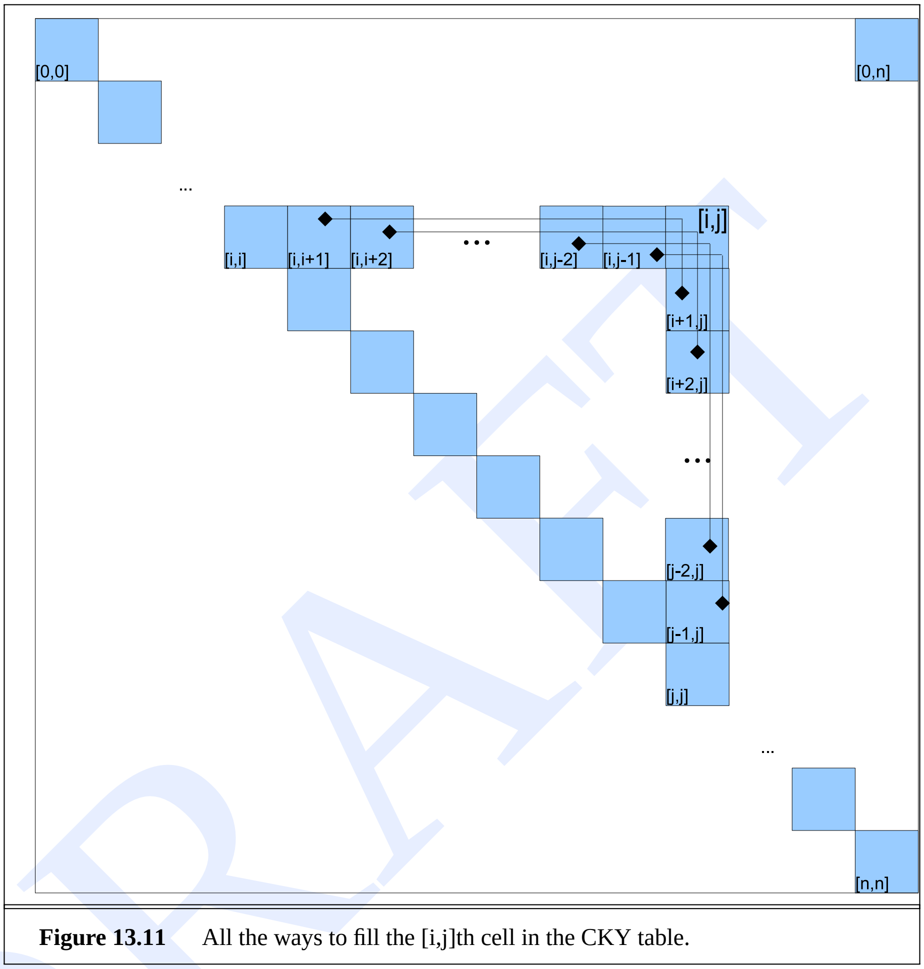


Figure 13.11 All the ways to fill the [i,j]th cell in the CKY table.

CKY Parsing

The algorithm given in Fig. 13.10 is a recognizer, not a parser; for it to succeed it simply has to find an S in cell $[0,N]$. To turn it into a parser capable of returning all possible parses for a given input, we'll make two simple changes to the algorithm: the first change is to augment the entries in the table so that each non-terminal is paired with pointers to the table entries from which it was derived (more or less as shown in Fig. 13.12), the second change is to permit multiple versions of the same non-terminal to be entered into the table (again as shown in Fig. 13.12.) With these changes, the completed table contains all the possible parses for a given input. Returning an arbitrary single parse consists of choosing an S from cell $[0,n]$ and then recursively retrieving its component constituents from the table.

Of course, returning all the parses for a given input may incur considerable cost. As we saw earlier, there may be an exponential number of parses associated with a given input. In such cases, returning all the parses will have an unavoidable exponential

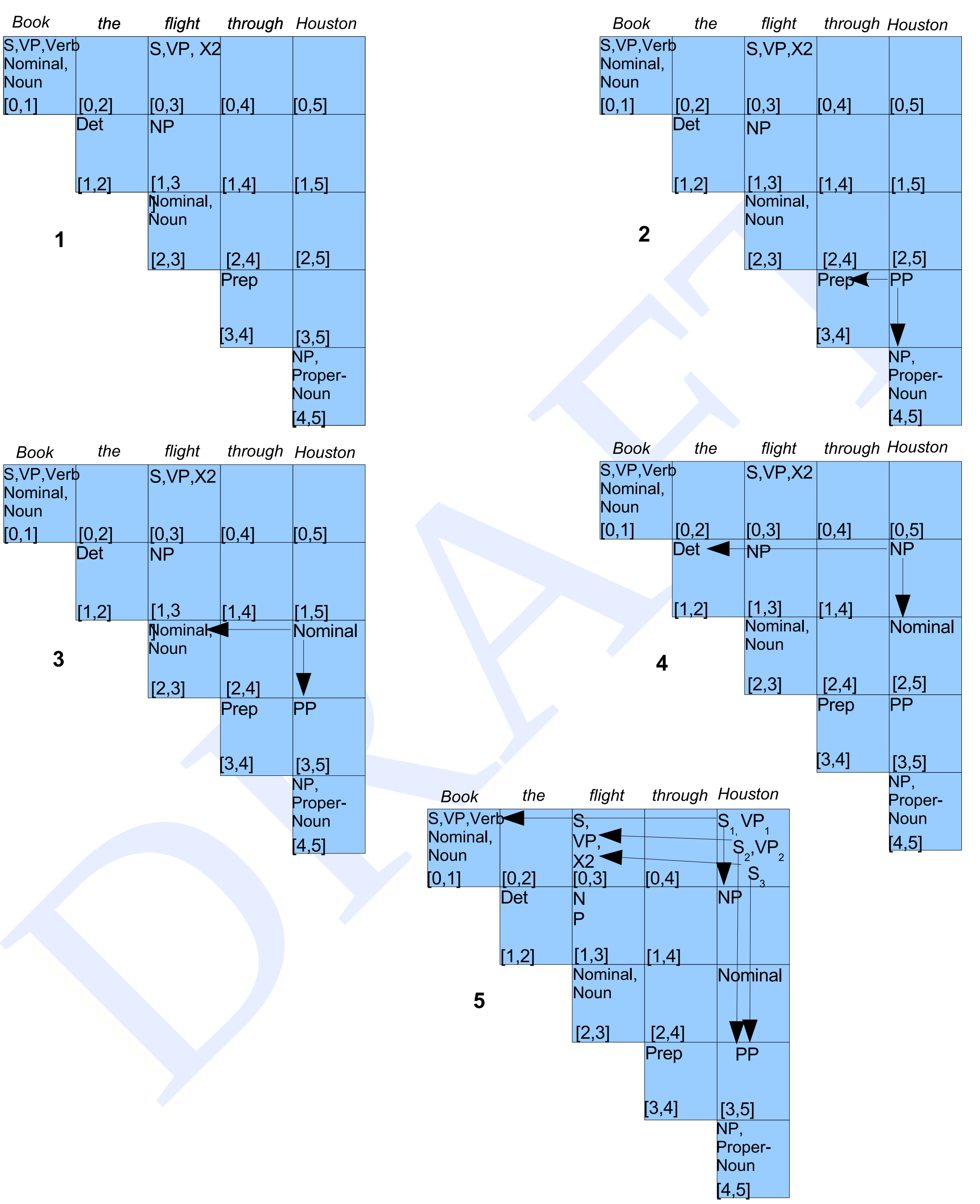


Figure 13.12 Filling the last column after reading the word *Houston*.

cost. Looking forward to Ch. 14, we can also think about retrieving the best parse for a given input by further augmenting the table to contain the probabilities of each entry. Retrieving the most probable parse consists of running a suitably modified version of the Viterbi algorithm from Ch. 5 over the completed parse table.

CKY in Practice

Finally, we should note that while the restriction to CNF does not pose a problem theoretically, it does pose some non-trivial problems in practice. Obviously, as things stand now, our parser isn't returning trees that are consistent with the grammar given to us by our friendly syntacticians. In addition to making our grammar developers unhappy, the conversion to CNF will complicate any syntax-driven approach to semantic analysis.

One approach to getting around these problems is to keep enough information around to transform our trees back to the original grammar as a post-processing step of the parse. This is trivial in the case of the transformation used for rules with length greater than 2. Simply deleting the new dummy non-terminals and promoting their daughters restores the original tree.

In the case of unit productions, it turns out to be more convenient to alter the basic CKY algorithm to handle them directly than it is to store the information needed to recover the correct trees. Exercise 13.3 asks you to make this change. Many of the probabilistic parsers presented in Ch. 14 use the CKY algorithm altered in just this manner. Another solution is to adopt a more complex dynamic programming solution that simply accepts arbitrary CFGs. The next section presents such an approach.

13.4.2 The Earley Algorithm

In contrast to the bottom-up search implemented by the CKY algorithm, the Earley algorithm (Earley, 1970) uses dynamic programming to implement a **top-down search** of the kind discussed earlier in Sec. 13.1.1. The core of the Earley algorithm is a single left-to-right pass that fills an array we'll call a **chart** that has $N+1$ entries. For each word position in the sentence, the chart contains a list of states representing the partial parse trees that have been generated so far. As with the CKY algorithm, the indexes represent the locations between the words in an input (as in $_0$ *Book* $_1$ *that* $_2$ *flight* $_3$). By the end of the sentence, the chart compactly encodes all the possible parses of the input. Each possible subtree is represented only once and can thus be shared by all the parses that need it.

CHART

The individual states contained within each chart entry contain three kinds of information: a subtree corresponding to a single grammar rule, information about the progress made in completing this subtree, and the position of the subtree with respect to the input. We'll use a • within the right-hand side of a state's grammar rule to indicate the progress made in recognizing it. The resulting structure is called a **dotted rule**. A state's position with respect to the input will be represented by two numbers indicating where the state begins and where its dot lies.

DOTTED RULE

Consider the following example states, which would be among those created by the Earley algorithm in the course of parsing Ex. 13.7:

(13.7) Book that flight.

$$\begin{aligned} S &\rightarrow \bullet\, VP,\ [0,0] \\ NP &\rightarrow Det \bullet Nominal,\ [1,2] \\ VP &\rightarrow V\ NP \bullet,\ [0,3] \end{aligned}$$

The first state, with its dot to the left of its constituent, represents a top-down prediction for this particular kind of S. The first 0 indicates that the constituent predicted by this state should begin at the start of the input; the second 0 reflects the fact that the dot lies at the beginning as well. The second state, created at a later stage in the processing of this sentence, indicates that an *NP* begins at position 1, that a *Det* has been successfully parsed and that a *Nominal* is expected next. The third state, with its dot to the right of all its two constituents, represents the successful discovery of a tree corresponding to a *VP* that spans the entire input.

The fundamental operation of an Earley parser is to march through the $N+1$ sets of states in the chart in a left-to-right fashion, processing the states within each set in order. At each step, one of the three operators described below is applied to each state depending on its status. In each case, this results in the addition of new states to the end of either the current, or next, set of states in the chart. The algorithm always moves forward through the chart making additions as it goes; states are never removed and the algorithm never backtracks to a previous chart entry once it has moved on. The presence of a state $S \rightarrow \alpha \bullet,\ [0,N]$ in the list of states in the last chart entry indicates a successful parse. Fig. 13.13 gives the complete algorithm.

The following three sections describe in detail the three operators used to process states in the chart. Each takes a single state as input and derives new states from it. These new states are then added to the chart as long as they are not already present. The PREDICTOR and the COMPLETER add states to the chart entry being processed, while the SCANNER adds a state to the next chart entry.

Predictor

As might be guessed from its name, the job of PREDICTOR is to create new states representing top-down expectations generated during the parsing process. PREDICTOR is applied to any state that has a non-terminal immediately to the right of its dot that is not a part-of-speech category. This application results in the creation of one new state for each alternative expansion of that non-terminal provided by the grammar. These new states are placed into the same chart entry as the generating state. They begin and end at the point in the input where the generating state ends.

For example, applying PREDICTOR to the state $S \rightarrow \bullet\, VP,\ [0,0]$ results in the addition of the following five states

$$\begin{aligned} VP &\rightarrow \bullet\, Verb,\ [0,0] \\ VP &\rightarrow \bullet\, Verb\ NP,\ [0,0] \\ VP &\rightarrow \bullet\, Verb\ NP\ PP,\ [0,0] \\ VP &\rightarrow \bullet\, Verb\ PP,\ [0,0] \\ VP &\rightarrow \bullet\, VP\ PP,\ [0,0] \end{aligned}$$

to the first chart entry.

```
function Earley-Parse(words, grammar) returns chart

  AddToChart(($\gamma \rightarrow \bullet S$, $[0,0]$), chart[0])
  for $i \leftarrow$ from 0 to Length(words) do
    for each state in chart[$i$] do
      if Incomplete?(state) and
             Next-Cat(state) is not a part of speech then
          Predictor(state)
      elseif Incomplete?(state) and
             Next-Cat(state) is a part of speech then
          Scanner(state)
      else
          Completer(state)
    end
  end
  return(chart)

 procedure Predictor(($A \rightarrow \alpha \bullet B\,\beta$, $[i,j]$))
    for each ($B \rightarrow \gamma$) in Grammar-Rules-For(B, grammar) do
        AddToChart(($B \rightarrow \bullet\,\gamma$, $[j,j]$), chart[$j$])
    end

 procedure Scanner(($A \rightarrow \alpha \bullet B\,\beta$, $[i,j]$))
    if $B \in$ Parts-of-Speech(word[j]) then
        AddToChart(($B \rightarrow$ word[$j$] $\bullet$, $[j,j+1]$), chart[j+1])

 procedure Completer(($B \rightarrow \gamma \bullet$, $[j,k]$))
    for each ($A \rightarrow \alpha \bullet B\,\beta$, $[i,j]$) in chart[$j$] do
        AddToChart(($A \rightarrow \alpha\, B \bullet \beta$, $[i,k]$), chart[k])
    end

 procedure AddToChart(state, chart-entry)
    if state is not already in chart-entry then
        Push-On-End(state, chart-entry)
    end
```

Figure 13.13 The Earley algorithm

Scanner

When a state has a part-of-speech category to the right of the dot, Scanner is called to examine the input and incorporate a state corresponding to the prediction of a word with a particular part-of-speech into the chart. This is accomplished by creating a new state from the input state with the dot advanced over the predicted input category. Note that unlike CKY, Earley uses top-down input to help deal with part-of-speech ambiguities; only those parts-of-speech of a word that are predicted by some existing state will find their way into the chart.

Returning to our example, when the state $VP \rightarrow \bullet\, Verb\ NP$, $[0,0]$ is processed, Scanner consults the current word in the input since the category following the dot is a part-of-speech. It then notes that *book* can be a verb, matching the expectation in the current state. This results in the creation of the new state $Verb \rightarrow book \bullet, [0,1]$. This new state is then added to the chart entry that *follows* the one currently being processed. The noun sense of *book* never enters the chart since it is not predicted by any rule at this position in the input.

Completer

COMPLETER is applied to a state when its dot has reached the right end of the rule. The presence of such a state represents the fact that the parser has successfully discovered a particular grammatical category over some span of the input. The purpose of COMPLETER is to find, and advance, all previously created states that were looking for this grammatical category at this position in the input. New states are then created by **copying** the older state, advancing the dot over the expected category, and installing the new state in the current chart entry.

In the current example, when the state $NP \rightarrow Det\ Nominal \bullet, [1,3]$ is processed, COMPLETER looks for incomplete states ending at position 1 and expecting an *NP*. It finds the states $VP \rightarrow Verb \bullet NP, [0,1]$ and $VP \rightarrow Verb \bullet NP\ PP, [0,1]$. This results in the addition of the new complete state $VP \rightarrow Verb\ NP \bullet, [0,3]$, and the new incomplete state $VP \rightarrow Verb\ NP \bullet PP, [0,3]$ to the chart.

A Complete Example

Fig. 13.14 shows the sequence of states created during the complete processing of Ex. 13.7; each row indicates the state number for reference, the dotted rule, the start and end points, and finally the function that added this state to the chart. The algorithm begins by seeding the chart with a top-down expectation for an *S*. This is accomplished by adding a dummy state $\gamma \rightarrow \bullet S, [0,0]$ to Chart[0]. When this state is processed, it is passed to PREDICTOR leading to the creation of the three states representing predictions for each possible type of *S*, and transitively to states for all of the left-corners of those trees. When the state $VP \rightarrow \bullet Verb, [0,0]$ is reached, SCANNER is called and the first word is read. A state representing the verb sense of *Book* is added to the entry for Chart[1]. Note that when the subsequent sentence initial *VP* states are processed, SCANNER will be called again. However, new states are not added since they would be identical to the *Verb* state already in the chart.

When all the states of Chart[0] have been processed, the algorithm moves on to Chart[1] where it finds the state representing the verb sense of *book*. This is a complete state with its dot to the right of its constituent and is therefore passed to COMPLETER. COMPLETER then finds the four previously existing *VP* states expecting a *V*erb at this point in the input. These states are copied with their dots advanced and added to Chart[1]. The completed state corresponding to an intransitive *VP* then leads to the creation of an *S* representing an imperative sentence. Alternatively, the dot in the transitive verb phrase leads to the creation of the three states predicting different forms of *NP*s. The state $NP \rightarrow \bullet Det\ Nominal, [1,1]$ causes SCANNER to read the word *that* and add a corresponding state to Chart[2].

Moving on to Chart[2], the algorithm finds the state representing the determiner sense of *that*. This complete state leads to the advancement of the dot in the *NP* state

Chart[0]	S0	$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
	S1	$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
	S2	$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
	S3	$S \rightarrow \bullet VP$	[0,0]	Predictor
	S4	$NP \rightarrow \bullet Pronoun$	[0,0]	Predictor
	S5	$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
	S6	$NP \rightarrow \bullet Det\ Nominal$	[0,0]	Predictor
	S7	$VP \rightarrow \bullet Verb$	[0,0]	Predictor
	S8	$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor
	S9	$VP \rightarrow \bullet Verb\ NP\ PP$	[0,0]	Predictor
	S10	$VP \rightarrow \bullet Verb\ PP$	[0,0]	Predictor
	S11	$VP \rightarrow \bullet VP\ PP$	[0,0]	Predictor
Chart[1]	S12	$Verb \rightarrow book \bullet$	[0,1]	Scanner
	S13	$VP \rightarrow Verb \bullet$	[0,1]	Completer
	S14	$VP \rightarrow Verb \bullet NP$	[0,1]	Completer
	S15	$VP \rightarrow Verb \bullet NP\ PP$	[0,1]	Completer
	S16	$VP \rightarrow Verb \bullet PP$	[0,1]	Completer
	S17	$S \rightarrow VP \bullet$	[0,1]	Completer
	S18	$VP \rightarrow VP \bullet PP$	[0,1]	Completer
	S19	$NP \rightarrow \bullet Pronoun$	[1,1]	Predictor
	S20	$NP \rightarrow \bullet Proper\text{-}Noun$	[1,1]	Predictor
	S21	$NP \rightarrow \bullet Det\ Nominal$	[1,1]	Predictor
	S22	$PP \rightarrow \bullet Prep\ NP$	[1,1]	Predictor
Chart[2]	S23	$Det \rightarrow that \bullet$	[1,2]	Scanner
	S24	$NP \rightarrow Det \bullet Nominal$	[1,2]	Completer
	S25	$Nominal \rightarrow \bullet Noun$	[2,2]	Predictor
	S26	$Nominal \rightarrow \bullet Nominal\ Noun$	[2,2]	Predictor
	S27	$Nominal \rightarrow \bullet Nominal\ PP$	[2,2]	Predictor
Chart[3]	S28	$Noun \rightarrow flight \bullet$	[2,3]	Scanner
	S29	$Nominal \rightarrow Noun \bullet$	[2,3]	Completer
	S30	$NP \rightarrow Det\ Nominal \bullet$	[1,3]	Completer
	S31	$Nominal \rightarrow Nominal \bullet Noun$	[2,3]	Completer
	S32	$Nominal \rightarrow Nominal \bullet PP$	[2,3]	Completer
	S33	$VP \rightarrow Verb\ NP \bullet$	[0,3]	Completer
	S34	$VP \rightarrow Verb\ NP \bullet PP$	[0,3]	Completer
	S35	$PP \rightarrow \bullet Prep\ NP$	[3,3]	Predictor
	S36	$S \rightarrow VP \bullet$	[0,3]	Completer
	S37	$VP \rightarrow VP \bullet PP$	[0,3]	Completer

Figure 13.14 Chart entries created during an Earley parse of *Book that flight.* Each entry shows the state, its start and end points, and the function that placed it in the chart.

predicted in Chart[1], and also to the predictions for the various kinds of *Nominal*. The first of these causes SCANNER to be called for the last time to process the word *flight*.

Finally moving on to Chart[3], the presence of the state representing *flight* leads in quick succession to the completion of an *NP*, transitive *VP*, and an *S*. The presence of the state $S \rightarrow VP\bullet,\ [0,3]$ in the last chart entry signals the discovery of a successful

Chart[1]	S12	*Verb* $\rightarrow$ *book* $\bullet$	[0,1]	Scanner
Chart[2]	S23	*Det* $\rightarrow$ *that* $\bullet$	[1,2]	Scanner
Chart[3]	S28	*Noun* $\rightarrow$ *flight* $\bullet$	[2,3]	Scanner
	S29	*Nominal* $\rightarrow$ *Noun* $\bullet$	[2,3]	(S28)
	S30	*NP* $\rightarrow$ *Det Nominal* $\bullet$	[1,3]	(S23, S29)
	S33	*VP* $\rightarrow$ *Verb NP* $\bullet$	[0,3]	(S12, S30)
	S36	*S* $\rightarrow$ *VP* $\bullet$	[0,3]	(S33)

Figure 13.15 States that participate in the final parse of *Book that flight*, including structural parse information.

parse.

It is useful to contrast this example with the CKY example given earlier. Although Earley managed to avoid adding an entry for the noun sense of *book*, its overall behavior is clearly much more promiscuous than CKY. This promiscuity arises from the purely top-down nature of the predictions that Earley makes. Exercise 13.6 asks you to improve the algorithm by eliminating some of these unnecessary predictions.

Retrieving Parse Trees from a Chart

As with the CKY algorithm, this version of the Earley algorithm is a recognizer not a parser. Valid sentences will simply leave the state $S \rightarrow \alpha\bullet, [0,N]$ in the chart. To retrieve parses from the chart the representation of each state must be augmented with an additional field to store information about the completed states that generated its constituents.

The information needed to fill these fields can be gathered by making a simple change to the COMPLETER function. Recall that COMPLETER creates new states by advancing existing incomplete states when the constituent following the dot has been discovered in the right place. The only change necessary is to have COMPLETER add a pointer to the older state onto a list of constituent-states for the new state. Retrieving a parse tree from the chart is then merely a matter of following pointers starting with the state (or states) representing a complete S in the final chart entry. Fig. 13.15 shows the chart entries produced by an appropriately updated COMPLETER that participate in the final parse for this example.

13.4.3 Chart Parsing

CHART PARSING

In both the CKY and Earley algorithms, the order in which events occur (adding entries to the table, reading words, making predictions, etc.) is statically determined by the procedures that make up these algorithms. Unfortunately, dynamically determining the order in which events occur based on the current information is often necessary for a variety of reasons. Fortunately, an approach advanced by Martin Kay and his colleagues (Kaplan, 1973; Kay, 1986) called **Chart Parsing** facilitates just such dynamic determination of the order in which chart entries are processed. This is accomplished through the introduction of an *agenda* to the mix. In this scheme, as states (called **edges** in this approach) are created they are added to an agenda that is kept ordered according to a policy that is specified *separately* from the main parsing algorithm. This can be viewed as another instance of state-space search that we've seen several times before. The FSA and FST recognition and parsing algorithms in Chs. 2 and 3 employed agendas with simple static policies, while the A^* decoding algorithm described in Ch. 9 is driven by an agenda that is ordered probabilistically.

Fig. 13.16 presents a generic version of a parser based on such a scheme. The main part of the algorithm consists of a single loop that removes a edge from the front of an agenda, processes it, and then moves on to the next entry in the agenda. When the agenda is empty, the parser stops and returns the chart. The policy used to order the elements in the agenda thus determines the order in which further edges are created and predictions are made.

```
function CHART-PARSE(words, grammar, agenda-strategy) returns chart

  INITIALIZE(chart, agenda, words)
  while agenda
      current-edge ← POP(agenda)
      PROCESS-EDGE(current-edge)
  return(chart)

procedure PROCESS-EDGE(edge)
    ADD-TO-CHART(edge)
    if INCOMPLETE?(edge)
        FORWARD-FUNDAMENTAL-RULE(edge)
    else
        BACKWARD-FUNDAMENTAL-RULE(edge)
    MAKE-PREDICTIONS(edge)

procedure FORWARD-FUNDAMENTAL(($A \rightarrow \alpha \bullet B\ \beta,\ [i,j]$))
  for each($B \rightarrow \gamma \bullet,\ [j,k]$) in chart
      ADD-TO-AGENDA($A \rightarrow \alpha\ B \bullet \beta,\ [i,k]$)

procedure BACKWARD-FUNDAMENTAL(($B \rightarrow \gamma \bullet,\ [j,k]$))
  for each($A \rightarrow \alpha \bullet B\ \beta,\ [i,j]$) in chart
      ADD-TO-AGENDA($A \rightarrow \alpha\ B \bullet \beta,\ [i,k]$)

procedure ADD-TO-CHART(edge)
  if edge is not already in chart then
      Add edge to chart

procedure ADD-TO-AGENDA(edge)
  if edge is not already in agenda then
      APPLY(agenda-strategy, edge, agenda)
```

Figure 13.16 A Chart Parsing Algorithm

FUNDAMENTAL RULE

The key principle in processing edges in this approach is what Kay termed the **fundamental rule** of chart parsing. The fundamental rule states that when the chart contains two contiguous edges where one of the edges provides the constituent that the other one needs, a new edge should be created that spans the original edges and incorporates the provided material. More formally, the fundamental rule states the following: if the chart contains two edges $A \rightarrow \alpha \bullet B\,\beta, [i,j]$ and $B \rightarrow \gamma \bullet, [j,k]$ then we should add the new edge $A \rightarrow \alpha\, B \bullet \beta\, [i,k]$ to the chart. It should be clear that the fundamental rule is a generalization of the basic table-filling operations found in both the CKY and Earley algorithms.

The fundamental rule is triggered in Fig. 13.16 when an edge is removed from the agenda and passed to the PROCESS-EDGE procedure. Note that the fundamental rule itself does not specify which of the two edges involved has triggered the processing. PROCESS-EDGE handles both cases by checking to see whether or not the edge in question is complete. If it is complete than the algorithm looks earlier in the chart to see if any existing edge can be advanced; if it is incomplete than it looks later in the chart to see if it can be advanced by any pre-existing edge later in the chart.

The next piece of the algorithm that needs to be filled in is the method for making predictions based on the edge being processed. There are two key components to making predictions in chart parsing: the events that trigger predictions, and the nature of a predictions. The nature of these components varies depending on whether we are pursuing a top-down or bottom-up strategy. As in Earley, top-down predictions are triggered by expectations that arise from incomplete edges that have been entered into the chart; bottom-up predictions are triggered by the discovery of completed constituents. Fig. 13.17 illustrates how these two strategies can be integrated into the chart parsing algorithm.

```
procedure MAKE-PREDICTIONS(edge)
  if Top-Down and INCOMPLETE?(edge)
     TD-PREDICT(edge)
  elsif Bottom-Up and COMPLETE?(edge)
     BU-PREDICT(edge)

procedure TD-PREDICT(($A \rightarrow \alpha \bullet B\,\beta$, $[i,j]$))
  for each($B \rightarrow \gamma$) in grammar do
      ADD-TO-AGENDA($B \rightarrow \bullet\,\gamma$, $[j,j]$)

procedure BU-PREDICT(($B \rightarrow \gamma \bullet$, $[i,j]$))
  for each($A \rightarrow B\,\beta$) in grammar
      ADD-TO-AGENDA($A \rightarrow B \bullet \beta$, $[i,j]$)
```

Figure 13.17 A Chart Parsing Algorithm

Obviously we've left out many of the bookkeeping details that would have to be specified to turn this approach into a real parser. Among the details that have to be worked out are how the INITIALIZE procedure gets things started, how and when words are read, the organization of the chart, and specifying an agenda strategy. Indeed, in describing the approach here, Kay (1986) refers to it as an **algorithm!schema** rather than an algorithm, since it more accurately specifies an entire family of parsers rather than any particular parser. Exercise 13.7 asks you to explore some of the available

ALGORITHM!SCHEMA

choices by implementing various chart parsers.

13.5 PARTIAL PARSING

PARTIAL PARSE

SHALLOW PARSE

Many language-processing tasks simply do not require complex, complete parse trees for all inputs. For these tasks, a **partial parse**, or **shallow parse**, of input sentences may be sufficient. For example, **information extraction** systems generally do not extract *all* the possible information from a text; they simply identify and classify the segments in a text that are likely to contain valuable information. Similarly, information retrieval systems may choose to index documents based on a select subset of the constituents found in a text.

Not surprisingly, there are many different approaches to partial parsing. Some approaches make use of cascades of FSTs, of the kind discussed in Ch. 3, to to produce representations that closely approximate the kinds of trees we've been assuming in this chapter and the last. These approaches typically produce flatter trees than the ones we've been discussing. This flatness arises from the fact that such approaches generally defer decisions that may require semantic or contextual factors, such as prepositional phrase attachments, coordination ambiguities, and nominal compound analyses. Nevertheless the intent is to produce parse-trees that link all the major constituents in an input.

CHUNKING

An alternative style of partial parsing is known as **chunking**. Chunking is the process of identifying and classifying the flat non-overlapping segments of a sentence that constitute the basic non-recursive phrases corresponding to the major parts-of-speech found in most wide-coverage grammars. This set typically includes noun phrases, verb phrases, adjective phrases, and prepositional phrases; in other words, the phrases that correspond to the content-bearing parts-of-speech. Of course, not all applications require the identification of all of these categories; indeed the most common chunking task is to simply find all the base noun phrases in a text.

Since chunked texts lack a hierarchical structure, a simple bracketing notation is sufficient to denote the location and the type of the chunks in a given example. The following example illustrates a typical bracketed notation.

(13.8) [$_{NP}$ The morning flight] [$_{PP}$ from] [$_{NP}$ Denver] [$_{VP}$ has arrived.]

This bracketing notation makes clear the two fundamental tasks that are involved in chunking: finding the non-overlapping extents of the chunks, and assigning the correct label to the discovered chunks.

Note that in this example all the words are contained in some chunk. This will not be the case in all chunking applications. In many settings, a good number of the words in any input will fall outside of any chunk. This is, for example, the norm in systems that are only interested in finding the base-NPs in their inputs, as illustrated by the following example.

(13.9) [$_{NP}$ The morning flight] from [$_{NP}$ Denver] has arrived.

The details of what constitutes a syntactic base-phrase for any given system varies according to the syntactic theories underlying the system and whether the phrases are being derived from a treebank. Nevertheless, some standard guidelines are followed in most systems. First and foremost, base phrases of a given type do not recursively contain any constituents of the same type. Eliminating this kind of recursion leaves us with the problem of determining the boundaries of the non-recursive phrases. In most approaches, base-phrases include the headword of the phrase, along with any pre-head material within the constituent, while crucially excluding any post-head material. Eliminating post-head modifiers from the major categories automatically removes the need to resolve attachment ambiguities. Note that exclusion does lead to certain oddities such as the fact that *PP*s and *VP*s often consist solely of their heads. Thus our earlier example *a flight from Indianapolis to Houston on TWA* is reduced to the following:

(13.10) $[_{NP}$ a flight] $[_{PP}$ from] $[_{NP}$ Indianapolis]$[_{PP}$ to]$[_{NP}$ Houston]$[_{PP}$ on]$[_{NP}$ TWA].

13.5.1 Finite-State Rule-Based Chunking

Syntactic base-phrases of the kind we're considering can be characterized by finite-state automata (or finite-state rules, or regular expressions) of the kind discussed earlier in Chs. 2 and 3. In finite-state rule-based chunking, a set of rules is hand-crafted to capture the phrases of interest for any particular application. In most rule-based systems, chunking proceeds from left-to-right, finding the longest matching chunk from the beginning of the sentence, it then continues with the first word after the end of the previously recognized chunk. The process continues until the end of the sentence. This is a greedy process and is not guaranteed to find the best global analysis for any given input.

The primary limitation placed on these chunk rules is that they can not contain any recursion; the right-hand side of the rule can not reference directly, or indirectly, the category that the rule is designed to capture. In other words, rules of the form *NP* → *Det Nominal* are fine, but rules such as *Nominal* → *Nominal PP* are not. Consider the following example chunk rules adapted from Abney (1996).

$$\begin{aligned} NP &\rightarrow (Det)\ Noun{*}\ \ Noun \\ NP &\rightarrow Proper\text{-}Noun \\ VP &\rightarrow Verb \\ VP &\rightarrow Aux\ Verb \end{aligned}$$

The process of turning these rules into a single finite-state transducer is the same we introduced in Ch. 3 to capture spelling and phonological rules for English. Finite state transducers are created corresponding to each rule and are then unioned together to form a single machine that can then be determinized and minimized.

As we saw in Ch. 3, a major benefit of the finite-state approach is the ability to use the output of earlier transducers as inputs to subsequent transducers to form **cascades**. In **partial parsing**, this technique can be used to more closely approximate the output of true context-free parsers. In this approach, an initial set of transducers is used, in the way just described, to find a subset of syntactic base-phrases. These base-phrases are then passed as input to further transducers that detect larger and larger constituents such as prepositional phrases, verb phrases, clauses, and sentences. Con-

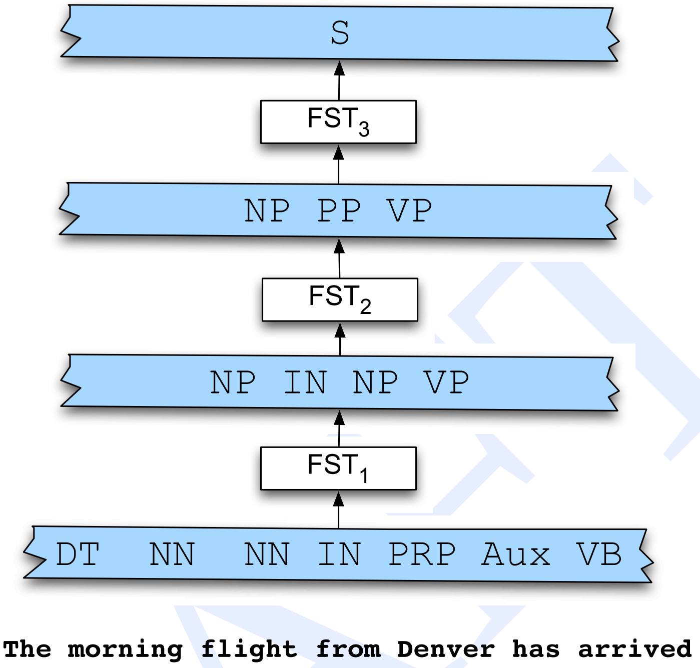


Figure 13.18 Chunk-based partial parsing via a set of finite-set cascades. FST$_1$ transduces from part-of-speech tags to base noun phrases and verb phrases. FST$_2$ finds prepositional phrases. Finally, FST$_3$ detects sentences.

sider the following rules, again adapted from Abney (1996).

$$FST_2\ PP \rightarrow Preposition\ NP$$
$$FST_3\ \ S \rightarrow PP*\ NP\ PP*\ VP\ PP*$$

Combining these two machines with the earlier rule-set results in a three machine cascade. The application of this cascade to Ex. 13.8 is shown in Fig. 13.18.

13.5.2 Machine Learning-Based Approaches to Chunking

As with part-of-speech tagging, an alternative to rule-based processing is to use supervised machine learning techniques to *train* a chunker using annotated data as a training set. As described earlier in Ch. 6, we can view the task as one of **sequential classification**, where a classifier is trained to label each element of the input in sequence. Any of the standard approaches to training classifiers apply to this problem. In the work that pioneered this approach, Ramshaw and Marcus (1995) used the transformation-based learning method described in Ch. 5.

The critical first step in such an approach is to find a way to view the chunking process that is amenable to sequential classification. A particularly fruitful approach is to treat chunking as a tagging task similar to part-of-speech tagging (Ramshaw and

Marcus, 1995). In this approach, a small tagset simultaneously encodes both the segmentation and the labeling of the chunks in the input. The standard way to do this has come to be called **IOB tagging** and is accomplished by introducing tags to represent the beginning (B) and internal (I) parts of each chunk, as well as those elements of the input that are outside (O) any chunk. Under this scheme, the size of the tagset is $(2n+1)$ where n is the number of categories to be classified. The following example shows the tagging version of the bracketing notation given earlier for Ex. 13.8 on pg. 25.

IOB TAGGING

(13.11) *The morning flight from Denver has arrived*
B_NP I_NP I_NP B_PP B_NP B_VP I_VP

The same sentence with only the base-NPs tagged illustrates the role of the O tags.

(13.12) *The morning flight from Denver has arrived.*
B_NP I_NP I_NP O B_NP O O

Notice that there is no explicit encoding of the end of a chunk in this scheme; the end of any chunk is implicit in any transition from an I or B, to a B or O tag. This encoding reflects the notion that when sequentially labeling words, it is generally quite a bit easier (at least in English) to detect the beginning of a new chunk than it is to know when a chunk has ended. Not surprisingly, there are a variety of other tagging schemes that represent chunks in subtly different ways, including some that explicitly mark the end of constituents. Tjong Kim Sang and Veenstra (1999) describe three variations on this basic tagging scheme and investigate their performance on a variety of chunking tasks.

Given such a tagging scheme, building a chunker consists of training a classifier to label each word of an input sentence with one of the IOB tags from the tagset. Of course, training requires training data consisting of the phrases of interest delimited and marked with the appropriate category. The direct approach is to annotate a representative corpus. Unfortunately, annotation efforts can be both expensive and time-consuming. It turns out that the best place to find such data for chunking, is in one of the already existing treebanks described earlier in Ch. 12.

Resources such as the Penn Treebank provide a complete syntactic parse for each sentence in a corpus. Therefore, base syntactic phrases can be extracted from the constituents provided by the Treebank parses. Finding the kinds of phrases we're interested in is relatively straightforward; we simply need to know the appropriate nonterminal names in the collection. Finding the boundaries of the chunks entails finding the head, and then including the material to the left of the head, ignoring the text to the right. This latter process is somewhat error-prone since it relies on the accuracy of the head-finding rules described earlier in Ch. 12.

Having extracted a training corpus from a treebank, we must now cast the training data into a form that's useful for training classifiers. In this case, each input can be represented as a set of features extracted from **a context window** that surrounds the word to be classified. Using a window that extends two words before, and two words after the word being classified seems to provide reasonable performance. Features extracted from this window include: the words themselves, their parts-of-speech, as well as the chunk tags of the preceding inputs in the window.

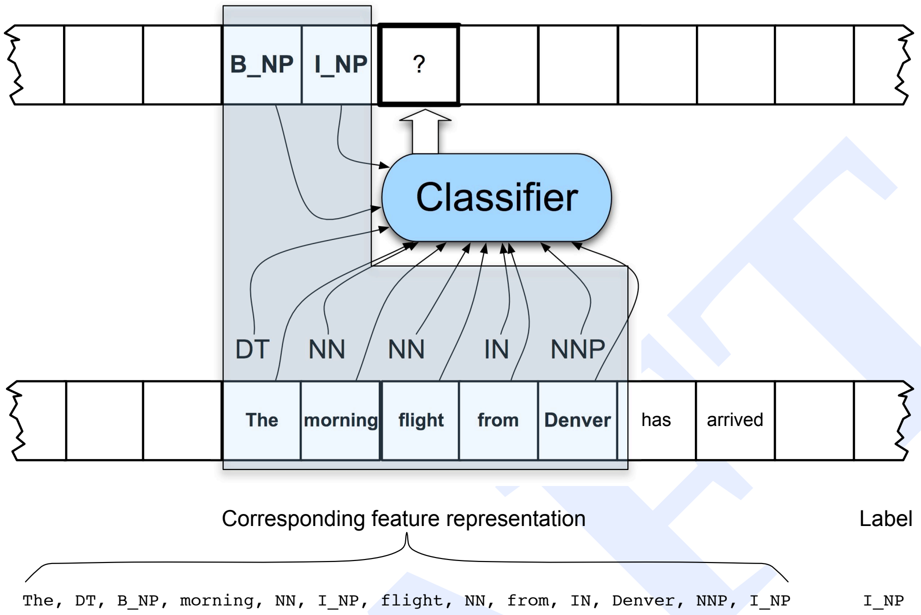

B_NP
I_NP
?
Classifier
DT
NN
NN
IN
NNP
The
morning
flight
from
Denver
has
arrived
Corresponding feature representation
Label
The, DT, B_NP, morning, NN, I_NP, flight, NN, from, IN, Denver, NNP, I_NP
I_NP

Figure 13.19 The sequential classifier-based approach to chunking. The chunker slides a context window over the sentence classifying words as it proceeds. At this point the classifier is attempting to label *flights*. Features derived from the context typically include: the current, previous and following words; the current, previous and following parts-of-speech; and the previous assignments of chunk-tags.

Fig. 13.19 illustrates this scheme with the example given earlier. During training, the classifier would be provided with a training vector consisting of the values of 12 features (using Penn Treebank tags) as shown. To be concrete, during training the classifier is given the 2 words to the right of the decision point along with their part-of-speech tags and their chunk tags, the word to be tagged along with its part-of-speech, the two words that follow along with their parts-of speech, and finally the correct chunk tag, in this case I_NP. During classification, the classifier is given the same vector without the answer and is asked to assign the most appropriate tag from its tagset.

13.5.3 Evaluating Chunking Systems

As with the evaluation of part-of-speech taggers, the evaluation of chunkers proceeds by comparing the output of a chunker against gold-standard answers provided by human annotators. However, unlike part-of-speech tagging and speech recognition, word-by-word accuracy measures are not adequate. Instead, chunkers are evaluated using measures borrowed from the field of information retrieval. In particular, the notions of precision, recall and the F measure are employed.

Precision measures the percentage of chunks that were provided by a system that were correct. Correct here means that both the boundaries of the chunk and the chunk's label are correct. Precision is therefore defined as:

$$\textbf{Precision:} = \frac{\text{Number of correct chunks given by system}}{\text{Total number of chunks given by system}}$$

Recall measures the percentage of chunks actually present in the input that were correctly identified by the system. Recall is defined as:

$$\textbf{Recall:} = \frac{\text{Number of correct chunks given by system}}{\text{Total number of actual chunks in the text}}$$

F-MEASURE

The **F-measure** (van Rijsbergen, 1975) provides a way to combine these two measures into a single metric. The F-measure is defined as:

$$F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

The β parameter is used to differentially weight the importance of recall and precision, based perhaps on the needs of an application. Values of $\beta > 1$ favor recall, while values of $\beta < 1$ favor precision. When $\beta = 1$, precision and recall are equally balanced; this is sometimes called $F_{\beta=1}$ or just F_1:

$$F_1 = \frac{2PR}{P+R} \tag{13.13}$$

The F-measure derives from a weighted harmonic mean of precision and recall. The harmonic mean of a set of numbers is the reciprocal of the arithmetic mean of the reciprocals:

$$\mathrm{HarmonicMean}(a_1, a_2, a_3, a_4, ..., a_n) = \frac{n}{\frac{1}{a_1}\frac{1}{a_2}\frac{1}{a_3}\dots\frac{1}{a_n}} \tag{13.14}$$

and hence F-measure is

$$F = \frac{1}{\frac{1}{\alpha P} \times \frac{1}{(1-\alpha)R}} \quad \text{or} \left(\text{with } \beta^2 = \frac{1-\alpha}{\alpha}\right) \quad \mathrm{F} = \frac{(\beta^2+1)\mathrm{PR}}{\beta^2\mathrm{P}+\mathrm{R}} \tag{13.15}$$

The best current systems achieve an F-measure of around .96 on the task of base-NP chunking. Learning-based systems designed to find a more complete set of base-phrases, such as the ones given in Fig. 13.20, achieve F-measures in the .92 to .94 range. The exact choice of learning approach seems to have little impact on these results; a wide-range of machine learning approaches achieve essentially the same results (Cardie et al., 2000). FST-based systems of the kind discussed in Sec. 13.5.1 achieved F-measures ranging from .85 to .92 on this task.

Factors limiting the performance of current systems include the accuracy of the part-of-speech taggers used to provide features for the system during testing, inconsistencies in the training data introduced by the process of extracting chunks from parse trees, and difficulty resolving ambiguities involving conjunctions. Consider the following examples that involve pre-nominal modifiers and conjunctions.

(13.16) [$_{NP}$ Late arrivals and departures] are commonplace during winter.

(13.17) [$_{NP}$ Late arrivals] and [$_{NP}$ cancellations] are commonplace during winter.

In the first example, *late* is shared by both *arrivals* and *departures* yielding a single long base-NP. In the second example, *late* is not shared and modifies *arrivals* alone, thus yielding two base-NPs. Distinguishing these two situations, and others like them, requires access to semantic and context information unavailable to current chunkers.

Label	Category	Proportion (%)	Example
NP	Noun Phrase	51	*The most frequently cancelled flight*
VP	Verb Phrase	20	*may not arrive*
PP	Prepositional Phrase	20	*to Houston*
ADVP	Adverbial Phrase	4	*earlier*
SBAR	Subordinate Clause	2	*that*
ADJP	Adjective Phrase	2	*late*

Figure 13.20 Most frequent base-phrases used in the 2000 CONLL shared task. These chunks correspond to the major categories contained in the Penn Treebank.

13.6 Summary

The two major ideas introduced in this chapter are those of **parsing** and **partial parsing**. Here's a summary of the main points we covered about these ideas:

- Parsing can be viewed as a **search** problem.
- Two common architectural metaphors for this search are **top-down** (starting with the root *S* and growing trees down to the input words) and **bottom-up** (starting with the words and growing trees up toward the root *S*).
- **Ambiguity** combined with the **repeated parsing of sub-trees** pose problems for simple backtracking algorithms.
- A sentence is **structurally ambiguous** if the grammar assigns it more than one possible parse.
- Common kinds of structural ambiguity include **PP-attachment**, **coordination ambiguity** and **noun-phrase bracketing ambiguity**.
- The **dynamic programming** parsing algorithms use a table of partial-parses to efficiently parse ambiguous sentences. The **CKY**, **Earley**, and **Chart-Parsing** algorithms all use dynamic-programming to solve the repeated parsing of sub-trees problem.
- The CKY algorithm restricts the form of its grammar to Chomsky-Normal Form; the Earley and Chart-parsers accept unrestricted context-free grammars.
- Many practical problems including **information extraction** problems can be solved without full parsing.
- Partial parsing and chunking are methods for identifying shallow syntactic constituents in a text.
- High accuracy partial parsing can be achieved either through rule-based or machine learning-based methods.

Bibliographical and Historical Notes

Writing about the history of compilers, Knuth notes:

> In this field there has been an unusual amount of parallel discovery of the same technique by people working independently.

Well, perhaps not unusual, if multiple discovery is the norm (see page **??**). But there has certainly been enough parallel publication that this history will err on the side of succinctness in giving only a characteristic early mention of each algorithm; the interested reader should see Aho and Ullman (1972).

Bottom-up parsing seems to have been first described by Yngve (1955), who gave a breadth-first bottom-up parsing algorithm as part of an illustration of a machine translation procedure. Top-down approaches to parsing and translation were described (presumably independently) by at least Glennie (1960), Irons (1961), and Kuno and Oettinger (1963). Dynamic programming parsing, once again, has a history of independent discovery. According to Martin Kay (personal communication), a dynamic programming parser containing the roots of the CKY algorithm was first implemented by John Cocke in 1960. Later work extended and formalized the algorithm, as well as proving its time complexity (Kay, 1967; Younger, 1967; Kasami, 1965). The related **well-formed substring table** (**WFST**) seems to have been independently proposed by Kuno (1965), as a data structure which stores the results of all previous computations in the course of the parse. Based on a generalization of Cocke's work, a similar data-structure had been independently described by Kay (1967) and Kay (1973). The top-down application of dynamic programming to parsing was described in Earley's Ph.D. dissertation (Earley, 1968) and Earley (1970). Sheil (1976) showed the equivalence of the WFST and the Earley algorithm. Norvig (1991) shows that the efficiency offered by all of these dynamic programming algorithms can be captured in any language with a *memoization* function (such as LISP) simply by wrapping the *memoization* operation around a simple top-down parser.

WFST

While parsing via cascades of finite-state automata had been common in the early history of parsing (Harris, 1962), the focus shifted to full CFG parsing quite soon afterward. Church (1980) argued for a return to finite-state grammars as a processing model for natural language understanding; other early finite-state parsing models include Ejerhed (1988). Abney (1991) argued for the important practical role of shallow parsing. Much recent work on shallow parsing applies machine learning to the task of learning the patterns; see for example Ramshaw and Marcus (1995), Argamon et al. (1998), Munoz et al. (1999).

The classic reference for parsing algorithms is Aho and Ullman (1972); although the focus of that book is on computer languages, most of the algorithms have been applied to natural language. A good programming languages textbook such as Aho et al. (1986) is also useful.

EXERCISES

13.1 Implement the algorithm to convert arbitrary context-free grammars to CNF. Apply your program to the $\mathscr{L}_1$ grammar.

13.2 Implement the CKY algorithm and test it using your converted $\mathscr{L}_1$ grammar.

13.3 Rewrite the CKY algorithm given on page 13.10 so that it can accept grammars that contain unit productions.

13.4 Augment the Earley algorithm of Fig. 13.13 to enable parse trees to be retrieved from the chart by modifying the pseudocode for the COMPLETER as described on page 22.

13.5 Implement the Earley algorithm as augmented in the previous exercise. Check it on a test sentence using the $\mathscr{L}_1$ grammar.

13.6 Alter the Earley algorithm so that it makes better use of bottom-up information to reduce the number of useless predictions.

13.7 Attempt to recast the CKY and Earley algorithms in the chart parsing paradigm.

13.8 Discuss the relative advantages and disadvantages of partial parsing versus full parsing.

13.9 Implement a more extensive finite-state grammar for noun-groups using the examples given in Sec. 13.5 and test it on some sample noun-phrases. If you have access to an on-line dictionary with part-of-speech information, start with that; if not, build a more restricted system by hand.

13.10 Discuss how you would augment a parser to deal with input that may be incorrect, such as spelling errors or misrecognitions from a speech recognition system.

Abney, S. (1996). Partial parsing via finite-state cascades. *Natural Language Engineering*, *2*(4), 337–344.

Abney, S. P. (1991). Parsing by chunks. In Berwick, R. C., Abney, S. P., and Tenny, C. (Eds.), *Principle-Based Parsing: Computation and Psycholinguistics*, pp. 257–278. Kluwer, Dordrecht.

Aho, A. V., Sethi, R., and Ullman, J. D. (1986). *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA.

Aho, A. V. and Ullman, J. D. (1972). *The Theory of Parsing, Translation, and Compiling*, Vol. 1. Prentice-Hall, Englewood Cliffs, NJ.

Argamon, S., Dagan, I., and Krymolowski, Y. (1998). A memory-based approach to learning shallow natural language patterns. In *COLING/ACL-98*, Montreal, pp. 67–73. ACL.

Bacon, F. (1620). *Novum Organum*. Annotated edition edited by Thomas Fowler published by Clarendon Press, Oxford, 1889.

Cardie, C., Daelemans, W., Ndellec, C., and Sang, E. T. K. (Eds.). (2000). *Proceedings of the Fourth Conference on Computational Language Learning*, Lisbon, Portugal.

Church, K. W. and Patil, R. (1982). Coping with syntactic ambiguity. *American Journal of Computational Linguistics*, *8*(3-4), 139–149.

Church, K. W. (1980). On memory limitations in natural language processing. Master's thesis, MIT. Distributed by the Indiana University Linguistics Club.

Earley, J. (1968). *An Efficient Context-Free Parsing Algorithm*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, *6*(8), 451–455. Reprinted in Grosz et al. (1986).

Ejerhed, E. I. (1988). Finding clauses in unrestricted text by finitary and stochastic methods. In *Second Conference on Applied Natural Language Processing*, pp. 219–227. ACL.

Glennie, A. (1960). On the syntax machine and the construction of a universal compiler. Tech. rep. No. 2, Contr. NR 049-141, Carnegie Mellon University (at the time Carnegie Institute of Technology), Pittsburgh, PA†.

Harris, Z. S. (1962). *String Analysis of Sentence Structure*. Mouton, The Hague.

Irons, E. T. (1961). A syntax directed compiler for ALGOL 60. *Communications of the ACM*, *4*, 51–55.

Kaplan, R. M. (1973). A general syntactic processor. In Rustin, R. (Ed.), *Natural Language Processing*, pp. 193–241. Algorithmics Press, New York.

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Tech. rep. AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA†.

Kay, M. (1967). Experiments with a powerful parser. In *Proc. 2eme Conference Internationale sur le Traitement Automatique des Langues*, Grenoble.

Kay, M. (1973). The MIND system. In Rustin, R. (Ed.), *Natural Language Processing*, pp. 155–188. Algorithmics Press, New York.

Kay, M. (1986). Algorithm schemata and data structures in syntactic processing. In *Readings in natural language processing*, pp. 35–70. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Kuno, S. (1965). The predictive analyzer and a path elimination technique. *Communications of the ACM*, *8*(7), 453–462.

Kuno, S. and Oettinger, A. G. (1963). Multiple-path syntactic analyzer. In Popplewell, C. M. (Ed.), *Information Processing 1962: Proceedings of the IFIP Congress 1962*, Munich, pp. 306–312. North-Holland. Reprinted in Grosz et al. (1986).

Munoz, M., Punyakanok, V., Roth, D., and Zimak, D. (1999). A learning approach to shallow parsing. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-99)*, College Park, MD, pp. 168–178. ACL.

Norvig, P. (1991). Techniques for automatic memoization with applications to context-free parsing. *Computational Linguistics*, *17*(1), 91–98.

Ramshaw, L. A. and Marcus, M. P. (1995). Text chunking using transformation-based learning. In *Proceedings of the Third Annual Workshop on Very Large Corpora*, pp. 82–94. ACL.

Sheil, B. A. (1976). Observations on context free parsing. *SMIL: Statistical Methods in Linguistics*, *1*, 71–109.

Tjong Kim Sang, E. F. and Veenstra, J. (1999). Representing text chunks. In *Proceedings of EACL 1999*, pp. 173–179.

van Rijsbergen, C. J. (1975). *Information Retrieval*. Butterworths, London.

Yngve, V. H. (1955). Syntax and the problem of multiple meaning. In Locke, W. N. and Booth, A. D. (Eds.), *Machine Translation of Languages*, pp. 208–226. MIT Press, Cambridge, MA.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, *10*, 189–208.

14 STATISTICAL PARSING

Two roads diverged in a wood, and I –
I took the one less traveled by...
Robert Frost, *The Road Not Taken*

The characters in Damon Runyon's short stories are willing to bet "on any proposition whatever", as Runyon says about Sky Masterson in *The Idyll of Miss Sarah Brown*; from the probability of getting aces back-to-back to the odds against a man being able to throw a peanut from second base to home plate. There is a moral here for language processing: with enough knowledge we can figure the probability of just about anything. The last two chapters have introduced sophisticated models of syntactic structure and its parsing. In this chapter we show that it is possible to build probabilistic models of syntactic knowledge and use some of this probabilistic knowledge in efficient probabilistic parsers.

One crucial use of probabilistic parsing is to solve the problem of **disambiguation**. Recall from Ch. 13 that sentences on average tend to be very syntactically ambiguous, due to problems like **coordination ambiguity** and **attachment ambiguity**. The CKY and Earley parsing algorithms could represent these ambiguities in an efficient way, but were not equipped to resolve them. A probabilistic parser offers a solution to the problem: compute the probability of each interpretation, and choose the most-probable interpretation. Thus, due to the prevalence of ambiguity, most modern parsers used for natural language understanding tasks (thematic role labeling, summarization, question-answering, machine translation) are of necessity probabilistic.

Another important use of probabilistic grammars and parsers is in **language modeling** for speech recognition. We saw that *N*-gram grammars are used in speech recognizers to predict upcoming words, helping constrain the acoustic model search for words. Probabilistic versions of more sophisticated grammars can provide additional predictive power to a speech recognizer. Of course humans have to deal with the same problems of ambiguity as do speech recognizers, and it is interesting that psychological experiments suggest that people use something like these probabilistic grammars in human language-processing tasks (e.g., human reading or speech understanding).

The most commonly used probabilistic grammar is the **probabilistic context-free grammar** (PCFG), a probabilistic augmentation of context-free grammars in which each rule is associated with a probability. We introduce PCFGs in the next section, showing how they can be trained on a hand-labeled Treebank grammar, and how they can be parsed. We present the most basic parsing algorithm for PCFGs, which is the probabilistic version of the **CKY algorithm** that we saw in Ch. 13.

We then show a number of ways that we can improve on this basic probability model (PCFGs trained on Treebank grammars). One method of improving a trained Treebank grammar is to change the names of the non-terminals. By making the non-terminals sometimes more specific and sometimes more general, we can come up with a grammar with a better probability model that leads to improved parsing scores. Another augmentation of the PCFG works by adding more sophisticated conditioning factors, extending PCFGs to handle probabilistic **subcategorization** information and probabilistic **lexical dependencies**.

Finally, we describe the standard PARSEVAL metrics for evaluating parsers, and discuss some psychological results on human parsing.

14.1 Probabilistic Context-Free Grammars

PCFG
SCFG

The simplest augmentation of the context-free grammar is the **Probabilistic Context-Free Grammar** (**PCFG**), also known as the **Stochastic Context-Free Grammar** (**SCFG**), first proposed by Booth (1969). Recall that a context-free grammar G is defined by four parameters (N, Σ, P, S); a probabilistic context-free grammar augments each rule in P with a conditional probability. A PCFG is thus defined by the following components:

N a set of **non-terminal symbols** (or **variables**)

Σ a set of **terminal symbols** (disjoint from N)

R a set of **rules** or productions, each of the form $A \rightarrow \beta\ [p]$, where A is a non-terminal, β is a string of symbols from the infinite set of strings $(\Sigma \cup N)*$, and p is a number between 0 and 1 expressing $P(\beta|A)$

S a designated **start symbol**

That is, a PCFG differs from a standard CFG by augmenting each rule in R with a conditional probability:

$$A \rightarrow \beta \quad [p] \tag{14.1}$$

Here p expresses the probability that the given non-terminal A will be expanded to the sequence β. That is, p is the conditional probability of a given expansion β given the left-hand-side (LHS) non-terminal A. We can represent this probability as

$$P(A \rightarrow \beta)$$

or as

$$P(A \rightarrow \beta|A)$$

$S \rightarrow NP\ VP$	[.80]	*Det* $\rightarrow$ *that* [.10] $\mid$ *a* [.30] $\mid$ *the* [.60]
$S \rightarrow Aux\ NP\ VP$	[.15]	*Noun* $\rightarrow$ *book* [.10] $\mid$ *flight* [.30]
$S \rightarrow VP$	[.05]	$\mid$ *meal* [.15] $\mid$ *money* [.05]
$NP \rightarrow Pronoun$	[.35]	$\mid$ *flights* [.40] $\mid$ *dinner* [.10]
$NP \rightarrow Proper\text{-}Noun$	[.30]	*Verb* $\rightarrow$ *book* [.30] $\mid$ *include* [.30]
$NP \rightarrow Det\ Nominal$	[.20]	$\mid$ *prefer*; [.40]
$NP \rightarrow Nominal$	[.15]	*Pronoun* $\rightarrow$ *I* [.40] $\mid$ *she* [.05]
$Nominal \rightarrow Noun$	[.75]	$\mid$ *me* [.15] $\mid$ *you* [.40]
$Nominal \rightarrow Nominal\ Noun$	[.20]	*Proper-Noun* $\rightarrow$ *Houston* [.60]
$Nominal \rightarrow Nominal\ PP$	[.05]	$\mid$ *TWA* [.40]
$VP \rightarrow Verb$	[.35]	*Aux* $\rightarrow$ *does* [.60] $\mid$ *can* [40]
$VP \rightarrow Verb\ NP$	[.20]	*Preposition* $\rightarrow$ *from* [.30] $\mid$ *to* [.30]
$VP \rightarrow Verb\ NP\ PP$	[.10]	$\mid$ *on* [.20] $\mid$ *near* [.15]
$VP \rightarrow Verb\ PP$	[.15]	$\mid$ *through* [.05]
$VP \rightarrow Verb\ NP\ NP$	[.05]	
$VP \rightarrow VP\ PP$	[.15]	
$PP \rightarrow Preposition\ NP$	[1.0]	

Figure 14.1 A PCFG which is a probabilistic augmentation of the $\mathcal{L}_1$ miniature English CFG grammar and lexicon of Fig. **??** in Ch. 13. These probabilities were made up for pedagogical purposes and are not based on a corpus (since any real corpus would have many more rules, and so the true probabilities of each rule would be much smaller).

or as

$$P(RHS|LHS)$$

Thus if we consider all the possible expansions of a non-terminal, the sum of their probabilities must be 1:

$$\sum_{\beta} P(A \rightarrow \beta) = 1$$

Fig. 14.1 shows a PCFG: a probabilistic augmentation of the $\mathcal{L}_1$ miniature English CFG grammar and lexicon . Note that the probabilities of all of the expansions of each non-terminal sum to 1. Also note that these probabilities were made up for pedagogical purposes. In any real grammar there are a great many more rules for each non-terminal and hence the probabilities of any particular rule would tend to be much smaller.

CONSISTENT

A PCFG is said to be **consistent** if the sum of the probabilities of all sentences in the language equals 1. Certain kinds of recursive rules cause a grammar to be inconsistent by causing infinitely looping derivations for some sentences. For example a rule $S \rightarrow S$ with probability 1 would lead to lost probability mass due to derivations that never terminate. See Booth and Thompson (1973) for more details on consistent and inconsistent grammars.

How are PCFGs used? A PCFG can be used to estimate a number of useful probabilities concerning a sentence and its parse tree(s), including the probability of a particular parse tree (useful in disambiguation) and the probability of a sentence or a piece of a sentence (useful in language modeling). Let's see how this works.

14.1.1 PCFGs for Disambiguation

A PCFG assigns a probability to each parse tree T (i.e., each **derivation**) of a sentence S. This attribute is useful in **disambiguation**. For example, consider the two parses of the sentence "Book the dinner flights" shown in Fig. 14.2. The sensible parse on the left means "Book flights that serve dinner". The nonsensical parse on the right, however, would have to mean something like "Book flights on behalf of 'the dinner'?", the way that a structurally similar sentence like "Can you book John flights?" means something like "Can you book flights on behalf of John?".

The probability of a particular parse T is defined as the product of the probabilities of all the n rules used to expand each of the n non-terminal nodes in the parse tree T, (where each rule i can be expressed as $LHS_i \rightarrow RHS_i$):

$$P(T,S) = \prod_{i=1}^{n} P(RHS_i|LHS_i) \tag{14.2}$$

The resulting probability $P(T,S)$ is both the joint probability of the parse and the sentence, and also the probability of the parse $P(T)$. How can this be true? First, by the definition of joint probability:

$$P(T,S) = P(T)P(S|T) \tag{14.3}$$

But since a parse tree includes all the words of the sentence, $P(S|T)$ is 1. Thus:

$$P(T,S) = P(T)P(S|T) = P(T) \tag{14.4}$$

The probability of each of the trees in Fig. 14.2 can be computed by multiplying together the probabilities of each of the rules used in the derivation. For example, the probability of the left tree in Figure 14.2a (call it T_{left}) and the right tree (Figure 14.2b or T_{right}) can be computed as follows:

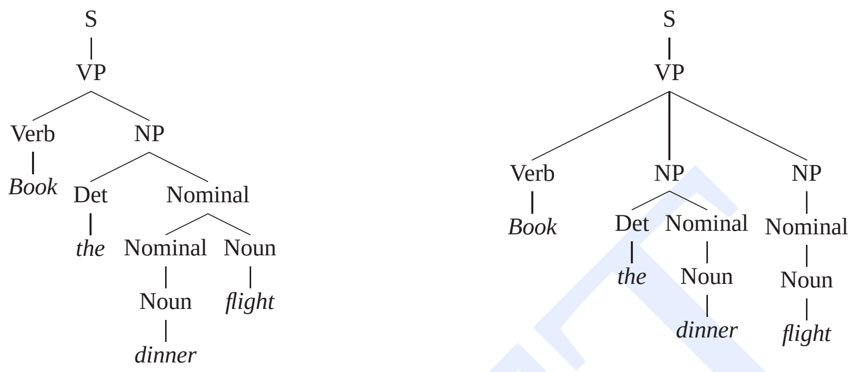
$$\begin{aligned} P(T_{left}) &= .05 * .20 * .20 * .20 * .75 * .30 * .60 * .10 * .40 = \mathbf{2.2 \times 10^{-6}} \\ P(T_{right}) &= .05 * .10 * .20 * .15 * .75 * .75 * .30 * .60 * .10 * .40 = \mathbf{6.1 \times 10^{-7}} \end{aligned}$$

We can see that the left (transitive) tree in Fig. 14.2(a) has a much higher probability than the ditransitive tree on the right. Thus this parse would correctly be chosen by a disambiguation algorithm which selects the parse with the highest PCFG probability.

Let's formalize this intuition that picking the parse with the highest probability is the correct way to do disambiguation. Consider all the possible parse trees for a given sentence S. The string of words S is called the **yield** of any parse tree over S. Thus out of all parse trees with a yield of S, the disambiguation algorithm picks the parse tree which is most probable given S:

YIELD

$$\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\text{argmax}} P(T|S) \tag{14.5}$$



Rules			P	Rules			P
S	→	VP	.05	S	→	VP	.05
VP	→	Verb NP	.20	VP	→	Verb NP NP	.10
NP	→	Det Nominal	.20	NP	→	Det Nominal	.20
Nominal	→	Nominal Noun	.20	NP	→	Nominal	.15
Nominal	→	Noun	.75	Nominal	→	Noun	.75
				Nominal	→	Noun	.75
Verb	→	book	.30	Verb	→	book	.30
Det	→	the	.60	Det	→	the	.60
Noun	→	dinner	.10	Noun	→	dinner	.10
Noun	→	flights	.40	Noun	→	flights	.40

Figure 14.2 Two parse trees for an ambiguous sentence, The transitive parse (a) corresponds to the sensible meaning “Book flights that serve dinner”, while the ditransitive parse (b) to the nonsensical meaning “Book flights on behalf of ‘the dinner’”.

By definition, the probability $P(T|S)$ can be rewritten as $P(T,S)/P(S)$, thus leading to:

(14.6)
$$\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\text{argmax}} \frac{P(T,S)}{P(S)}$$

Since we are maximizing over all parse trees for the same sentence, $P(S)$ will be a constant for each tree, so we can eliminate it:

(14.7)
$$\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\text{argmax}} P(T,S)$$

Furthermore, since we showed above that $P(T,S) = P(T)$, the final equation for choosing the most likely parse neatly simplifies to choosing the parse with the highest probability:

(14.8)
$$\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\text{argmax}} P(T)$$

14.1.2 PCFGs for Language Modeling

A second attribute of a PCFG is that it assigns a probability to the string of words constituting a sentence. This is important in **language modeling**, whether for use in speech recognition, machine translation, spell-correction, augmentative communication, or other applications. The probability of an unambiguous sentence is $P(T,S) = P(T)$ or just the probability of the single parse tree for that sentence. The probability of an ambiguous sentence is the sum of the probabilities of all the parse trees for the sentence:

$$\begin{aligned} P(S) &= \sum_{T s.t. S=\text{yield}(T)} P(T,S) && (14.9) \\ &= \sum_{T s.t. S=\text{yield}(T)} P(T) && (14.10) \end{aligned}$$

An additional feature of PCFGs that is useful for language modeling is their ability to assign a probability to substrings of a sentence. For example, suppose we want to know the probability of the next word w_i in a sentence given all the words we've seen so far $w_1, \ldots, w_{i-1}$. The general formula for this is:

$$P(w_i|w_1,w_2,\ldots,w_{i-1}) = \frac{P(w_1,w_2,\ldots,w_{i-1},w_i,\ldots)}{P(w_1,w_2,\ldots,w_{i-1},\ldots)} \qquad (14.11)$$

We saw in Ch. 4 a simple approximation of this probability using N-grams, conditioning on only the last word or two instead of the entire context; thus the **bigram approximation** would give us:

$$P(w_i|w_1,w_2,\ldots,w_{i-1}) \approx \frac{P(w_{i-1},w_i)}{P(w_{i-1})} \qquad (14.12)$$

But the fact that the N-gram model can only make use of a couple words of context means it is ignoring potentially useful prediction cues. Consider predicting the word *after* in the following sentence from Chelba and Jelinek (2000):

(14.13) the contract ended with a loss of 7 cents after trading as low as 9 cents

A trigram grammar must predict *after* from the words *7 cents*, while it seems clear that the verb *ended* and the subject *contract* would be useful predictors that a PCFG-based parser could help us make use of. Indeed, it turns out that a PCFGs allow us to condition on the entire previous context $w_1, w_2, \ldots, w_{i-1}$ shown in Equation (14.11). We'll see the details of ways to use PCFGs and augmentations of PCFGs as language models in Sec. 14.9.

In summary, this section and the previous one have shown that PCFGs can be applied both to disambiguation in syntactic parsing and to word prediction in language modeling. Both of these applications require that we be able to compute the probability of parse tree T for a given sentence S. The next few sections introduce some algorithms for computing this probability.

14.2 PROBABILISTIC CKY PARSING OF PCFGS

The parsing problem for PCFGs is to produce the most-likely parse $\hat{T}$ for a given sentence S, i.e.,

(14.14)
$$\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\text{argmax}} P(T)$$

The algorithms for computing the most-likely parse are simple extensions of the standard algorithms for parsing; there are probabilistic versions of both the CKY and Earley algorithms of Ch. 13. Most modern probabilistic parsers are based on the **probabilistic CKY (Cocke-Kasami-Younger)** algorithm, first described by Ney (1991).

PROBABILISTIC CKY

As with the CKY algorithm, we will assume for the probabilistic CKY algorithm that the PCFG is in Chomsky normal form. Recall from page **??** that grammars in CNF are restricted to rules of the form $A \rightarrow B\,C$, or $A \rightarrow w$. That is, the right-hand side of each rule must expand to either two non-terminals or to a single terminal.

For the CKY algorithm, we represented each sentence as having indices between the words. Thus an example sentence like

(14.15) Book the flight through Houston.

would assume the following indices between each word:

(14.16) ⓪ Book ① the ② flight ③ through ④ Houston ⑤

Using these indices, each constituent in the CKY parse tree is encoded in a two-dimensional matrix. Specifically, for a sentence of length n and a grammar that contains V non-terminals, we use the upper-triangular portion of an $(n+1) \times (n+1)$ matrix. For CKY, each cell $table[i, j]$ contained a list of constituents that could span the sequence of words from i to j. For probabilistic CKY, it's slightly simpler to think of the constituents in each cell as constituting a third dimension of maximum length V. This third dimension corresponds to each nonterminal that can be placed in this cell, and the value of the cell is then a probability for that nonterminal/constituent rather than a list of constituents. In summary, each cell $[i, j, A]$ in this $(n+1) \times (n+1) \times V$ matrix is the probability of a constituent A that spans positions i through j of the input.

Fig. 14.3 gives pseudocode for this probabilistic CKY algorithm, extending the basic CKY algorithm from Fig. **??**.

Like the CKY algorithm, the probabilistic CKY algorithm as shown in Fig. 14.3 requires a grammar in Chomsky Normal Form. Converting a probabilistic grammar to CNF requires that we also modify the probabilities so that the probability of each parse remains the same under the new CNF grammar. Exercise 14.2 asks you to modify the algorithm for conversion to CNF in Ch. 13 so that it correctly handles rule probabilities.

In practice, we more often use a generalized CKY algorithm which handles unit productions directly rather than converting them to CNF. Recall that Exercise **??** asked you to make this change in CKY; Exercise 14.3 asks you to extend this change to probabilistic CKY.

Let's see an example of the probabilistic CKY chart, using the following mini-grammar which is already in CNF:

```
function PROBABILISTIC-CKY(words,grammar) returns most probable parse
                                              and its probability
  for $j \leftarrow$ from 1 to LENGTH(words) do
     for all { $A$ | $A \rightarrow words[j] \in grammar$ }
        $table[j-1, j, A] \leftarrow P(A \rightarrow words[j])$
     for $i \leftarrow$ from $j-2$ downto 0 do
        for $k \leftarrow i+1$ to $j-1$ do
           for all { $A$ | $A \rightarrow BC \in grammar$,
                  and $table[i,k,B] > 0$ and $table[k,j,C] > 0$ }
              if ($table[i,j,A] < P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$) then
                 $table[i,j,A] \leftarrow P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$
                 $back[i,j,A] \leftarrow \{k,B,C\}$
  return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
```

Figure 14.3 The probabilistic CKY algorithm for finding the maximum probability parse of a string of *num_words* words given a PCFG grammar with *num_rules* rules in Chomsky Normal Form. *back* is an array of back-pointers used to recover the best parse. The *build_tree* function is left as an exercise to the reader.

S	$\rightarrow$	$NP\ VP$	.80	Det	$\rightarrow$	the	.50
NP	$\rightarrow$	$Det\ N$	.30	Det	$\rightarrow$	a	.40
VP	$\rightarrow$	$V\ NP$	.20	N	$\rightarrow$	$meal$	.01
V	$\rightarrow$	$includes$	.05	N	$\rightarrow$	$flight$	.02

Given this grammar, Fig. 14.4 shows the first steps in the probabilistic CKY parse of this sentence:

(14.17) The flight includes a meal

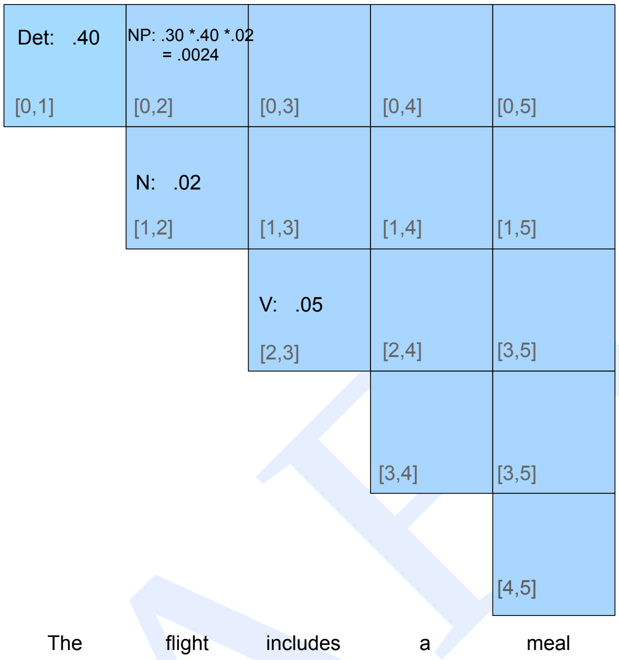
14.3 Learning PCFG Rule Probabilities

TREEBANK

Where do PCFG rule probabilities come from? There are two ways to learn probabilities for the rules of a grammar. The simplest way is to use a **treebank**, a corpus of already-parsed sentences. Recall that we introduced in Ch. 12 the idea of treebanks and the commonly-used **Penn Treebank** (Marcus et al., 1993), a collection of parse trees in English, Chinese, and other languages distributed by the Linguistic Data Consortium. Given a treebank, the probability of each expansion of a non-terminal can be computed by counting the number of times that expansion occurs and then normalizing.

$$P(\alpha \rightarrow \beta | \alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\sum_{\gamma} \text{Count}(\alpha \rightarrow \gamma)} = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)} \qquad (14.18)$$

If we don't have a treebank, but we do have a (non-probabilistic) parser, we can generate the counts we need for computing PCFG rule probabilities by first parsing a corpus of sentences with the parser. If sentences were unambiguous, it would be as



Det: .40 [0,1]	NP: .30 *.40 *.02 = .0024 [0,2]	[0,3]	[0,4]	[0,5]
	N: .02 [1,2]	[1,3]	[1,4]	[1,5]
		V: .05 [2,3]	[2,4]	[3,5]
			[3,4]	[3,5]
				[4,5]
The	flight	includes	a	meal

Figure 14.4 The beginning of the probabilistic CKY matrix. Filling out the rest of the chart is left as Exercise 14.4 for the reader.

simple as this: parse the corpus, increment a counter for every rule in the parse, and then normalize to get probabilities.

But wait! Since most sentences are ambiguous, i.e. have multiple parses, we don't know which parse to count the rules in. Instead, we need to keep a separate count for each parse of a sentence and weight each of these partial counts by the probability of the parse it appears in. But to get these parse probabilities to weight the rules we need to already have a probabilistic parser.

INSIDE-OUTSIDE

EXPECTATION
E-STEP (EXPECTATION STEP) IN EM
MAXIMIZATION
M-STEP (MAXIMIZATION STEP) IN EM

The intuition for solving this chicken-and-egg problem is to incrementally improve our estimates by beginning with a parser with equal rule probabilities, parsing the sentence, compute a probability for each parse, use these probabilities to weight the counts, then reestimate the rule probabilities, and so on, until our probabilities converge. The standard algorithm for computing this is called the **inside-outside** algorithm, and was proposed by Baker (1979) as a generalization of the forward-backward algorithm of Ch. 6. Like forward-backward, inside-outside is a special case of the EM (expectation-maximization) algorithm, and hence has two steps: the **expectation** step, or **E-step (expectation step) in EM**, and the **maximization** step, or **M-step (maximization step) in EM**. See Lari and Young (1990) or Manning and Schütze (1999) for a complete description of the algorithm.

This use of the inside-outside algorithm to estimate the rule probabilities for a grammar is actually a kind of limited use of inside-outside. The inside-outside algorithm can actually be used not only to set the rule probabilities, but even to induce the grammar rules themselves. It turns out, however, that grammar induction is so difficult that inside-outside by itself is not a very successful grammar inducer; see the end notes for pointers to other grammar induction algorithms.

14.4 Problems with PCFGs

While probabilistic context-free grammars are a natural extension to context-free grammars, they have two main problems as probability estimators:

poor independence assumptions: CFG rules impose an independence assumption on probabilities, resulting in poor modeling of structural dependencies across the parse tree.

lack of lexical conditioning: CFG rules don't model syntactic facts about specific words, leading to problems with subcategorization ambiguities, preposition attachment, and coordinate structure ambiguities.

Because of these problems, most current probabilistic parsing models use some augmented version of PCFGs, or modify the Treebank-based grammar in some way. In the next few sections after discussing the problems in more detail we will introduce some of these augmentations.

14.4.1 Independence assumptions miss structural dependencies between rules

Let's look at these problems in more detail. Recall that in a CFG the expansion of a non-terminal is independent of the context, i.e., of the other nearby non-terminals in the parse tree. Similarly, in a PCFG, the probability of a particular rule like $NP \rightarrow Det\ N$ is also independent of the rest of the tree. By definition, the probability of a group of independent events is the product of their probabilities. These two facts explain why in a PCFG we compute the probability of a tree by just multiplying the probabilities of each non-terminal expansion.

Unfortunately this CFG independence assumption results in poor probability estimates. This is because in English the choice of how a node expands can after all be dependent on the location of the node in the parse tree. For example, in English it turns out that NPs that are syntactic **subjects** are far more likely to be pronouns, while NPs that are syntactic **objects** are far more likely to be non-pronominal (e.g., a proper noun or a determiner noun sequence), as shown by these statistics for NPs in the Switchboard corpus (Francis et al., 1999): [1]

[1] Distribution of subjects from 31,021 declarative sentences; distribution of objects from 7,489 sentences. This tendency is caused by the use of subject position to realize the **topic** or old information in a sentence (Givón, 1990). Pronouns are a way to talk about old information, while non-pronominal ("lexical") noun-phrases are often used to introduce new referents. We'll talk more about new and old information in Ch. 21.

	Pronoun	Non-Pronoun
Subject	91%	9%
Object	34%	66%

Unfortunately there is no way to represent this contextual difference in the probabilities in a PCFG. Consider two expansions of the non-terminal *NP* as a pronoun or as a determiner+noun. How shall we set the probabilities of these two rules? If we set their probabilities to their overall probability in the Switchboard corpus, the two rules have about equal probability.

$$\begin{aligned} NP &\rightarrow DT\ NN \quad .28 \\ NP &\rightarrow PRP \quad .25 \end{aligned}$$

Because PCFGs don't allow a rule probability to be conditioned on surrounding context, this equal probability is all we get; there is no way to capture the fact that in subject position, the probability for $NP \rightarrow PRP$ should go up to .91, while in object position, the probability for $NP \rightarrow DT\ NN$ should go up to .66.

These dependencies could be captured if the probability of expanding an NP as a pronoun (e.g., $NP \rightarrow PRP$) versus a lexical NP (e.g., $NP \rightarrow DT\ NN$) were *conditioned* on whether the NP was a subject or an object. Sec. 14.5 will introduce the technique of **parent annotation** for adding this kind of conditioning.

14.4.2 Lack of sensitivity to lexical dependencies

A second class of problems with PCFGs is their lack of sensitivity to the words in the parse tree. Words do play a role in PCFGs, since the parse probability includes the probability of a word given a part-of-speech (i.e., from rules like $V \rightarrow$ *sleep*, $NN \rightarrow$ *book*, etc).

PREPOSITIONAL PHRASE ATTACHMENT

But it turns out that lexical information is useful in other places in the grammar, such as in resolving **prepositional phrase attachment** (**PP**) ambiguities. Since prepositional phrases in English can modify a noun phrase or a verb phrase, when a parser finds a prepositional phrase, it must decide where to **attach** it into the tree. Consider the following examples:

(14.19) Workers dumped sacks into a bin.

VP ATTACHMENT

NP ATTACHMENT

Fig. 14.5 shows two possible parse trees for this sentence; the one on the left is the correct parse; Fig. 14.6 shows another perspective on the preposition attachment problem, demonstrating that resolving the ambiguity in Fig. 14.5 is equivalent to deciding whether to attach the prepositional phrase into the rest of the tree at the NP or VP nodes; we say that the correct parse requires **VP attachment** while the incorrect parse implies **NP attachment**.

Why doesn't a PCFG already deal with PP attachment ambiguities? Note that the two parse trees in Fig. 14.5 have almost the exact same rules; they differ only in that the left-hand parse has has this rule:

$$VP \rightarrow VBD\ NP\ PP$$

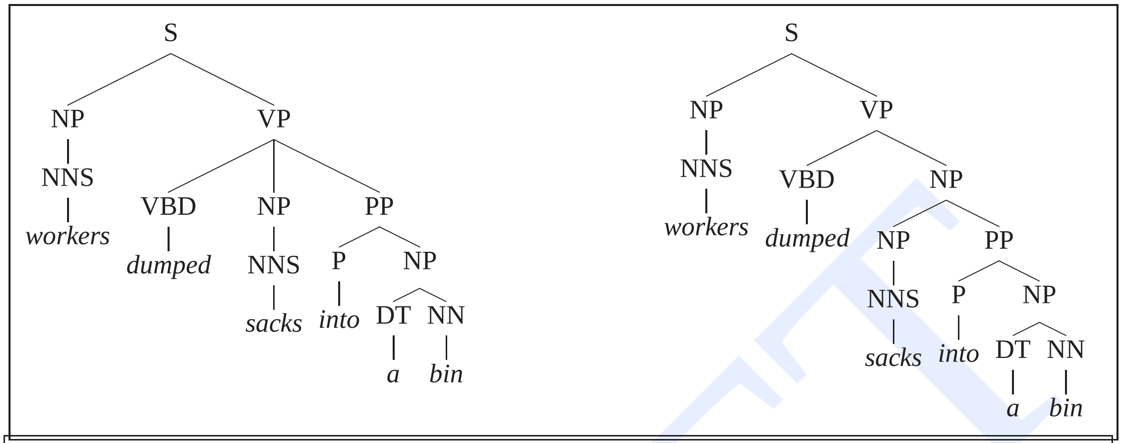


Figure 14.5 Two possible parse trees for a **prepositional phrase attachment ambiguity**. The left parse is the sensible one, in which 'into a bin' describes the resulting location of the sacks. In the right incorrect parse, the sacks to be dumped are the ones which are already 'into a bin', whatever that could mean.

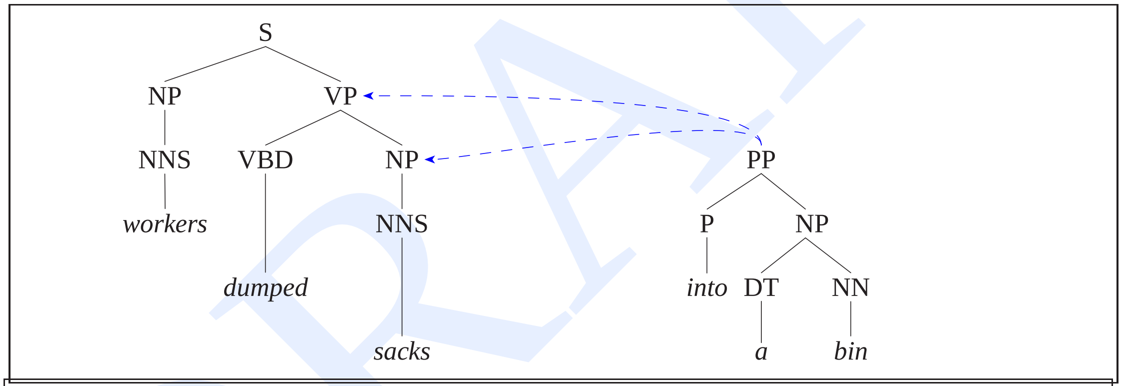


Figure 14.6 Another view of the preposition attachment problem; should the PP on the right attach to the VP or NP nodes of the partial parse tree on the left?

while the right-hand parse has these:

$$\begin{aligned} VP &\rightarrow VBD\ NP \\ NP &\rightarrow NP\ PP \end{aligned}$$

Depending on how these probabilities are set, a PCFG will **always** either prefer NP attachment or VP attachment. As it happens, NP attachment is slightly more common in English, and so if we trained these rule probabilities on a corpus, we might always prefer NP attachment, causing us to misparse this sentence.

But suppose we set the probabilities to prefer the VP attachment for this sentence. Now we would misparse the following sentence which requires NP attachment:

(14.20) fishermen caught tons of herring

What is the information in the input sentence which lets us know that (14.20) requires NP attachment while (14.19) requires VP attachment?

It should be clear that these preferences come from the identities of the verbs, nouns and prepositions. It seems that the affinity between the verb *dumped* and the preposition *into* is greater than the affinity between the noun *sacks* and the preposition *into*, thus leading to VP attachment. On the other hand in (14.20) , the affinity between *tons* and *of* is greater than that between *caught* and *of*, leading to NP attachment.

LEXICAL DEPENDENCY

Thus in order to get the correct parse for these kinds of examples, we need a model which somehow augments the PCFG probabilities to deal with these **lexical dependency** statistics for different verbs and prepositions.

Coordination ambiguities are another case where lexical dependencies are the key to choosing the proper parse. Fig. 14.7 shows an example from Collins (1999), with two parses for the phrase *dogs in houses and cats*. Because *dogs* is semantically a better conjunct for *cats* than *houses* (and because dogs can't fit inside cats) the parse *[dogs in [$_{NP}$ houses and cats]]* is intuitively unnatural and should be dispreferred. The two parses in Fig. 14.7, however, have exactly the same PCFG rules and thus a PCFG will assign them the same probability.

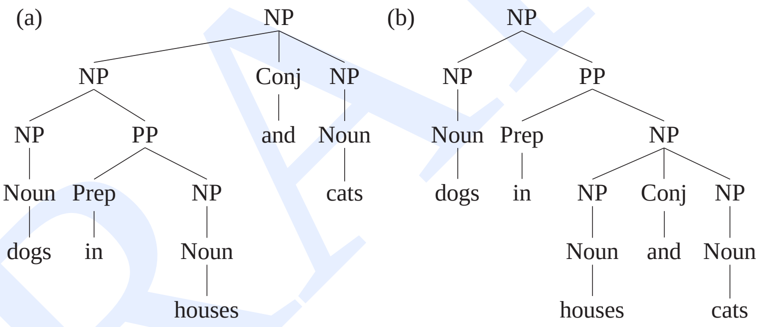


Figure 14.7 An instance of coordination ambiguity. Although the left structure is intuitively the correct one, a PCFG will assign them identically probabilities since both structure use the exact same rules. After Collins (1999).

In summary, we have shown in this section and the previous one that probabilistic context-free grammars are incapable of modeling important **structural** and **lexical** dependencies. In the next two sections we sketch current methods for augmenting PCFGs to deal with both these issues.

14.5 Improving PCFGs by Splitting and Merging Nonterminals

Let's start with the first of the two problems with PCFGs mentioned above: their inability to model structural dependencies, like the fact that NPs in subject position tend to be pronouns, where NPs in object position tend to have full lexical (non-pronominal)

SPLIT

form. How could we augment a PCFG to correctly model this fact? One idea would be to **split** the NP non-terminal into two versions: one for subjects, one for objects. Having two nodes (e.g., $NP_{subject}$ and NP_{object}) would allow us to correctly model their different distributional properties, since we would have different probabilities for the rule $NP_{subject} \rightarrow PRP$ and the rule $NP_{object} \rightarrow PRP$.

PARENT ANNOTATION

One way to implement this intuition of splits is to do **parent annotation** (Johnson, 1998), in which we annotate each node with its parent in the parse tree. Thus a node NP which is the subject of the sentence, and hence has parent S, would be annotated NPˆS, while a direct object NP, whose parent is VP, would be annotated NPˆVP. Fig. 14.8 shows an example of a tree produced by a grammar that parent annotates the phrasal non-terminals (like NP and VP).

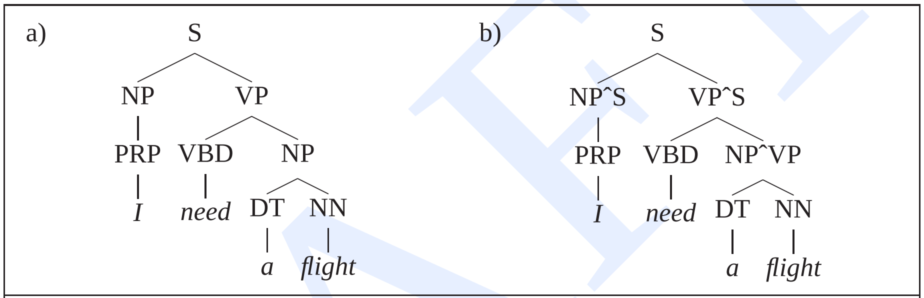


Figure 14.8 A standard PCFG parse tree (a) and one which has **parent annotation** on the nodes which aren't preterminal (b). All the non-terminal nodes (except the preterminal part-of-speech nodes) in parse (b) have been annotated with the identity of their parent.

In addition to splitting these phrasal nodes, we can also improve a PCFG by splitting the preterminal part-of-speech nodes (Klein and Manning, 2003b). For example, different kinds of adverbs (RB) tend to occur in different syntactic positions: the most common adverbs with ADVP parents are *also* and *now*, with VP parents are *n't* and *not*, and with NP parents *only* and *just*. Thus adding tags like RBˆADVP, RBˆVP, and RBˆNP can be useful in improving PCFG modeling.

Similarly, the Penn Treebank tag IN is used to mark a wide variety of parts-of-speech, including subordinating conjunctions (*while*, *as*, *if*), complementizers (*that*, *for*), and prepositions (*of*, *in*, *from*). Some of these differences can be captured by parent annotation (subordinating conjunctions occur under S, prepositions under PP), while others require specifically splitting the pre-terminal nodes. Fig. 14.9 shows an example from Klein and Manning (2003b), where even a parent annotated grammar incorrectly parses *works* as a noun in *to see if advertising works*. Splitting preterminals to allow *if* to prefer a sentential complement results in the correct verbal parse.

In order to deal with cases where parent annotation is insufficient, we can also hand-write rules that specify a particular node split based on other features of the tree. For example to distinguish between complementizer IN and subordinating conjunction IN, both of which can have the same parent, we could write rules conditioned on other aspects of the tree such as the lexical identity (the lexeme *that* is likely to be a complementizer, *as* a subordinating conjunction).

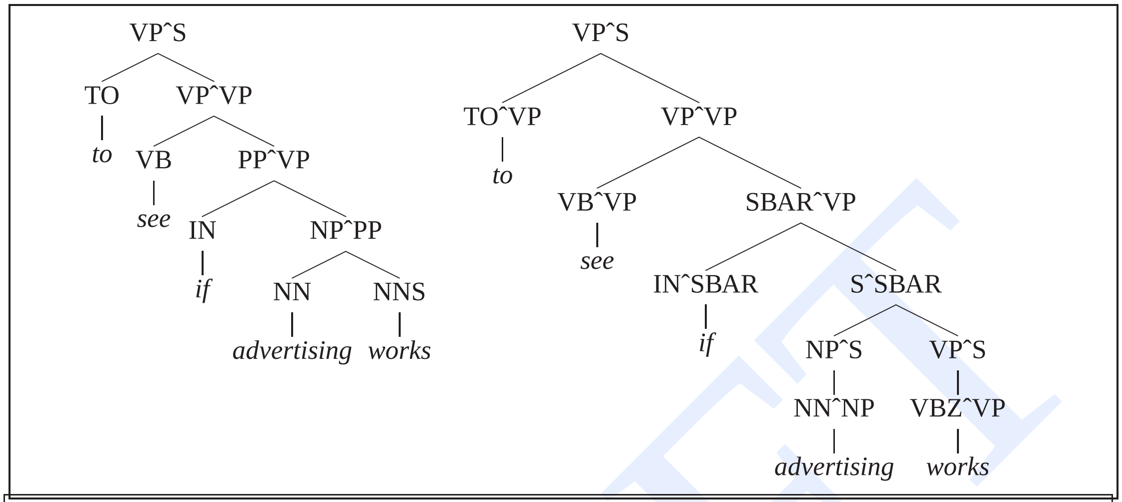


Figure 14.9 An incorrect parse even with a parent annotated parse (left). The correct parse (right), was produced by a grammar in which the pre-terminal nodes have been split, allowing the probabilistic grammar to capture the fact that *if* prefers sentential complements; adapted from Klein and Manning (2003b).

Node-splitting is not without problems; it increases the size of the grammar, and hence reduces the amount of training data available for each grammar rule, leading to overfitting. Thus it is important to split to just the correct level of granularity for a particular training set. While early models involved hand-written rules to try to find an optimal number of rules (Klein and Manning, 2003b), modern models automatically search for the optimal splits. The **split and merge** algorithm of Petrov et al. (2006), for example starts with a simple X-bar grammar, and then alternately splits the non-terminals, and merges together non-terminals, finding the set of annotated nodes which maximizes the likelihood of the training set treebank. As of the time of this writing, the performance of the Petrov et al. (2006) algorithm as the best of any known parsing algorithm on the Penn Treebank.

SPLIT AND MERGE

14.6 Probabilistic Lexicalized CFGs

The previous section showed that a simple probabilistic CKY algorithm for parsing raw PCFGs can achieve extremely high parsing accuracy if the grammar rule symbols are redesigned via automatic splits and merges.

In this section, we discuss an alternative family of models in which instead of modifying the grammar rules, we modify the probabilistic model of the parser to allow for **lexicalized** rules. The resulting family of lexicalized parsers includes the well-known **Collins parser** (Collins, 1999) and **Charniak parser** (Charniak, 1997), both of which are publicly available and widely used throughout natural language processing.

COLLINS PARSER

CHARNIAK PARSER

We saw in Sec. **??** in Ch. 12 that syntactic constituents could be associated with a lexical **head**, and we defined a **lexicalized grammar** in which each non-terminal in

LEXICALIZED GRAMMAR

the tree is annotated with its lexical head, where a rule like $VP \rightarrow VBD\ NP\ PP$ would be extended as:

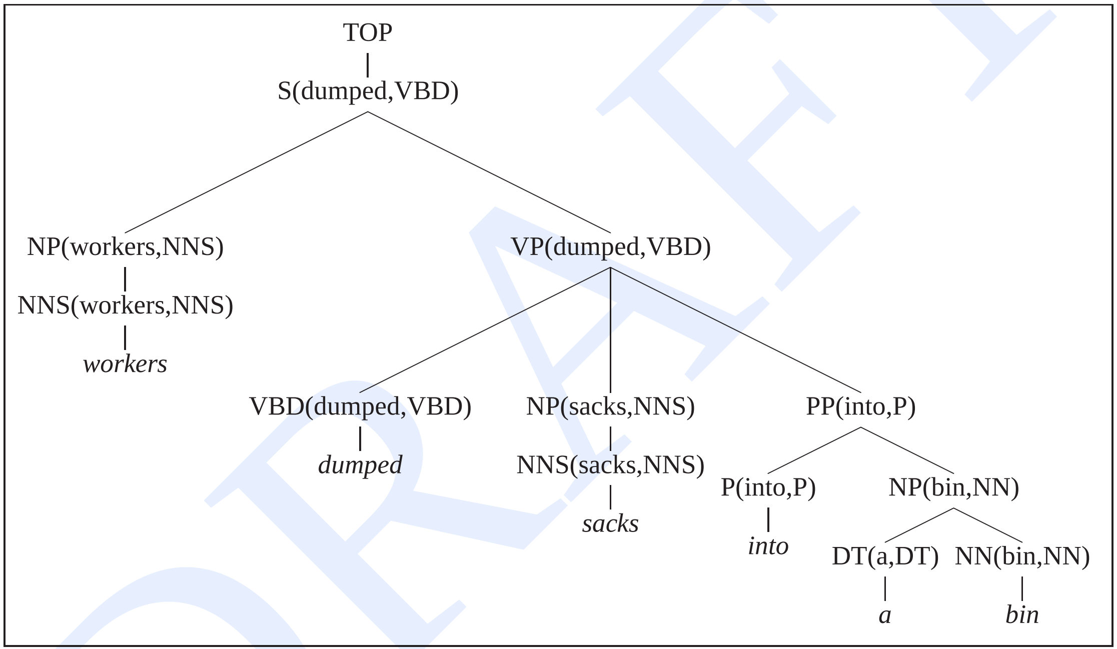
(14.21) $$VP(dumped) \rightarrow VBD(dumped)\ \ NP(sacks)\ \ PP(into)$$

HEAD TAG

In the standard type of lexicalized grammar we actually make a further extension, which is to associate the **head tag**, the part-of-speech tags of the headwords, with the nonterminal symbols as well. Each rule is thus lexicalized by both the headword and the head tag of each constituent resulting in a format for lexicalized rules like:

(14.22) $$VP(dumped,VBD) \rightarrow VBD(dumped,VBD)\ \ NP(sacks,NNS)\ \ PP(into,IN)$$

We show a lexicalized parse tree with head tags in Fig. 14.10, extended from Fig. **??**.



Internal Rules					**Lexical Rules**		
TOP	→	S(dumped,VBD)			NNS(workers,NNS)	→	workers
S(dumped,VBD)	→	NP(workers,NNS)	VP(dumped,VBD)		VBD(dumped,VBD)	→	dumped
NP(workers,NNS)	→	NNS(workers,NNS)			NNS(sacks,NNS)	→	sacks
VP(dumped,VBD)	→	VBD(dumped, VBD)	NP(sacks,NNS)	PP(into,P)	P(into,P)	→	into
PP(into,P)	→	P(into,P)	NP(bin,NN)		DT(a,DT)	→	a
NP(bin,NN)	→	DT(a,DT)	NN(bin,NN)		NN(bin,NN)	→	bin

Figure 14.10 A lexicalized tree, including head tags, for a WSJ sentence, adapted from Collins (1999). Below we show the PCFG rules that would be needed for this parse tree, internal rules on the left, and lexical rules on the right.

In order to generate such a lexicalized tree, each PCFG rule must be augmented to identify one right-hand side constituent to be the head daughter. The headword for a node is then set to the headword of its head daughter, and the head tag to the part-of-speech tag of the headword. Recall that we gave in Fig. **??** a set of hand-written rules for identifying the heads of particular constituents.

A natural way to think of a lexicalized grammar is like parent annotation, i.e. as a simple context-free grammar with many copies of each rule, one copy for each possible headword/head tag for each constituent. Thinking of a probabilistic lexicalized CFG in this way would lead to the set of simple PCFG rules shown below the tree in Fig. 14.10.

LEXICAL RULES

INTERNAL RULES

Note that Fig. 14.10 shows two kinds of rules: **lexical rules**, which express the expansion of a preterminal to a word, and **internal rules**, which express the other rule expansions. We need to distinguish these kinds of rules in a lexicalized grammar because they are associated with very different kinds of probabilities. The lexical rules are deterministic, i.e., have probability 1.0, since a lexicalized preterminal like $NN(bin,NN)$ can only expand to the word *bin*. But for the internal rules we will need to estimate probabilities.

Suppose we were to treat a probabilistic lexicalized CFG like a really big CFG that just happened to have lots of very complex non-terminals and estimate the probabilities for each rule from maximum likelihood estimates. Thus, using Eq. 14.18, the MLE estimate for the probability for the rule *P(VP(dumped,VBD) → VBD(dumped, VBD) NP(sacks,NNS) PP(into,P))* would be:

$$P(VP(dumped,VBD) \rightarrow VBD(dumped,VBD)NP(sacks,NNS)PP(into,P))$$

$$= \frac{Count(VP(dumped,VBD) \rightarrow VBD(dumped,VBD)NP(sacks,NNS)PP(into,P))}{Count(VP(dumped,VBD))} \quad (14.23)$$

But there's no way we can get good estimates of counts like those in (14.23), because they are so specific: we're very unlikely to see many (or even any) instances of a sentence with a verb phrase headed by *dumped* that has one NP argument headed by *sacks* and a PP argument headed by *into*. In other words, counts of fully lexicalized PCFG rules like this will be far too sparse and most rule probabilities will come out zero.

The idea of lexicalized parsing is to make some further independence assumptions to break down each rule, so that we would estimate the probability

$$P\,(\,VP(dumped,VBD) \rightarrow VBD(dumped,VBD)\;\; NP(sacks,NNS)\;\; PP(into,P)\,) \quad (14.24)$$

as the product of smaller independent probability estimates for which we could acquire reasonable counts. The next section summarizes one such method, the Collins parsing method.

14.6.1 The Collins Parser

Modern statistical parsers differ in exactly which independence assumptions they make. In this section we describe a simplified version of Collins's (1999) Model 1, but there are a number of other parsers that are worth knowing about; see the summary at the end of the chapter.

The first intuition of the Collins parser is to think of the right-hand side of every (internal) CFG rule as consisting of a head non-terminal, together with the non-terminals to the left of the head, and the non-terminals to the right of the head. In the abstract, we think about these rules as follows:

$$LHS \rightarrow L_n\, L_{n-1} \ldots L_1\, H\, R_1 \ldots R_{n-1}\, R_n \tag{14.25}$$

Since this is a lexicalized grammar, each of the symbols like L_1 or R_3 or H or LHS is actually a complex symbol representing the category and its head and head tag, like *VP(dumped,VP)* or *NP(sacks,NNS)*.

Now instead of computing a single MLE probability for this rule, we are going to break down this rule via a neat generative story, a slight simplification of what is called Collins Model 1. This new generative story is that given the left-hand side, we first generate the head of the rule, and then generate the dependents of the head, one by one, from the inside out. Each of these generation steps will have its own probability.

We are also going to add a special STOP non-terminal at the left and right edges of the rule; this non-terminal will allow the model to know when to stop generating dependents on a given side. We'll generate dependents on the left side of the head until we've generated STOP on the left side of the head, at which point we move to the right side of the head and start generating dependents there until we generate STOP. So it's as if we are generating a rule augmented as follows:

$$P(\,VP(dumped,VBD) \rightarrow \text{STOP}\ VBD(dumped,VBD)\ \ NP(sacks,NNS)\ \ PP(into,P)\ \ \text{STOP} \tag{14.26}$$

Let's see the generative story for this augmented rule. We're going to make use of three kinds of probabilities: P_H for generating heads, P_L for generating dependents on the left, and P_R for generating dependents on the right.

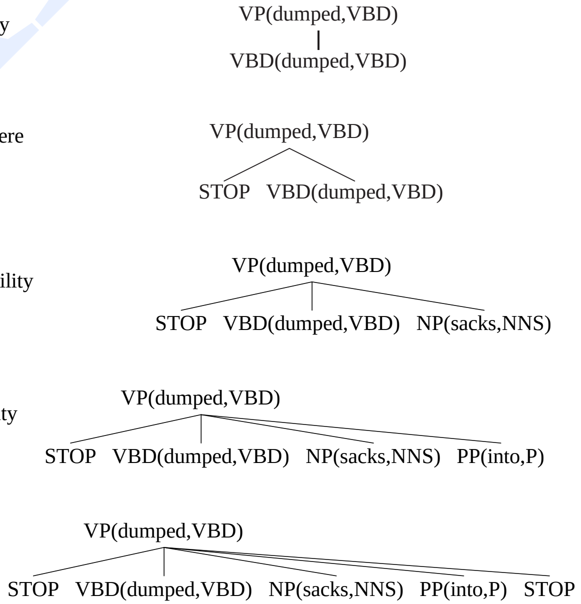
1) First generate the head VBD(dumped,VBD) with probability
P(H|LHS) = P(VBD(dumped,VBD) | VP(dumped,VBD))

2) Then generate the left dependent (which is STOP, since there isn't one) with probability
P(STOP| VP(dumped,VBD) VBD(dumped,VBD)

3) Then generate right dependent NP(sacks,NNS) with probability
P_r(NP(sacks,NNS| VP(dumped,VBD), VBD(dumped,VBD))

4) Then generate the right dependent PP(into,P) with probability
P_r(PP(into,P) | VP(dumped,VBD), VBD(dumped,VBD))

5) Finally generate the right dependent STOP with probability
P_r(STOP | VP(dumped,VBD), VBD(dumped,VBD))


VP(dumped,VBD)
VBD(dumped,VBD)
VP(dumped,VBD)
STOP VBD(dumped,VBD)
VP(dumped,VBD)
STOP VBD(dumped,VBD) NP(sacks,NNS)
VP(dumped,VBD)
STOP VBD(dumped,VBD) NP(sacks,NNS) PP(into,P)
VP(dumped,VBD)
STOP VBD(dumped,VBD) NP(sacks,NNS) PP(into,P) STOP

In summary, the probability of this rule:

$$P\,(\;VP(dumped,VBD) \rightarrow VBD(dumped,VBD)\;\; NP(sacks,NNS)PP(into,P)\;) \tag{14.27}$$

is estimated as:

$$\begin{aligned} P_H(VBD|VP,dumped) \;\; &\times\;\; P_L(STOP|VP,VBD,dumped) \\ &\times\;\; P_R(NP(sacks,NNS)|VP,VBD,dumped) \\ &\times\;\; P_R(PP(into,P)|VP,VBD,dumped) \\ &\times\;\; P_R(STOP|VP,VBD,dumped) \end{aligned} \tag{14.28}$$

Each of these probabilities can be estimated from much smaller amounts of data than the full probability in (14.27). For example, the maximum likelihood estimate for the component probability $P_R(NP(sacks,NNS)|VP,VBD,dumped)$ is:

$$\begin{aligned} &P_R(NP(sacks,NNS)|VP,VBD,dumped) = \\ &\frac{\text{Count}(\;VP(dumped,VBD)\;\;\text{with}\;\;NNS(sacks)\text{as a daughter somewhere on the right}\;)}{\text{Count}(\;VP(dumped,VBD)\;)} \end{aligned} \tag{14.29}$$

These counts are much less subject to sparsity problems than complex counts like those in (14.27).

More generally, if we use h to mean a headword together with its tag, l to mean a word+tag on the left and r to mean mean a word+tag on the right, the probability of an entire rule can be expressed as:

1. Generate the head of the phrase $H(hw,ht)$ with probability $P_H(H(hw,ht)|P,hw,ht)$
2. Generate modifiers to the left of the head with total probability:

   $$\prod_{i=1}^{n+1} P_L(L_i(lw_i,lt_i)|P,H,hw,ht)$$

   such that $L_{n+1}(lw_{n+1},lt_{n+1})$ =STOP, and we stop generating once we've generated a STOP token.
3. Generate modifiers to the right of the head with total probability:

   $$\prod_{i=1}^{n+1} P_P(R_i(rw_i,rt_i)|P,H,hw,ht)$$

   such that $R_{n+1}(rw_{n+1},rt_{n+1}) = STOP$, and we stop generating once we've generated a STOP token.

14.6.2 Advanced: Further Details of the Collins Parser

DISTANCE

The actual Collins parser models are more complex (in a couple of ways) than the simple model presented in the previous section. Collins Model 1 includes a **distance** feature. Thus instead of computing P_L and P_R as follows:

$$P_L(L_i(lw_i, lt_i)|P, H, hw, ht) \tag{14.30}$$

$$P_R(R_i(rw_i, rt_i)|P, H, hw, ht) \tag{14.31}$$

Collins Model 1 conditions also on a distance feature:

$$P_L(L_i(lw_i, lt_i)|P, H, hw, ht, distance_L(i-1)) \tag{14.32}$$

$$P_R(R_i(rw_i, rt_i)|P, H, hw, ht, distance_R(i-1)) \tag{14.33}$$

The distance measure is a function of the sequence of words *below* the previous modifiers (i.e. the words which are the yield of each modifier non-terminal we have already generated on the left). Fig. 14.11, adapted from Collins (2003) shows the computation of the probability $P(R_2(rh_2, rt_2)|P, H, hw, ht, distance_R(1))$:

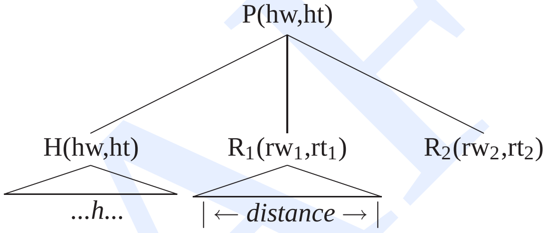


Figure 14.11 The next child R_2 is generated with probability $P(R_2(rh_2, rt_2)|P, H, hw, ht, distance_R(1))$. The distance is the yield of the previous dependent nonterminal R_1. Had there been another intervening dependent, its yield would have been included as well. Adapted from Collins (2003).

The simplest version of this distance measure is just a tuple of two binary features based on the surface string below these previous dependencies: (1) is the string of length zero? (i.e. were were no previous words generated?) (2) does the string contain a verb?

Collins Model 2 adds more sophisticated features, conditioning on subcategorization frames for each verb, and distinguishing arguments from adjuncts.

Finally, smoothing is as important for statistical parsers as it was for N-gram models. This is particularly true for lexicalized parsers, since (even using the Collins or other methods of independence assumptions) the lexicalized rules will otherwise condition on many lexical items that may never occur in training.

Consider the probability $P_R(R_i(rw_i, rt_i)|P, hw, ht)$. What do we do if a particular right-hand side constituent never occurs with this head? The Collins model addresses this problem by interpolating three backed-off models: fully lexicalized (conditioning on the headword), backing off to just the head tag, and altogether unlexicalized:

Backoff Level	$P_R(R_i(rw_i, rt_i\|...)$	Example
1	$P_R(R_i(rw_i, rt_i)\|P, hw, ht)$	P_R(NP(sacks,NNS)|VP, VBD, dumped)
2	$P_R(R_i(rw_i, rt_i)\|P, ht)$	$P_R(NP(sacks, NNS)\|VP, VBD)$
3	$P_R(R_i(rw_i, rt_i)\|P)$	$P_R(NP(sacks, NNS)\|VP)$

Similar backoff models are built also for P_L and P_H. Although we've used the word 'backoff', in fact these are not backoff models but interpolated models. The three models above are linearly interpolated, where e_1, e_2, and e_3 are the maximum likelihood estimates of the three backoff models above:

$$P_R(\ldots) = \lambda_1 e_1 + (1-\lambda_1)(\lambda_2 e_2 + (1-\lambda_2)e_3)$$

The values of $\lambda_1 and \lambda_2$ are set to implement Witten-Bell discounting (Witten and Bell, 1991) following Bikel et al. (1997).

Unknown words are dealt with in the Collins model by replacing any unknown word in the test set, and any word occurring less than 6 times in the training set, with a special UNKNOWN word token. Unknown words in the test set are assigned a part-of-speech tag in a preprocessing step by the Ratnaparkhi (1996) tagger; all other words are tagged as part of the parsing process.

The parsing algorithm for the Collins model is an extension of probabilistic CKY; see Collins (2003). Extending the CKY algorithm to handle basic lexicalized probabilities is left as an exercise for the reader.

14.7 EVALUATING PARSERS

The standard techniques for evaluating parsers and grammars are called the PARSEVAL measures, and were proposed by Black et al. (1991) based on the same ideas from signal-detection theory that we saw in earlier chapters. The intuition of the PARSEVAL metric is to measure how much the **constituents** in the hypothesis parse tree look like the constituents in a hand-labeled gold reference parse. PARSEVAL thus assumes we have a human-labeled "gold standard" parse tree for each sentence in the test set; we generally draw these gold standard parses from a treebank like the Penn Treebank.

Given these gold standard reference parses for a test set, a given constituent in a hypothesis parse C_h of a sentence s is labeled "correct" if there is a constituent in the reference parse C_r with the same starting point, ending point, and non-terminal symbol.

We can then measure the precision and recall just as we did for chunking in the previous chapter.

$$\textbf{labeled recall:} = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of correct constituents in reference parse of } s}$$

$$\textbf{labeled precision:} = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of total constituents in hypothesis parse of } s}$$

F-MEASURE

As with other uses of precision and recall, instead of reporting them separately, we often report a single number, the **F-measure** (van Rijsbergen, 1975): The F-measure is defined as:

$$F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

The β parameter is used to differentially weight the importance of recall and precision, based perhaps on the needs of an application. Values of $\beta > 1$ favor recall, while values of $\beta < 1$ favor precision. When $\beta = 1$, precision and recall are equally balanced; this is sometimes called $F_{\beta=1}$ or just F_1:

$$F_1 = \frac{2PR}{P+R} \tag{14.34}$$

The F-measure derives from a weighted harmonic mean of precision and recall. Recall that the harmonic mean of a set of numbers is the reciprocal of the arithmetic mean of the reciprocals:

$$\mathrm{HarmonicMean}(\mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3, \mathrm{a}_4, ..., \mathrm{a}_\mathrm{n}) = \frac{\mathrm{n}}{\frac{1}{\mathrm{a}_1}\frac{1}{\mathrm{a}_2}\frac{1}{\mathrm{a}_3}\cdots\frac{1}{\mathrm{a}_\mathrm{n}}} \tag{14.35}$$

and hence F-measure is

$$F = \frac{1}{\frac{1}{\alpha P} \times \frac{1}{(1-\alpha)R}} \qquad \text{or} \left(\text{with } \beta^2 = \frac{1-\alpha}{\alpha}\right) \qquad \mathrm{F} = \frac{(\beta^2+1)\mathrm{PR}}{\beta^2\mathrm{P}+\mathrm{R}} \tag{14.36}$$

We additionally use a new metric, crossing brackets, for each sentence s:

cross-brackets: the number of constituents for which the reference parse has a bracketing such as ((A B) C) but the hypothesis parse has a bracketing such as (A (B C)).

As of the time of this writing, the performance of modern parsers that are trained and tested on the Wall Street Journal treebank is somewhat higher than 90% recall, 90% precision, and about 1% cross-bracketed constituents per sentence.

For comparing parsers which use different grammars, the PARSEVAL metric includes a canonicalization algorithm for removing information likely to be grammar-specific (auxiliaries, pre-infinitival “to”, etc.) and for computing a simplified score. The interested reader should see Black et al. (1991). The canonical publicly-available implementation of the PARSEVAL metrics is called `evalb` (Sekine and Collins, 1997). EVALB

You might wonder why we don’t evaluate parsers by measuring how many *sentences* are parsed correctly, instead of measuring *constituent* accuracy. The reason we use constituents is that measuring constituents gives us a more fine-grained metric. This is especially true for long sentences, where most parsers don’t get a perfect parse. If we just measured sentence accuracy, we wouldn’t be able to distinguish between a parse that got most of the constituents wrong, and one that just got one constituent wrong.

Nonetheless, constituents are not always an optimal domain for parser evaluation. For example, using the PARSEVAL metrics requires that our parser produce trees in the exact same format as the gold standard. That means that if we want to evaluate a parser which produces different styles of parses (dependency parses, or LFG feature-structures, etc.) against say the Penn Treebank (or against another parser which produces Treebank format), we need to map the output parses into Treebank format. A related problem is that constituency may not be the level we care the most about. We might be more interested in how well the parser does at recovering grammatical dependencies (subject, object, etc), which could give us a better metric for how useful the parses would be to semantic understanding. For these purposes we can use alternative evaluation metrics based on measuring the precision and recall of labeled dependencies, where the labels indicate the grammatical relations (Lin, 1995; Carroll et al., 1998; Collins et al., 1999). Kaplan et al. (2004), for example, compared the Collins (1999) parser with the Xerox XLE parser (Riezler et al., 2002), which produces much richer semantic representations, by converting both parse trees to a dependency representation.

14.8 Advanced: Discriminative Reranking

The models we have seen of parsing so far, the PCFG parser and the Collins lexicalized parser, are generative parsers. By this we mean that the probabilistic model implemented in these parsers gives us the probability of generating a particular sentence by assigning a probability to each choice the parser could make in this generation procedure.

Generative models have some significant advantages; they are easy to train using maximum likelihood and they give us an explicit model of how different sources of evidence are combined. But generative parsing models also make it hard to incorporate arbitrary kinds of information into the probability model. This is because the probability is based on the generative derivation of a sentence; it is difficult to add features that are not local to a particular PCFG rule.

Consider for example how to represent global facts about tree structure. Parse trees in English tend to be right-branching; we'd therefore like our model to assign a higher probability to a tree which is more right-branching, all else being equal. It is also the case that heavy constituents (those with a large number of words) tend to appear later in the sentence. Or we might want to condition our parse probabilities on global facts like the identity of the speaker (perhaps some speakers are more likely to use complex relative clauses, or use the passive). Or we might want to condition on complex discourse factors across sentences. None of these kinds of global factors is trivial to incorporate into the generative models we have been considering. A simplistic model that for example makes each non-terminal dependent on how right-branching the tree is in the parse so far, or makes each NP non-terminal sensitive to the number of relative clauses the speaker or writer used in previous sentences, would result in counts that are far too sparse.

DISCRIMINATIVE RERANKING

We discussed this problem in Ch. 6, where the need for these kinds of global features motivated the use of log-linear (MEMM) models for POS tagging instead of HMMs. For parsing, there are two broad classes of discriminative models: dynamic programming approaches and two-stage models of parsing that use **discriminative reranking**. We'll discuss discriminative reranking in the rest of this section; see the end of the chapter for pointers to discriminative dynamic programming approaches.

N-BEST LIST

In the first stage of a discriminative reranking system, we can run a normal statistical parser of the type we've described so far. But instead of just producing the single best parse, we modify the parser to produce a ranked list of parses together with their probabilities. We call this ranked list of *N* parses the ***N*-best list** (the *N*-best list was first introduced in Ch. 9 when discussing multiple-pass decoding models for speech recognition). There are various ways to modify statistical parsers to produce an N-best list of parses; see the end of the chapter for pointers to the literature. For each sentence in the training set and the test set, we run this N-best parser and produce a set of N parse/probability pairs.

The second stage of a discriminative reranking model is a classifier which takes each of these sentences with their N parse/probability pairs as input, extracts some large set of features and chooses the single best parse from the N-best list. We can use any type of classifier for the reranking, such as the log-linear classifiers introduced in Ch. 6.

A wide variety of features can be used for reranking. One important feature to include is the parse probability assigned by the first-stage statistical parser. Other features might include each of the CFG rules in the tree, the number of parallel conjuncts, how heavy each constituent is, measures of how right-branching the parse tree is, how many times various tree fragments occur, bigrams of adjacent non-terminals in the tree, and so on.

The two-stage architecture has a weakness: the accuracy rate of the complete architecture can never be better than the accuracy rate of the best parse in the first-stage N-best list. This is because the reranking approach is merely choosing one of the N-best parses; even if we picked the very best parse in the list, we can't get 100% accuracy if the correct parse isn't in the list! Therefore it is important to consider the ceiling **oracle accuracy** (often measured in F-measure) of the N-best list. The oracle accuracy (F-measure) of a particular N-best list is the accuracy (F-measure) we get if we chose the parse that had the highest accuracy. We call this an **oracle** accuracy because it relies on perfect knowledge (as if from an oracle) of which parse to pick.[2] Of course it only makes sense to implement discriminative reranking if the N-best F-measure is higher than the 1-best F-measure. Luckily this is often the case; for example the Charniak (2000) parser has an F-measure of 0.897 on section 23 of the Penn Treebank, but the Charniak and Johnson (2005) algorithm for producing the 50-best parses has a much higher oracle F-measure of 0.968.

ORACLE ACCURACY

14.9 Advanced: Parser-Based Language Modeling

We said earlier that statistical parsers can take advantage of longer-distance information than N-grams, which suggests that they might do a better job at language modeling/word prediction. It turns out that if we have a very large amount of training data, a 4-gram or 5-gram grammar is nonetheless still the best way to do language modeling. But in situations where there is not enough data for such huge models, parser-based language models are beginning to be developed which have higher accuracy N-gram models.

Two common applications for language modeling are speech recognition and machine translation. The simplest way to use a statistical parser for language modeling for either of these applications is via a two-stage algorithm of the type discussed in the

[2] We introduced this same oracle idea in Ch. 9 when we talked about the **lattice error rate**.

previous section and in Sec. **??**. In the first stage, we run a normal speech recognition decoder, or machine translation decoder, using a normal *N*-gram grammar. But instead of just producing the single best transcription or translation sentence, we modify the decoder to produce a ranked *N*-best list of transcriptions/translations sentences, each one together with its probability (or, alternatively, a lattice).

Then in the second stage, we run our statistical parser and assign a parse probability to each sentence in the *N*-best list or lattice. We then rerank the sentences based on this parse probability and choose the single best sentence. This algorithm can work better than using a simple trigram grammar. For example, on the task of recognizing spoken sentences from the Wall Street Journal using this two-stage architecture, the probabilities assigned by the Charniak (2001) parser improved the word error rate by about 2 percent absolute, over a simple trigram grammar computed on 40 million words (Hall and Johnson, 2003). We can either use the parse probabilities assigned by the parser as-is, or we can linearly combine it with the original *N*-gram probability.

An alternative to the two-pass architecture, at least for speech recognition, is to modify the parser to run strictly left-to-right, so that it can incrementally give the probability of the next word in the sentence. This would allow the parser to be fit directly into the first-pass decoding pass and obviate the second-pass altogether. While a number of such left-to-right parser-based language modeling algorithms exist (Stolcke, 1995; Jurafsky et al., 1995; Roark, 2001; Xu et al., 2002), it is fair to say that it is still early days for the field of parser-based statistical language models.

14.10 Human Parsing

SENTENCE PROCESSING

Are the kinds of probabilistic parsing models we have been discussing also used by humans when they are parsing? This question lies in a field called **human sentence processing**? Recent studies suggest that there are at least two ways in which humans apply probabilistic parsing algorithms, although there is still disagreement on the details.

READING TIME

One family of studies has shown that when humans read, the predictability of a word seems to influence the **reading time**; more predictable words are read more quickly. One way of defining predictability is from simple bigram measures. For example, Scott and Shillcock (2003) had participants read sentences while monitoring their gaze with an **eye-tracker**. They constructed the sentences so that some would have a verb-noun pair with a high bigram probability (such as (14.37a)) and others a verb-noun pair with a low bigram probability (such as (14.37b)).

(14.37) a) **HIGH PROB:** One way to **avoid confusion** is to make the changes during vacation;

b) **LOW PROB:** One way to **avoid discovery** is to make the changes during vacation

They found that the higher the bigram predictability of a word, the shorter the time that participants looked at the word (the **initial-fixation duration**).

While this result only provides evidence for *N*-gram probabilities, more recent experiments have suggested that the probability of an upcoming word given the syntactic parse of the preceding sentence prefix also predicts word reading time Hale (2006), Levy (2007).

Interestingly, this effect of probability on reading time has also been shown for morphological structure; the time to recognize a word is influenced by entropy of the word and the entropy of the word's morphological paradigm Moscoso del Prado Martín et al. (2004).

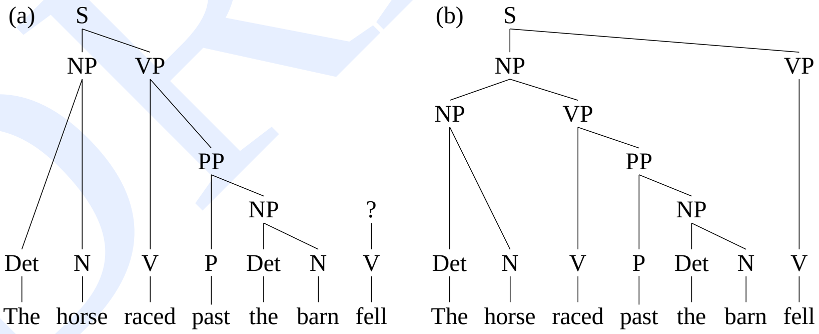
The second family of studies has examined how humans disambiguate sentences which have multiple possible parses, suggesting that humans prefer whichever parse is more probable. These studies often rely on a specific class of temporarily ambiguous sentences called **garden-path** sentences. These sentences, first described by Bever (1970), are sentences which are cleverly constructed to have three properties that combine to make them very difficult for people to parse:

GARDEN-PATH

1. They are **temporarily ambiguous**: The sentence is unambiguous, but its initial portion is ambiguous.
2. One of the two or more parses in the initial portion is somehow preferable to the human parsing mechanism.
3. But the dispreferred parse is the correct one for the sentence.

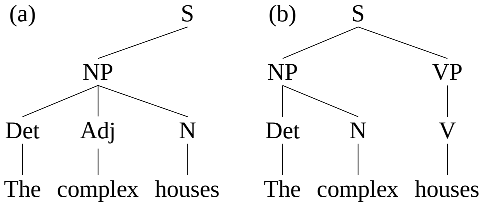
The result of these three properties is that people are "led down the garden path" toward the incorrect parse, and then are confused when they realize it's the wrong one. Sometimes this confusion is quite conscious, as in Bever's example (14.38); in fact this sentence is so hard to parse that readers often need to be shown the correct structure. In the correct structure *raced* is part of a reduced relative clause modifying *The horse*, and means "The horse [which was raced past the barn] fell"; this structure is also present in the sentence "Students taught by the Berlitz method do worse when they get to France".

(14.38) The horse raced past the barn fell.

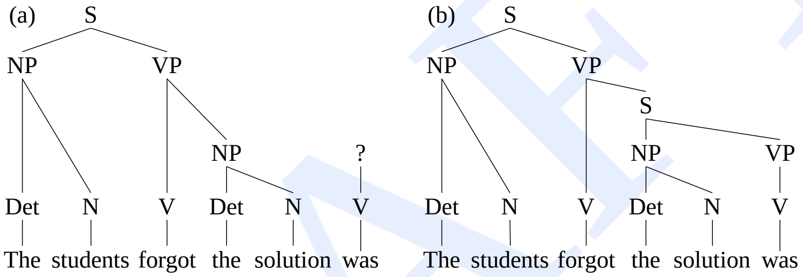

(a)
S
NP
VP
PP
NP
?
Det N V P Det N V
The horse raced past the barn fell
(b)
S
NP
VP
NP
VP
PP
NP
Det N V P Det N V
The horse raced past the barn fell

In Marti Hearst's example (14.39), subjects often misparse the verb *houses* as a noun (analyzing *the complex houses* as a noun phrase, rather than a noun phrase and a verb). Other times the confusion caused by a garden-path sentence is so subtle that it can only be measured by a slight increase in reading time. Thus in example (14.40) readers often mis-parse *the solution* as the direct object of *forgot* rather than as the subject of an embedded sentence. This mis-parse is subtle, and is only noticeable because experimental participants take longer to read the word *was* than in control sentences. This "mini-garden-path" effect at the word *was* suggests that subjects had chosen the direct object parse and had to re-analyze or rearrange their parse now that they realize they are in a sentential complement.

(14.39) The complex houses married and single students and their families.



(14.40) The student forgot the solution was in the back of the book.



While many factors seem to play a role in these preferences for a particular (incorrect) parse, at least one factor seems to be syntactic probabilities, especially lexicalized (subcategorization) probabilities. For example, the probability of the verb *forgot* taking a direct object ($VP \rightarrow V\ NP$) is higher than the probability of it taking a sentential complement ($VP \rightarrow V\ S$); this difference causes readers to expect a direct object after *forget* and be surprised (longer reading times) when they encounter a sentential complement. By contrast, a verb which prefers a sentential complement (like *hope*) didn't cause extra reading time at *was*.

Similarly, the garden path in (14.39) may be caused by the fact that $P(houses|Noun) > P(houses|Verb)$ and $P(complex|Adjective) > P(complex|Noun)$, and the garden path in (14.38) at least partially by the low probability of the reduced relative clause construction.

Besides grammatical knowledge, human parsing is affected by many other factors which we will describe later, including resource constraints (such as memory limitations, to be discussed in Ch. 15), thematic structure (such as whether a verb expects semantic *agents* or *patients*, to be discussed in Ch. 19) and discourse constraints (Ch. 21).

14.11 Summary

This chapter has sketched the basics of **probabilistic** parsing, concentrating on **probabilistic context-free grammars** and **probabilistic lexicalized context-free grammars**.

- Probabilistic grammars assign a probability to a sentence or string of words, while attempting to capture more sophisticated syntactic information than the *N*-gram grammars of Ch. 4.
- A **probabilistic context-free grammar** (**PCFG**) is a context-free grammar in which every rule is annotated with the probability of choosing that rule. Each PCFG rule is treated as if it were **conditionally independent**; thus the probability of a sentence is computed by **multiplying** the probabilities of each rule in the parse of the sentence.
- The probabilistic CKY (**Cocke-Kasami-Younger**) algorithm is a probabilistic version of the CKY parsing algorithm. There are also probabilistic versions of other parsers like the Earley algorithm.
- PCFG probabilities can be learning by counting in a **parsed corpus**, or by parsing a corpus. The **Inside-Outside** algorithm is a way of dealing with the fact that the sentences being parsed are ambiguous.
- Raw PCFGs suffer from poor independence assumptions between rules and lack of sensitivity to lexical dependencies.
- One way to deal with this problem is to split and merge non-terminals (automatically or by hand).
- **Probabilistic lexicalized CFG**s are another solution to this problem in which the basic PCFG model is augmented with a **lexical head** for each rule. The probability of a rule can then be conditioned on the lexical head or nearby heads.
- Parsers for lexicalized PCFGs (like the Charniak and Collins parsers) are based on extensions to probabilistic CKY parsing.
- Parsers are evaluated using three metrics: **labeled recall**, **labeled precision**, and **cross-brackets**.
- There is evidence based on **garden-path sentences** and other on-line sentence-processing experiments that the human parser uses some kinds of probabilistic information about grammar.

Bibliographical and Historical Notes

Many of the formal properties of probabilistic context-free grammars were first worked out by Booth (1969) and Salomaa (1969). Baker (1979) proposed the Inside-Outside algorithm for unsupervised training of PCFG probabilities, and used a CKY-style parsing algorithm to compute inside probabilities. Jelinek and Lafferty (1991) extended the CKY algorithm to compute probabilities for prefixes. Stolcke (1995) drew on both of these algorithms in adapting the Earley algorithm to use with PCFGs.

A number of researchers starting in the early 1990s worked on adding lexical dependencies to PCFGs, and on making PCFG rule probabilities more sensitive to surrounding syntactic structure. For example Schabes et al. (1988) and Schabes (1990) presented early work on the use of heads. Many papers on the use of lexical dependencies were first presented at the DARPA Speech and Natural Language Workshop in

June, 1990. A paper by Hindle and Rooth (1990) applied lexical dependencies to the problem of attaching prepositional phrases; in the question session to a later paper Ken Church suggested applying this method to full parsing (Marcus, 1990). Early work on such probabilistic CFG parsing augmented with probabilistic dependency information includes Magerman and Marcus (1991), Black et al. (1992), Bod (1993), and Jelinek et al. (1994), in addition to Collins (1996), Charniak (1997), and Collins (1999) discussed above. Other recent PCFG parsing models include Klein and Manning (2003a) and Petrov et al. (2006). .

This early lexical probabilistic work led initially to work focused on solving specific parsing problems like preposition-phrase attachment, using methods including Transformation Based Learning (TBL) (Brill and Resnik, 1994), Maximum Entropy (Ratnaparkhi et al., 1994), Memory-Based Learning (Zavrel and Daelemans, 1997), log-linear models (Franz, 1997), decision trees using semantic distance between heads (computed from WordNet) (Stetina and Nagao, 1997), and Boosting (Abney et al., 1999).

SUPERTAGGING

Another direction extended the lexical probabilistic parsing work to build probabilistic formulations of grammar other than PCFGs, such as probabilistic TAG grammar (Resnik, 1992; Schabes, 1992), based on the TAG grammars discussed in Ch. 12, probabilistic LR parsing (Briscoe and Carroll, 1993), and probabilistic link grammar (Lafferty et al., 1992). An approach to probabilistic parsing called **supertagging** extends the part-of-speech tagging metaphor to parsing by using very complex tags that are in fact fragments of lexicalized parse trees (Bangalore and Joshi, 1999; Joshi and Srinivas, 1994), based on the lexicalized TAG grammars of Schabes et al. (1988). For example the noun *purchase* would have a different tag as the first noun in a noun compound (where it might be on the left of a small tree dominated by Nominal) than as the second noun (where it might be on the right). Supertagging has also been applied to CCG parsing and HPSG parsing (Clark and Curran, 2004a; Matsuzaki et al., 2007; Blunsom and Baldwin, 2006). Non-supertagging statistical parsers for CCG include Hockenmaier and Steedman (2002).

Goodman (1997), Abney (1997), and Johnson et al. (1999) gave early discussions of probabilistic treatments of feature-based grammars. Other recent work on building statistical models of feature-based grammar formalisms like HPSG and LFG includes Riezler et al. (2002), Kaplan et al. (2004), and Toutanova et al. (2005).

We mentioned earlier that discriminative approaches to parsing fall into the two broad categories of dynamic programming methods and discriminative reranking methods. Recall that discriminative reranking approaches require *N*-best parses. Parsers based on A* search can easily be modified to generate *N*-best lists just by continuing the search past the first-best parse (Roark, 2001). Dynamic programming algorithms like the ones described in this chapter can be modified by eliminating the dynamic programming and using heavy pruning (Collins, 2000; Collins and Koo, 2005; Bikel, 2004), or via new algorithms (Jiménez and Marzal, 2000; Gildea and Jurafsky, 2002; Charniak and Johnson, 2005; Huang and Chiang, 2005), some adapted from speech recognition algorithms such as Schwartz and Chow (1990) (see Sec. **??**).

By contrast, in dynamic programming methods, instead of outputting and then reranking an *N*-best list, the parses are represented compactly in a chart, and log-linear and other methods are applied for decoding directly from the chart. Such modern methods include Johnson (2001), Clark and Curran (2004b), and Taskar et al. (2004). Other reranking developments include changing the optimization criterion (Titov and Henderson, 2006).

Another important recent area of research is dependency parsing; algorithms include Eisner's bilexical algorithm (Eisner, 1996b, 1996a, 2000), maximum spanning tree approaches (using on-line learning) (McDonald et al., 2005b, 2005a), and approaches based on building classifiers for parser actions (Kudo and Matsumoto, 2002; Yamada and Matsumoto, 2003; Nivre et al., 2006; Titov and Henderson, 2007). A distinction is usually made between projective and **non-projective dependencies**. Non-projective dependencies are those in which the dependency lines cross; this is not very common in English, but is very common in many languages with more free word order. Non-projective dependency algorithms include McDonald et al. (2005a) and Nivre (2007). The Klein-Manning parser combines dependency and constituency information (Klein and Manning, 2003c).

NON-PROJECTIVE DEPENDENCIES

Manning and Schütze (1999) has an extensive coverage of probabilistic parsing. Collins' (1999) dissertation includes a very readable survey of the field and introduction to his parser.

The field of grammar induction is closely related to statistical parsing, and a parser is often used as part of a grammar induction algorithm. One of the earliest statistical works in grammar induction was Horning (1969), who showed that PCFGs could be induced without negative evidence. Early modern probabilistic grammar work showed that simply using EM was insufficient (Lari and Young, 1990; Carroll and Charniak, 1992). Recent probabilistic work such as Yuret (1998), Clark (2001), Klein and Manning (2002), and Klein and Manning (2004), are summarized in Klein (2005) and Adriaans and van Zaanen (2004). Work since that summary includes Smith and Eisner (2005), Haghighi and Klein (2006), and Smith and Eisner (2007).

Exercises

14.1 Implement the CKY algorithm.

14.2 Modify the algorithm for conversion to CNF from Ch. 13 to correctly handle rule probabilities. Make sure that the resulting CNF assigns the same total probability to each parse tree.

14.3 Recall that Exercise **??** asked you to update the CKY algorithm to handles unit productions directly rather than converting them to CNF. Extend this change to probabilistic CKY.

14.4 Fill out the rest of the probabilistic CKY chart in Fig. 14.4.

14.5 Sketch out how the CKY algorithm would have to be augmented to handle lexicalized probabilities.

14.6 Implement your lexicalized extension of the CKY algorithm.

14.7 Implement the PARSEVAL metrics described in Sec. 14.7. Next either use a treebank or create your own hand-checked parsed testset. Now use your CFG (or other) parser and grammar and parse the testset and compute labeled recall, labeled precision, and cross-brackets.

Abney, S. P. (1997). Stochastic attribute-value grammars. *Computational Linguistics*, *23*(4), 597–618.

Abney, S. P., Schapire, R. E., and Singer, Y. (1999). Boosting applied to tagging and PP attachment. In *EMNLP/VLC-99*, College Park, MD, pp. 38–45.

Adriaans, P. and van Zaanen, M. (2004). Computational grammar induction for linguists. *Grammars; special issue with the theme "Grammar Induction"*, *7*, 57–68.

Baker, J. K. (1979). Trainable grammars for speech recognition. In Klatt, D. H. and Wolf, J. J. (Eds.), *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pp. 547–550.

Bangalore, S. and Joshi, A. K. (1999). Supertagging: An approach to almost parsing. *Computational Linguistics*, *25*(2), 237–265.

Bever, T. G. (1970). The cognitive basis for linguistic structures. In Hayes, J. R. (Ed.), *Cognition and the Development of Language*, pp. 279–352. Wiley.

Bikel, D. M. (2004). Intricacies of Collins' parsing model. *Computational Linguistics*, *30*(4), 479–511.

Bikel, D. M., Miller, S., Schwartz, R., and Weischedel, R. (1997). Nymble: a high-performance learning name-finder. In *Proceedings of ANLP-97*, pp. 194–201.

Black, E., Abney, S. P., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J. L., Liberman, M. Y., Marcus, M. P., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings DARPA Speech and Natural Language Workshop*, Pacific Grove, CA, pp. 306–311. Morgan Kaufmann.

Black, E., Jelinek, F., Lafferty, J. D., Magerman, D. M., Mercer, R. L., and Roukos, S. (1992). Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings DARPA Speech and Natural Language Workshop*, Harriman, NY, pp. 134–139. Morgan Kaufmann.

Blunsom, P. and Baldwin, T. (2006). Multilingual deep lexical acquisition for hpsgs via supertagging. In *EMNLP 2006*.

Bod, R. (1993). Using an annotated corpus as a stochastic grammar. In *EACL-93*, pp. 37–44.

Booth, T. L. (1969). Probabilistic representation of formal languages. In *IEEE Conference Record of the 1969 Tenth Annual Symposium on Switching and Automata Theory*, pp. 74–81.

Booth, T. L. and Thompson, R. A. (1973). Applying probability measures to abstract languages. *IEEE Transactions on Computers*, *C-22*(5), 442–450.

Brill, E. and Resnik, P. (1994). A rule-based approach to prepositional phrase attachment disambiguation. In *COLING-94*, Kyoto, pp. 1198–1204.

Briscoe, T. and Carroll, J. (1993). Generalized Probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, *19*(1), 25–59.

Carroll, G. and Charniak, E. (1992). Two experiments on learning probabilistic dependency grammars from corpora. Tech. rep. CS-92-16, Brown University.

Carroll, J., Briscoe, T., and Sanfilippo, A. (1998). Parser evaluation: a survey and a new proposal. In *LREC-98*, Granada, Spain, pp. 447–454.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine *n*-best parsing and MaxEnt discriminative reranking. In *ACL-05*, Ann Arbor.

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *AAAI-97*, Menlo Park, pp. 598–603. AAAI Press.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *Proceedings of the 1st Annual Meeting of the North American Chapter of the ACL (NAACL'00)*, Seattle, Washington, pp. 132–139.

Charniak, E. (2001). Immediate-head parsing for language models. In *ACL-01*, Toulouse, France.

Chelba, C. and Jelinek, F. (2000). Structured language modeling. *Computer Speech and Language*, *14*, 283–332.

Clark, A. (2001). The unsupervised induction of stochastic context-free grammars using distributional clustering. In *CoNLL-01*.

Clark, S. and Curran, J. R. (2004a). The importance of supertagging for wide-coverage CCG parsing. In *COLING-04*, pp. 282–288.

Clark, S. and Curran, J. R. (2004b). Parsing the WSJ using CCG and Log-Linear Models. In *ACL-04*, pp. 104–111.

Collins, M. and Koo, T. (2005). Discriminative reranking for natural language parsing. *Computational Linguistics*, *31*(1), 25–69.

Collins, M. (1996). A new statistical parser based on bigram lexical dependencies. In *ACL-96*, Santa Cruz, California, pp. 184–191.

Collins, M. (1999). *Head-driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

Collins, M. (2000). Discriminative reranking for natural language parsing. In *ICML 2000*, Stanford, CA, pp. 175–182.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational Linguistics*, *29*(4), 589–637.

Collins, M., Hajič, J., Ramshaw, L. A., and Tillmann, C. (1999). A statistical parser for Czech. In *ACL-99*, College Park, MA, pp. 505–512.

Eisner, J. (1996a). An empirical comparison of probability models for dependency grammar. Tech. rep. IRCS-96-11, Institute for Research in Cognitive Science, Univ. of Pennsylvania.

Eisner, J. (1996b). Three new probabilistic models for dependency parsing: An exploration. In *COLING-96*, Copenhagen, pp. 340–345.

Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In Bunt, H. and Nijholt, A. (Eds.), *Advances in Probabilistic and Other Parsing Technologies*, pp. 29–62. Kluwer.

Francis, H. S., Gregory, M. L., and Michaelis, L. A. (1999). Are lexical subjects deviant?. In *CLS-99*. University of Chicago.

Franz, A. (1997). Independence assumptions considered harmful. In *ACL/EACL-97*, Madrid, Spain, pp. 182–189.

Gildea, D. and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational Linguistics*, *28*(3), 245–288.

Givón, T. (1990). *Syntax: A functional typological introduction*. John Benjamins, Amsterdam.

Goodman, J. (1997). Probabilistic feature grammars. In *Proceedings of the International Workshop on Parsing Technology*.

Haghighi, A. and Klein, D. (2006). Prototype-driven grammar induction. In *COLING/ACL 2006*, pp. 881–888.

Hale, J. (2006). Uncertainty about the rest of the sentence. *Cognitive Science*, *30*(4), 609–642.

Hall, K. and Johnson, M. (2003). Language modeling using efficient best-first bottom-up parsing. In *IEEE ASRU-03*, pp. 507–512.

Hindle, D. and Rooth, M. (1990). Structural ambiguity and lexical relations. In *Proceedings DARPA Speech and Natural Language Workshop*, Hidden Valley, PA, pp. 257–262. Morgan Kaufmann.

Hindle, D. and Rooth, M. (1991). Structural ambiguity and lexical relations. In *Proceedings of the 29th ACL*, Berkeley, CA, pp. 229–236.

Hockenmaier, J. and Steedman, M. (2002). Generative models for statistical parsing with Combinatory Categorial Grammar. In *ACL-02*, Philadelphia, PA.

Horning, J. J. (1969). *A study of grammatical inference*. Ph.D. thesis, Stanford University.

Huang, L. and Chiang, D. (2005). Better k-best parsing. In *IWPT-05*, pp. 53–64.

Jelinek, F. and Lafferty, J. D. (1991). Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, *17*(3), 315–323.

Jelinek, F., Lafferty, J. D., Magerman, D. M., Mercer, R. L., Ratnaparkhi, A., and Roukos, S. (1994). Decision tree parsing using a hidden derivation model. In *ARPA Human Language Technologies Workshop*, Plainsboro, N.J., pp. 272–277. Morgan Kaufmann.

Jiménez, V. M. and Marzal, A. (2000). Computation of the *n* best parse trees for weighted and stochastic context-free grammars. In *Advances in Pattern Recognition: Proceedings of the Joint IAPR International Workshops, SSPR 2000 and SPR 2000*, Alicante, Spain, pp. 183–192. Springer.

Johnson, M. (1998). PCFG models of linguistic tree representations. *Computational Linguistics*, *24*(4), 613–632.

Johnson, M. (2001). Joint and conditional estimation of tagging and parsing models. In *ACL-01*, pp. 314–321.

Johnson, M., Geman, S., Canon, S., Chi, Z., and Riezler, S. (1999). Estimators for stochastic “unification-based” grammars. In *ACL-99*, pp. 535–541.

Joshi, A. K. and Srinivas, B. (1994). Disambiguation of super parts of speech (or supertags): Almost parsing. In *COLING-94*, Kyoto, pp. 154–160.

Jurafsky, D., Wooters, C., Tajchman, G., Segal, J., Stolcke, A., Fosler, E., and Morgan, N. (1995). Using a stochastic context-free grammar as a language model for speech recognition. In *IEEE ICASSP-95*, pp. 189–192. IEEE.

Kaplan, R. M., Riezler, S., King, T. H., Maxwell, J. T., Vasserman, A., and Crouch, R. (2004). Speed and accuracy in shallow and deep stochastic parsing. In *HLT-NAACL-04*.

Klein, D. (2005). *The unsupervised learning of Natural Language Structure*. Ph.D. thesis, Stanford University.

Klein, D. and Manning, C. D. (2001). Parsing and hypergraphs. In *The Seventh Internation Workshop on Parsing Technologies*.

Klein, D. and Manning, C. D. (2002). A generative constituent-context model for improved grammar induction. In *ACL-02*.

Klein, D. and Manning, C. D. (2003a). A* parsing: Fast exact Viterbi parse selection. In *HLT-NAACL-03*.

Klein, D. and Manning, C. D. (2003b). Accurate unlexicalized parsing. In *HLT-NAACL-03*.

Klein, D. and Manning, C. D. (2003c). Fast exact inference with a factored model for natural language parsing. In Becker, S., Thrun, S., and Obermayer, K. (Eds.), *Advances in Neural Information Processing Systems 15*. MIT Press.

Klein, D. and Manning, C. D. (2004). Corpus-based induction of syntactic structure: Models of dependency and constituency. In *ACL-04*.

Kudo, T. and Matsumoto, Y. (2002). Japanese dependency analysis using cascaded chunking. In *CoNLL-02*, pp. 63–69.

Lafferty, J. D., Sleator, D., and Temperley, D. (1992). Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the 1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language*.

Lari, K. and Young, S. J. (1990). The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, *4*, 35–56.

Levy, R. (2007). Expectation-based syntactic comprehension. *Cognition*. In press.

Lin, D. (1995). A dependency-based method for evaluating broad-coverage parsers. In *IJCAI-95*, Montreal, pp. 1420–1425.

Magerman, D. M. and Marcus, M. P. (1991). Pearl: A probabilistic chart parser. In *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics*, Berlin, Germany.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Marcus, M. P. (1990). Summary of session 9: Automatic acquisition of linguistic structure. In *Proceedings DARPA Speech and Natural Language Workshop*, Hidden Valley, PA, pp. 249–250. Morgan Kaufmann.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, *19*(2), 313–330.

Matsuzaki, T., Miyao, Y., and ichi Tsujii, J. (2007). Efficient hpsg parsing with supertagging and cfg-filtering. In *IJCAI-07*.

McDonald, R., Pereira, F. C. N., Ribarov, K., and Hajič, J. (2005a). Non-projective dependency parsing using spanning tree algorithms. In *HLT-EMNLP-05*.

McDonald, R., Crammer, K., and Pereira, F. C. N. (2005b). Online large-margin training of dependency parsers. In *ACL-05*, Ann Arbor, pp. 91–98.

Moscoso del Prado Martín, F., Kostic, A., and Baayen, R. H. (2004). Putting the bits together: An information theoretical perspective on morphological processing. *Cognition*, *94*(1), 1–18.

Ney, H. (1991). Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing*, *39*(2), 336–340.

Nivre, J. (2007). Incremental non-projective dependency parsing. In *NAACL-HLT 07*.

Nivre, J., Hall, J., and Nilsson, J. (2006). Maltparser: A data-driven parser-generator for dependency parsing. In *LREC-06*, pp. 2216–2219.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *COLING/ACL 2006*, Sydney, Australia, pp. 433–440. ACL.

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. In *EMNLP 1996*, Philadelphia, PA, pp. 133–142.

Ratnaparkhi, A., Reynar, J. C., and Roukos, S. (1994). A Maximum Entropy model for prepositional phrase attachment. In *ARPA Human Language Technologies Workshop*, Plainsboro, N.J., pp. 250–255.

Resnik, P. (1992). Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the 14th International Conference on Computational Linguistics*, Nantes, France, pp. 418–424.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., III, J. T. M., and Johnson, M. (2002). Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In *ACL-02*, Philadelphia, PA.

Roark, B. (2001). Probabilistic top-down parsing and language modeling. *Computational Linguistics*, *27*(2), 249–276.

Salomaa, A. (1969). Probabilistic and weighted grammars. *Information and Control*, *15*, 529–544.

Schabes, Y. (1990). *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA†.

Schabes, Y. (1992). Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th International Conference on Computational Linguistics*, Nantes, France, pp. 426–433.

Schabes, Y., Abeillé, A., and Joshi, A. K. (1988). Parsing strategies with 'lexicalized' grammars: Applications to Tree Adjoining Grammars. In *COLING-88*, Budapest, pp. 578–583.

Schwartz, R. and Chow, Y.-L. (1990). The N-best algorithm: An efficient and exact procedure for finding the N most likely sentence hypotheses. In *IEEE ICASSP-90*, Vol. 1, pp. 81–84. IEEE.

Scott, M. and Shillcock, R. (2003). Eye movements reveal the on-line computation of lexical probabilities during reading. *Psychological Science*, *14*(6), 648–652.

Sekine, S. and Collins, M. (1997). The evalb software. `http://cs.nyu.edu/cs/projects/proteus/evalb`.

Smith, D. A. and Eisner, J. (2007). Bootstrapping feature-rich dependency parsers with entropic priors. In *EMNLP/CoNLL 2007*, Prague, pp. 667–677.

Smith, N. A. and Eisner, J. (2005). Guiding unsupervised grammar induction using contrastive estimation. In *IJCAI Workshop on Grammatical Inference Applications*, Edinburgh, pp. 73–82.

Stetina, J. and Nagao, M. (1997). Corpus based PP attachment ambiguity resolution with a semantic dictionary. In Zhou, J. and Church, K. W. (Eds.), *Proceedings of the Fifth Workshop on Very Large Corpora*, Beijing, China, pp. 66–80.

Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, *21*(2), 165–202.

Taskar, B., Klein, D., Collins, M., Koller, D., and Manning, C. D. (2004). Max-margin parsing. In *EMNLP 2004*.

Titov, I. and Henderson, J. (2006). Loss minimization in parse reranking. In *EMNLP 2006*.

Titov, I. and Henderson, J. (2007). A latent variable model for generative dependency parsing. In *IWPT-07*.

Toutanova, K., Manning, C. D., Flickinger, D., and Oepen, S. (2005). Stochastic HPSG Parse Disambiguation using the Redwoods Corpus. *Research on Language & Computation*, *3*(1), 83–105.

van Rijsbergen, C. J. (1975). *Information Retrieval*. Butterworths, London.

Witten, I. H. and Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, *37*(4), 1085–1094.

Xu, P., Chelba, C., and Jelinek, F. (2002). A study on richer syntactic dependencies for structured language modeling. In *ACL-02*, pp. 191–198.

Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In Noord, G. V. (Ed.), *IWPT-03*, pp. 195–206.

Yuret, D. (1998). *Discovery of Linguistic Relations Using Lexical Attraction*. Ph.D. thesis, MIT.

Zavrel, J. and Daelemans, W. (1997). Memory-based learning: Using similarity for smoothing. In *ACL/EACL-97*, Madrid, Spain, pp. 436–443.

15 LANGUAGE AND COMPLEXITY

This is the dog, that worried the cat, that killed the rat, that ate the malt, that lay in the house that Jack built.

Mother Goose, *The House that Jack Built*

This is the malt that the rat that the cat that the dog worried killed ate.

Victor H. Yngve (1960)

Much of the humor in musical comedy and comic operetta comes from entwining the main characters in fabulously complicated plot twists. Casilda, the daughter of the Duke of Plaza-Toro in Gilbert and Sullivan's *The Gondoliers*, is in love with her father's attendant Luiz. Unfortunately, Casilda discovers she has already been married (by proxy) as a babe of six months to "the infant son and heir of His Majesty the immeasurably wealthy King of Barataria". It is revealed that this infant son was spirited away by the Grand Inquisitor and raised by a "highly respectable gondolier" in Venice as a gondolier. The gondolier had a baby of the same age and could never remember which child was which, and so Casilda was in the unenviable position, as she puts it, of "being married to one of two gondoliers, but it is impossible to say which". By way of consolation, the Grand Inquisitor informs her that "such complications frequently occur".

Luckily, such complications don't frequently occur in natural language. Or do they? In fact there are sentences that are so complex that they are hard to understand, such as Yngve's sentence above, or the sentence:

"*The Republicans who the senator who she voted for chastised were trying to cut all benefits for veterans*".

Studying such sentences, and more generally understanding what level of complexity tends to occur in natural language, is an important area of language processing. Complexity plays an important role, for example, in deciding when we need to use a particular formal mechanism. Formal mechanisms like finite automata, Markov models, transducers, phonological rewrite rules, and context-free grammars, can be described

POWER
COMPLEXITY

in terms of their **power**, or equivalently in terms of the **complexity** of the phenomena that they can describe. This chapter introduces the Chomsky hierarchy, a theoretical tool that allows us to compare the expressive power or complexity of these different formal mechanisms. With this tool in hand, we summarize arguments about the correct formal power of the syntax of natural languages, in particular English but also including a famous Swiss dialect of German that has the interesting syntactic property called **cross-serial dependencies**. This property has been used to argue that context-free grammars are insufficiently powerful to model the morphology and syntax of natural language.

In addition to using complexity as a metric for understanding the relation between natural language and formal models, the field of complexity is also concerned with what makes individual constructions or sentences hard to understand. For example we saw above that certain **nested** or **center-embedded** sentences are difficult for people to process. Understanding what makes some sentences difficult for people to process is an important part of understanding human parsing.

15.1 The Chomsky Hierarchy

GENERATIVE POWER

How are automata, context-free grammars, and phonological rewrite rules related? What they have in common is that each describes a **formal language**, which we have seen is a set of strings over a finite alphabet. But the kind of grammars we can write with each of these formalism are of different **generative power**. One grammar is of greater generative power or **complexity** than another if it can define a language that the other cannot define. We will show, for example, that a context-free grammar can be used to describe formal languages that cannot be described with a finite-state automaton.

CHOMSKY HIERARCHY

It is possible to construct a hierarchy of grammars, where the set of languages describable by grammars of greater power subsumes the set of languages describable by grammars of lesser power. There are many possible such hierarchies; the one that is most commonly used in computational linguistics is the **Chomsky hierarchy** (Chomsky, 1959), which includes four kinds of grammars: Fig. 15.1 shows the four grammars in the Chomsky hierarchy as well as a useful fifth type, the *mildly context-sensitive* languages.

This decrease in the generative power of languages from the most powerful to the weakest can in general be accomplished by placing constraints on the way the grammar rules are allowed to be written. Fig. 15.2 shows the five types of grammars in the extended Chomsky hierarchy, defined by the constraints on the form that rules must take. In these examples, A is a single non-terminal, and α, β, and γ are arbitrary strings of terminal and non-terminal symbols. They may be empty unless this is specifically disallowed below. x is an arbitrary string of terminal symbols.

RECURSIVELY ENUMERABLE

Turing-equivalent, **Type 0** or **unrestricted** grammars have no restrictions on the form of their rules, except that the left-hand side cannot be the empty string ϵ. Any (non-null) string can be written as any other string (or as ϵ). Type 0 grammars characterize the **recursively enumerable** languages, that is, those whose strings can be listed

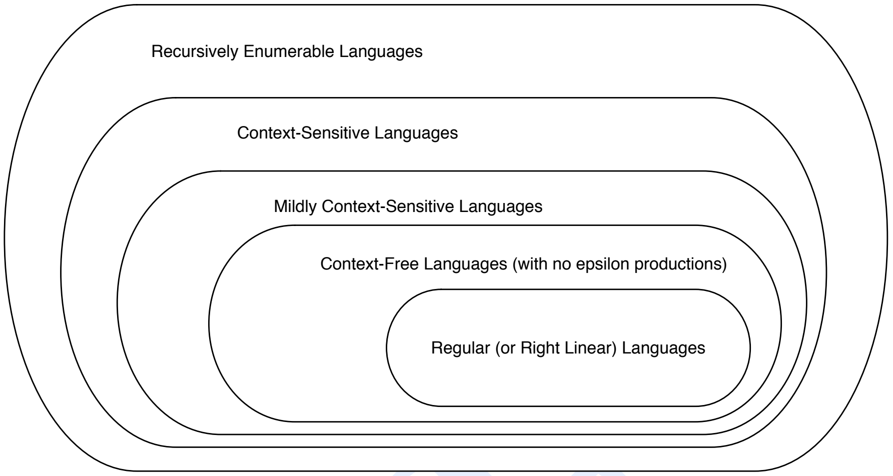

Recursively Enumerable Languages
Context-Sensitive Languages
Mildly Context-Sensitive Languages
Context-Free Languages (with no epsilon productions)
Regular (or Right Linear) Languages

Figure 15.1 A Venn diagram of the four languages on the Chomsky Hierarchy, augmented with a fifth class, the mildly context-sensitive languages.

Type	Common Name	Rule Skeleton	Linguistic Example
0	Turing Equivalent	$\alpha \rightarrow \beta$, s.t. $\alpha \neq \epsilon$	HPSG, LFG, Minimalism
1	Context Sensitive	$\alpha A\beta \rightarrow \alpha\gamma\beta$, s.t. $\gamma \neq \epsilon$	
-	Mildly Context Sensitive		TAG, CCG
2	Context Free	$A \rightarrow \gamma$	Phrase Structure Grammars
3	Regular	$A \rightarrow xB$ or $A \rightarrow x$	Finite State Automata

Figure 15.2 The Chomsky Hierarchy, augmented by the mildly context-sensitive grammars.

(enumerated) by a Turing Machine.

CONTEXT-SENSITIVE

Context-sensitive grammars have rules that rewrite a non-terminal symbol A in the context $\alpha A\beta$ as any non-empty string of symbols. They can be either written in the form $\alpha A\beta \rightarrow \alpha\gamma\beta$ or in the form $A \rightarrow \gamma / \alpha \underline{\hspace{1.5em}} \beta$. We have seen this latter version in the Chomsky-Halle representation of phonological rules (Chomsky and Halle, 1968) like this flapping rule:

$$/\text{t}/ \rightarrow [\text{dx}] \;/\; \acute{\text{V}} \underline{\hspace{2em}} \text{V}$$

While the form of these rules seems context-sensitive, Ch. 7 showed that phonological rule systems that do not have recursion are actually equivalent in power to the regular grammars.

Another way of conceptualizing a rule in a context-sensitive grammar is as rewriting a string of symbols δ as another string of symbols ϕ in a "non-decreasing" way; such that ϕ has at least as many symbols as δ.

CONTEXT-FREE

We studied **context-free** grammars in Ch. 12. Context-free rules allow any single non-terminal to be rewritten as any string of terminals and non-terminals. A non-terminal may also be rewritten as ϵ, although we didn't make use of this option in

Ch. 12.

RIGHT-LINEAR
LEFT-LINEAR

Regular grammars are equivalent to regular expressions. That is, a given regular language can be characterized either by a regular expression of the type we discussed in Chapter 2, or by a regular grammar. Regular grammars can either be **right-linear** or **left-linear**. A rule in a right-linear grammar has a single non-terminal on the left, and at most one non-terminal on the right-hand side. If there is a non-terminal on the right-hand side, it must be the last symbol in the string. The right-hand-side of left-linear grammars is reversed (the right-hand-side must start with (at most) a single non-terminal). All regular languages have both a left-linear and a right-linear grammar. For the rest of our discussion, we will consider only the right-linear grammars.

For example, consider the following regular (right-linear) grammar:

$$\begin{aligned} S &\rightarrow aA \\ S &\rightarrow bB \\ A &\rightarrow aS \\ B &\rightarrow bbS \\ S &\rightarrow \epsilon \end{aligned}$$

It is regular, since the left-hand-side of each rule is a single non-terminal and each right-hand side has at most one (rightmost) non-terminal. Here is a sample derivation in the language:

$$\begin{aligned} S \Rightarrow aA \Rightarrow aaS \Rightarrow aabB \Rightarrow aabbbS \Rightarrow aabbbaA \\ \Rightarrow aabbbaaS \Rightarrow aabbbaa \end{aligned}$$

We can see that each time S expands, it produces either aaS or $bbbS$; thus the reader should convince themself that this language corresponds to the regular expression $(aa \cup bbb)*$.

We will not present the proof that a language is regular if and only if it is generated by a regular grammar; it was first proved by Chomsky and Miller (1958) and can be found in textbooks like Hopcroft and Ullman (1979) and Lewis and Papadimitriou (1988). The intuition is that since the non-terminals are always at the right or left edge of a rule, they can be processed iteratively rather than recursively.

MILDLY
CONTEXT-SENSITIVE

The fifth class of languages and grammars that is useful to consider is the **mildly context-sensitive grammars** and the **mildly context-sensitive languages**. Mildly context-sensitive languages are a proper subset of the context-sensitive languages, and a proper superset of the context-free languages. The rules for mildly context-sensitive languages can be described in a number of ways; indeed it turns out that various grammar formalisms, including Tree-Adjoining Grammars (Joshi, 1985), Head Grammars Pollard (1984), Combinatory Categorial Grammars (CCG), (Steedman, 1996, 2000) and also a specific version of Minimalist Grammars (Stabler, 1997), are all weakly equivalent (Joshi et al., 1991).

15.2 How to Tell if a Language Isn't Regular

How do we know which type of rules to use for a given problem? Could we use regular expressions to write a grammar for English? Or do we need to use context-free rules or even context-sensitive rules? It turns out that for formal languages there are methods for deciding this. That is, we can say for a given formal language whether it is representable by a regular expression, or whether it instead requires a context-free grammar, and so on.

So if we want to know if some part of natural language (the phonology of English, let's say, or perhaps the morphology of Turkish) is representable by a certain class of grammars, we need to find a formal language that models the relevant phenomena and figure out which class of grammars is appropriate for this formal language.

Why should we care whether (say) the syntax of English is representable by a regular language? One main reason is that we'd like to know which type of rule to use in writing computational grammars for English. If English is regular, we would write regular expressions, and use efficient automata to process the rules. If English is context-free, we would write context-free rules and use the CKY algorithm to parse sentences, and so on.

Another reason to care is that it tells us something about the formal properties of different aspects of natural language; it would be nice to know where a language "keeps" its complexity; whether the phonological system of a language is simpler than the syntactic system, or whether a certain kind of morphological system is inherently simpler than another kind. It would be a strong and exciting claim, for example, if we could show that the phonology of English was capturable by a finite-state machine rather than the context-sensitive rules that are traditionally used; it would mean that English phonology has quite simple formal properties. Indeed, this fact was shown by Johnson (1972), and helped lead to the modern work in finite-state methods shown in Chapters 3 and 4.

15.2.1 The Pumping Lemma

The most common way to prove that a language is regular is to actually build a regular expression for the language. In doing this we can rely on the fact that the regular languages are closed under union, concatenation, Kleene star, complementation, and intersection. We saw examples of union, concatenation, and Kleene star in Ch. 2. So if we can independently build a regular expression for two distinct parts of a language, we can use the union operator to build a regular expression for the whole language, proving that the language is regular.

Sometimes we want to prove that a given language is *not* regular. An extremely useful tool for doing this is the **Pumping Lemma**. There are two intuitions behind this lemma. (Our description of the pumping lemma draws from Lewis and Papadimitriou (1988) and Hopcroft and Ullman (1979).) First, if a language can be modeled by a finite automaton with a finite number of states, we must be able to decide with a bounded amount of memory whether any string was in the language or not. This amount of memory can be different for different automata, but for a given automaton it can't

PUMPING LEMMA

grow larger for different strings (since a given automaton has a fixed number of states). Thus the memory needs must not be proportional to the length of the input. This means for example that languages like $a^n b^n$ are not likely to be regular, since we would need some way to remember what n was in order to make sure that there were an equal number of a's and b's. The second intuition relies on the fact that if a regular language has any long strings (longer than the number of states in the automaton), there must be some sort of loop in the automaton for the language. We can use this fact by showing that if a language *doesn't* have such a loop, then it can't be regular.

Let's consider a language L and the corresponding deterministic FSA M, which has N states. Consider an input string also of length N. The machine starts out in state q_0; after seeing 1 symbol it will be in state q_1; after N symbols it will be in state q_n. In other words, a string of length N will go through $N+1$ states (from q_0 to q_N). But there are only N states in the machine. This means that at least two of the states along the accepting path (call them q_i and q_j) must be the same. In other words, somewhere on an accepting path from the initial to final state, there must be a loop. Fig. 15.3 shows an illustration of this point. Let x be the string of symbols that the machine reads on going from the initial state q_0 to the beginning of the loop q_i. y is the string of symbols that the machine reads in going through the loop. z is the string of symbols from the end of the loop (q_j) to the final accepting state (q_N).

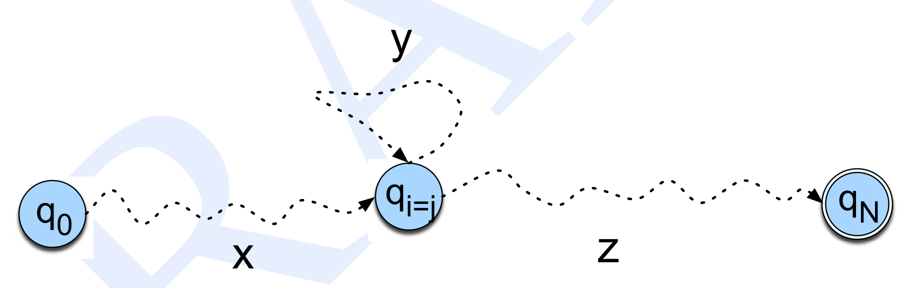


Figure 15.3 A machine with N states accepting a string xyz of N symbols

The machine accepts the concatenation of these three strings of symbols, that is, xyz. But if the machine accepts xyz it must accept xz! This is because the machine could just skip the loop in processing xz. Furthermore, the machine could also go around the loop any number of times; thus it must also accept $xyyz$, $xyyyz$, $xyyyyz$, and so on. In fact, it must accept any string of the form $xy^n z$ for $n \geq 0$.

The version of the pumping lemma we give is a simplified one for infinite regular languages; stronger versions can be stated that also apply to finite languages, but this one gives the flavor of this class of lemmas:

> **Pumping Lemma**. Let L be an infinite regular language. Then there are strings x, y, and z, such that $y \neq \epsilon$ and $xy^n z \in L$ for $n \geq 0$.

The pumping lemma states that if a language is regular, then there is some string y that can be "pumped" appropriately. But this doesn't mean that if we can pump some string y, the language must be regular. Non-regular languages may also have strings that can be pumped. Thus the lemma is not used for showing that a language *is* regular. Rather it is used for showing that a language *isn't* regular, by showing that in some language there is no possible string that can be pumped in the appropriate way.

Let's use the pumping lemma to show that the language $a^n b^n$ (i.e., the language consisting of strings of *a*s followed by an equal number of *b*s) is not regular. We must show that any possible string s that we pick cannot be divided up into three parts x, y, and z such that y can be pumped. Given a random string s from $a^n b^n$, we can distinguish three ways of breaking s up, and show that no matter which way we pick, we cannot find some y that can be pumped:

1. y is composed only of *a*s. (This implies that x is all *a*s too, and z contains all the *b*s, perhaps preceded by some *a*s.) But if y is all *a*s, that means $xy^n z$ has more *a*s than xyz. But this means it has more *a*s than *b*s, and so cannot be a member of the language $a^n b^n$!
2. y is composed only of *b*s. The problem here is similar to case 1; If y is all *b*s, that means $xy^n z$ has more *b*s than xyz, and hence has more *b*s than *a*s.
3. y is composed of both *a*s and *b*s (this implies that x is only *a*s, while z is only *b*s). This means that $xy^n z$ must have some *b*s before *a*s, and again cannot be a member of the language $a^n b^n$!

Thus there is no string in $a^n b^n$ that can be divided into x, y, z in such a way that y can be pumped, and hence $a^n b^n$ is not a regular language.

But while $a^n b^n$ is not a regular language, it is a context-free language. In fact, the context-free grammar that models $a^n b^n$ only takes two rules! Here they are:

$$
\begin{aligned}
S &\rightarrow a\ S\ b \\
S &\rightarrow \epsilon
\end{aligned}
$$

Here's a sample parse tree using this grammar to derive the sentence *aabb*:

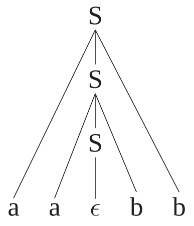


Figure 15.4 Context-free parse tree for *aabb*.

There is also a pumping lemma for context-free languages, that can be used whether or not a language is context-free; complete discussions can be found in Hopcroft and Ullman (1979) and Partee et al. (1990).

15.2.2 Are English and Other Natural Languages Regular Languages?

> *"How's business?" I asked.*
> *"Lousy and terrible." Fritz grinned richly. "Or I pull off a new deal in the next month or I go as a gigolo,"*
> "Either . . . or . . . ," *I corrected, from force of professional habit.*
> *"I'm speaking a lousy English just now," drawled Fritz, with great self-satisfaction. "Sally says maybe she'll give me a few lessons."*
>
> Christopher Isherwood, "Sally Bowles", from *Goodbye to Berlin*. 1935

Consider a formal version of the English language modeled as a set of strings of words. Is this language a regular language? It is generally agreed that natural languages like English, viewed in this way, are not regular, although most attempted proofs of this are well-known to be incorrect.

One kind of argument that is often made informally is that English number agreement cannot be captured by a regular grammar, because of the potentially unbounded distance between the subject and the verb in sentences like these:

(15.1) Which *problem* did your professor say she thought *was* unsolvable?

(15.2) Which *problems* did your professor say she thought *were* unsolvable?

In fact, a simple regular grammar *can* model number agreement, as Pullum and Gazdar (1982) show. Here's their regular (right-linear) grammar that models these sentences:

$$\begin{array}{rcl} S & \rightarrow & \text{Which problem did your professor say T} \\ S & \rightarrow & \text{Which problems did your professor say U} \\ T & \rightarrow & \text{she thought T} \mid \text{you thought T} \mid \text{was unsolvable} \\ U & \rightarrow & \text{she thought U} \mid \text{you thought U} \mid \text{were unsolvable} \end{array}$$

So a regular grammar could model English agreement. This grammar isn't elegant, and would have a huge explosion in the number of grammar rules, but that's not relevant to the question of the regularity or non-regularity of English.

Another common flaw with previously attempted proofs, pointed out by Mohri and Sproat (1998), is that the fact that a language L contains a subset L' at position P' in the Chomsky hierarchy does not imply that the language L is also at position P'. For example, a regular language can contain as a proper subset a context-free language. Thus the following two languages are context-free

$$L_1 = \{a^n b^n \ : n \in N\} \tag{15.3}$$

$$L_2 = \{ww^R \ : w \in \Sigma^*\} \tag{15.4}$$

and yet both L_1 and L_2 are contained in the regular language L:

$$L = \{a^p b^q \ : p, q \in N\} \tag{15.5}$$

Thus, the fact that a language L contains a sublanguage that is very complex says nothing about the overall complexity of language L.

CENTER-EMBEDDED

There are correct proofs that English (or rather "the set of strings of English words considered as a formal language") is not a regular language, based on the pumping lemma. A proof by Partee et al. (1990), for example, is based on a famous class of sentences with **center-embedded** structures (Yngve, 1960); here is a variant of these sentences:

The cat likes tuna fish.
The cat the dog chased likes tuna fish.
The cat the dog the rat bit chased likes tuna fish.
The cat the dog the rat the elephant admired bit chased likes tuna fish.

These sentences get harder to understand as they get more complex. For now, let's assume that the grammar of English allows an indefinite number of embeddings. Then in order to show that English is not regular, we need to show that languages with sentences like these are isomorphic to some non-regular language. Since every fronted *NP* must have its associated verb, these sentences are of the form:

(the + noun)^n (transitive verb)$^{n-1}$ likes tuna fish.

The idea of the proof will be to show that sentences of these structures can be produced by intersecting English with a regular expression. We will then use the pumping lemma to prove that the resulting language isn't regular.

In order to build a simple regular expression that we can intersect with English to produce these sentences, we define regular expressions for the noun groups (A) and the verbs (B):

A = { the cat, the dog, the rat, the elephant, the kangaroo,... }
B = { chased, bit, admired, ate, befriended, ... }

Now if we take the regular expression `/A* B* likes tuna fish/` and intersect it with English (considered as a set of strings), the resulting language is:

$$L = x^n y^{n-1} \textit{ likes tuna fish}, \;\; x \in A, y \in B$$

This language L can be shown to be non-regular via the pumping lemma (see Exercise 15.2). Since the intersection of English with a regular language is not a regular language, English cannot be a regular language either (since the regular languages are closed under intersection).

There is a well-known flaw, or at least an overly strong assumption with this proof, which is the assumption that these structures can be nested indefinitely. Sentences of English are clearly bounded by some finite length; perhaps we can safely say that all sentences of English are less than a billion words long. If the set of sentences is finite, then all natural languages are clearly finite-state. This is a flaw with all such proofs about the formal complexity of natural language. We will ignore this objection for now, since conveniently imagining that English has an infinite number of sentences can prove enlightening in understanding the properties of finite English.

A more worrisome potential flaw with this proof is that it depends on the assumption that these double relativizations of objects are strictly grammatical (even if hard to process). The research of Karlsson (2007) suggests that, while some kinds of center-embeddings are grammatical, these double relativizations of objects are in fact ungrammatical. In any case, sentences like this get hard much faster than a billion words, and are difficult to understand after a couple nestings. We will return to this issue in Sec. 15.4.

15.3 Is Natural Language Context-Free?

The previous section argued that English (considered as a set of strings) doesn't seem like a regular language. The natural next question to ask is whether English is a context-free language. This question was first asked by Chomsky (1956), and has an interesting history; a number of well-known attempts to prove English and other languages non-context-free have been published, and all except two have been disproved after publication. One of these two correct (or at least not-yet disproved) arguments derives from the syntax of a dialect of Swiss German; the other from the morphology of Bambara, a Northwestern Mande language spoken in Mali and neighboring countries (Culy, 1985). The interested reader should see Pullum (1991, pp. 131–146) for an extremely witty history of both the incorrect and correct proofs; this section will merely summarize one of the correct proofs, the one based on Swiss German.

Both of the correct arguments, and most of the incorrect ones, make use of the fact that the following languages, and ones that have similar properties, are not context-free:

$$\{xx \mid x \in \{a,b\}^*\} \tag{15.6}$$

This language consists of sentences containing two identical strings concatenated. The following related language is also not context-free:

$$a^n b^m c^n d^m \tag{15.7}$$

The non-context-free nature of such languages can be shown using the pumping lemma for context-free languages.

The attempts to prove that the natural languages are not a subset of the context-free languages do this by showing that natural languages have a property of these *xx* languages called **cross-serial dependencies**. In a cross-serial dependency, words or larger structures are related in left-to-right order as shown in Fig. 15.5. A language that has arbitrarily long cross-serial dependencies can be mapped to the *xx* languages.

CROSS-SERIAL DEPENDENCIES

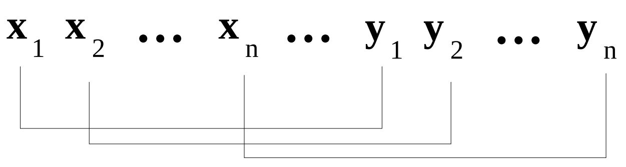


Figure 15.5 A schematic of a cross-serial dependency.

The successful proof, independently proposed by Huybregts (1984) and Shieber (1985) (as we might expect from the prevalence of multiple discovery in science; see page **??**) shows that a dialect of Swiss German spoken in Zürich has cross-serial constraints which make certain parts of that language equivalent to the non-context-free language $a^nb^mc^nd^m$. The intuition is that Swiss German allows a sentence to have a string of dative nouns followed by a string of accusative nouns, followed by a string of dative-taking verbs, followed by a string of accusative-taking verbs.

We will follow the version of the proof presented in Shieber (1985). First, he notes that Swiss German allows verbs and their arguments to be ordered cross-serially. Assume that all the example clauses we present below are preceded by the string *"Jan säit das"* ("Jan says that"):

(15.8)

. . . mer	*em Hans*	*es*	*huus*	*hälfed*	*aastriiche.*
. . . we	Hans/DAT	the	house/ACC	helped	paint.

". . . we helped Hans paint the house."

Notice the cross-serial nature of the semantic dependency: both nouns precede both verbs, and *em Hans* (Hans) is the argument of *hälfed* (helped) while *es huus* (the house) is the argument of *aastriiche* (paint). Furthermore, there is a cross-serial case dependency between the nouns and verbs; *hälfed* (helped) requires the dative, and *em Hans* is dative, while *aastriiche* (paint) takes the accusative, and *es huus* (the house) is accusative.

Shieber points out that this case marking can occur even across triply embedded cross-serial clauses like the following:

(15.9)

. . . mer	*d'chind*	*em Hans*	*es*	*huus*	*haend*	*wele*	*laa*	*hälfe*	*aastriiche.*
. . . we	the children/ACC	Hans/DAT	the	house/ACC	have	wanted to	let	help	paint.

". . . we have wanted to let the children help Hans paint the house."

Shieber notes that among such sentences, those with all dative NPs preceding all accusative NPs, and all dative-subcategorizing V's preceding all accusative-subcategorizing V's are acceptable.

(15.10) Jan säit das mer (d'chind)* (em Hans)* es huus haend wele laa* hälfe* aastriche.

Let's call the regular expression above R. Since it's a regular expression (you see it only has concatenation and Kleene stars) it must define a regular language, and so we can intersect R with Swiss German, and if the result is context free, so is Swiss German.

But it turns out that Swiss German requires that the number of verbs requiring dative objects (*hälfe*) must equal the number of dative NPs (*em Hans*) and similarly for accusatives. Furthermore, an arbitrary number of verbs can occur in a subordinate clause of this type (subject to performance constraints). This means that the result of intersecting this regular language with Swiss German is the following language:

(15.11) L = Jan säit das mer (d'chind)^n(em Hans)^m es huus haend wele (laa)^n (hälfe)^m aastriiche.

But this language is of the form $wa^nb^mxc^nd^my$, which is not context-free!

So we can conclude that Swiss German is not context free.

15.4 Complexity and Human Processing

We noted in passing earlier that many of the sentences that were used to argue for the non-finite state nature of English (like the "center-embedded" sentences) are quite difficult to understand. If you are a speaker of Swiss German (or if you have a friend who is), you will notice that the long cross-serial sentences in Swiss German are also rather difficult to follow. Indeed, as Pullum and Gazdar (1982) point out,

> precisely those construction-types that figure in the various proofs that English is not context-free appear to cause massive difficulty in the human processing system. . .

This brings us to a second use of the term **complexity**. In the previous section we talked about the complexity of a language. Here we turn to a question that is as much psychological as computational: the complexity of an individual sentence. Why are certain sentences hard to comprehend? Can this tell us anything about computational processes?

Many things can make a sentence hard to understand. For example we saw in Ch. 14 that a word is read more slowly if it is unpredictable; i.e., has a low *N*-gram probability or a low parse probability. We also saw in Ch. 14 **garden-path sentences** where ambiguity can cause difficulty; if there are multiple possible parses, a human reader (or listener) sometimes chooses the incorrect parse, leading to a double-take when switching back to the other parse. Other factors that affect sentence difficulty include implausible meanings and bad handwriting.

Another kind of difficulty seems to be related to human memory limitations, and it is this particular kind of complexity (often called "linguistic complexity" or "syntactic complexity") that bears an interesting relation to the formal-language complexity from the previous section.

Consider these sentences from Gibson (1998) that cause difficulties when people try to read them (we will use the # to mean that a sentence causes extreme processing difficulty). In each case the (ii) example is significantly more complex than the (i) example:

(15.12) (i) The cat likes tuna fish.
(ii) #The cat the dog the rat the goat licked bit chased likes tuna fish.

(15.13) (i) The child damaged the pictures which were taken by the photographer who the professor met at the party.
(ii) #The pictures which the photographer who the professor met at the party took were damaged by the child.

(15.14) (i) The fact that the employee who the manager hired stole office supplies worried the executive.
(ii) #The executive who the fact that the employee stole office supplies worried hired the manager.

The earliest work on sentences of this type noticed that they all exhibit *nesting* or *center-embedding* (Chomsky, 1957; Yngve, 1960; Chomsky and Miller, 1963; Miller and Chomsky, 1963). That is, they all contain examples where a syntactic category A is nested within another category B, and surrounded by other words (X and Y):

$[_B$ X $[_A]$ Y$]$

In each of the examples above, part (i) has zero or one embedding, while part (ii) has two or more embeddings. For example in (15.12ii) above, there are three reduced relative clauses embedded inside each other:

(15.15) # $[_S$ The cat $[_{S'}$ the dog $[_{S'}$ the rat $[_{S'}$ the elephant admired$]$ bit$]$ chased$]$ likes tuna fish$]$.

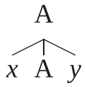
In (15.13ii), the relative clause *who the professor met at the party* is nested in between *the photographer* and *took*. The relative clause *which the photographer . . . took* is then nested between *The pictures* and *were damaged by the child*.

(15.16) #The pictures [which the photographer [who the professor met at the party] took] were damaged by the child.

The difficulty with these nested structures is not caused by ungrammaticality, since the structures that are used in the complex sentences in (15.12ii)–(15.14ii) are the same ones used in the easier sentences (15.12i)–(15.14i). The difference between the easy and complex sentences seems to relate to the number of embeddings. But there is no natural way to write a grammar that allows N embeddings but not $N+1$ embeddings. Rather, the complexity of these sentences seems to be a processing phenomenon; some fact about the human parsing mechanism is unable to deal with these kinds of multiple nestings, in English and in other languages (Cowper, 1976; Babyonyshev and Gibson, 1999).

SELF-EMBEDDED

The difficulty of these sentences seems to have something to do with *memory limitations*. Early formal grammarians suggested that this might have something to do with how the parser processed embeddings. For example Yngve (1960) suggested that the human parser is based on a limited-size stack, and that the more incomplete phrase-structure rules the parser needs to store on the stack, the more complex the sentence. Miller and Chomsky (1963) hypothesized that **self-embedded** structures are particularly difficult. A self-embedded structure contains a syntactic category A nested within another example of A, and surrounded by other words (x and y below); such structures might be difficult because a stack-based parser might confuse two copies of the rule on the stack.



The intuitions of these early models are important, although we no longer believe that the complexity problems have to do with an actual stack. For example, we now know that there are complexity differences between sentences that have the same number of embeddings, such as the well-known difference between subject-extracted relative clauses (15.17ii) and object-extracted relative clauses (15.17i):

(15.17) (i) $[_S$ The reporter $[_{S'}$ who $[_S$ the senator attacked $]]$ admitted the error $]$.
(ii) $[_S$ The reporter $[_{S'}$ who $[_S$ attacked the senator $]]$ admitted the error $]$.

The object-extracted relative clauses are more difficult to process, as measured for example by the amount of time it takes to read them, and other factors (MacWhinney, 1977, 1982; MacWhinney and Csaba Pléh, 1988; Ford, 1983; Wanner and Maratsos, 1978; King and Just, 1991; Gibson, 1998). Indeed, Karlsson (2007) has shown in a study of seven languages that the grammaticality of center embeddings depends a lot on the particular syntactic structure (e.g., relative clauses versus double relativization of objects) being embedded. Another problem for the old-fashioned stack-based models is the fact that discourse factors can make some doubly nested relative clauses easier to process, such as the following double nested example:

(15.18) The pictures [that the photographer [who I met at the party] took] turned out very well.

What seems to make this structure less complex is that one of the embedded NPs is the word *I*; pronouns like *I* and *you* seem to be easier to process, perhaps because they do not introduce a new entity to the discourse.

One human parsing model that accounts for all of this data is the Dependency Locality Theory (Gibson, 1998, 2003). The intuition of the DLT is that object relatives are difficult because they have two nouns that appear before any verb. The reader must hold on to these two nouns without knowing how they will fit into the sentences.

More specifically, the DLT proposes that the processing cost of integrating a new word *w* is proportional to the distance between *w* and the syntactic item with which *w* is being integrated. Distance is measured not just in words, but in how many new phrases or discourse referents have to be held in memory at the same time. Thus the memory load for a word is higher if there have been many intervening *new discourse referents* since the word has been predicted. Thus the DLT predicts that a sequence of NPs can be made easier to process if one of them is a pronoun that is already active in the discourse, explaining (15.18).

In summary, the complexity of these ‘center-embedded’ and other examples does seem to be related to memory, although not in as direct a link to parsing stack size as was first thought 40 years ago. Understanding the relationship between these memory factors and the statistical parsing factors mentioned in Ch. 14 is an exciting research area that is just beginning to be investigated.

15.5 Summary

This chapter introduced two different ideas of **complexity**: the complexity of a formal language, and the complexity of a human sentence.

- Grammars can be characterized by their **generative power**. One grammar is of greater generative power or **complexity** than another if it can define a language that the other cannot define. The **Chomsky hierarchy** is a hierarchy of grammars based on their generative power. It includes **Turing equivalent**, **context-sensitive**, **context-free**, and **regular** grammars.
- The **pumping lemma** can be used to prove that a given language is **not regular**. English is not a regular language, although the kinds of sentences that make English non-regular are exactly those that are hard for people to parse. Despite many decades of attempts to prove the contrary, English does, however, seem to be a context-free language. The syntax of Swiss-German and the morphology of Bambara, by contrast, are not context-free and seem to require mildly context-sensitive grammars.

- Certain **center-embedded** sentences are hard for people to parse. Many theories agree that this difficulty is somehow caused by **memory limitations** of the human parser.

Bibliographical and Historical Notes

Chomsky (1956) first asked whether finite-state automata or context-free grammars were sufficient to capture the syntax of English. His suggestion in that paper that English syntax contained "examples that are not easily explained in terms of phrase structure" was a motivation for his development of syntactic transformations.

Choksky's proof was based on the language $\{xx^R : x \in \{a,b\}^*\}$. x^R means "the reverse of x", so each sentence of this language consists of a string of *a*s and *b*s followed by the reverse or "mirror image" of the string. This language is not regular; Partee et al. (1990) shows this by intersecting it with the regular language aa^*bbaa^*. The resulting language is $a^nb^2a^n$; it is left as an exercise for the reader (Exercise 15.3) to show that this is not regular by the pumping lemma.

Chomsky proof shows that English had mirror-like properties, relying on multiple embeddings of the following English syntactic structures, where $S_1, S_2, \ldots, S_n$ are declarative sentences in English,

- If S_1, then S_2
- Either S_3, or S_4
- The man who said S_5 is arriving today

See Chomsky (1956) for details.

Pullum (1991, pp. 131–146) is the definitive historical study of research on the non-context-free-ness of natural language. The early history of attempts to prove natural languages non-context-free is summarized in Pullum and Gazdar (1982). The pumping lemma was originally presented by Bar-Hillel et al. (1961), who also offer a number of important proofs about the closure and decidability properties of finite-state and context-free languages. Further details, including the pumping lemma for context-free languages (also due to Bar-Hillel et al. (1961)) can be found in a textbook in automata theory such as Hopcroft and Ullman (1979).

Yngve's idea that the difficulty of center-embedded sentences could be explained if the human parser was finite-state was taken up by Church (1980) in his master's thesis. He showed that a finite-state parser that implements this idea could also explain a number of other grammatical and psycholinguistic phenomena. While the cognitive modeling field has turned toward more sophisticated models of complexity, Church's work can be seen as the beginning of the return to finite-state models in speech and language processing that characterized the 1980s and 1990s.

There are a number of other ways of looking at complexity that we didn't have space to go into here. One is whether language processing is NP-complete. **NP-complete** is the name of a class of problems which are suspected to be particularly difficult to process. Barton et al. (1987) prove a number of complexity results about

NP-COMPLETE

the NP-completeness of natural language recognition and parsing. Among other things, they showed that

1. Maintaining lexical and agreement feature ambiguities over a potentially infinite-length sentence causes the problem of recognizing sentences in some unification-based formalisms like Lexical-Functional Grammar to be NP-complete.
2. Two-level morphological parsing (or even just mapping between lexical and surface form) is also NP-complete.

Recent work has also begun to link processing complexity with information-theoretic measures like Kolmogorov complexity (Juola, 1999).

Finally, recent work has looked at the expressive power of different kinds of probabilistic grammars, showing for example that weighted context-free grammars (in which each rule has a weight) and probabilistic context-free grammars (in which the weights of the rules for a non-terminal must sum to 1) are equally expressive (Smith and Johnson, 2007; Abney et al., 1999; Chi, 1999).

Exercises

15.1 Is the language $a^n b^2 a^n$ context-free?

15.2 Use the pumping lemma to show this language is not regular:

$$L = x^n y^{n-1} \mathit{likes\ tuna\ fish}, x \in A, y \in B$$

15.3 Partee et al. (1990) showed that the language $xx^R, x \in a, b*$ is not regular, by intersecting it with the regular language aa^*bbaa^*. The resulting language is $a^n b^2 a^n$. Use the pumping lemma to show that this language is not regular, completing the proof that $xx^R, x \in a, b*$ is not regular.

15.4 Build a context-free grammar for the language

$$L = \{xx^R | x \in a, b*\}$$

Abney, S. P., McAllester, D. A., and Pereira, F. C. N. (1999). Relating probabilistic grammars and automata. In *ACL-99*.

Babyonyshev, M. and Gibson, E. (1999). The complexity of nested structures in Japanese. *Language*, *75*(3), 423–450.

Bar-Hillel, Y., Perles, M., and Shamir, E. (1961). On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, *14*, 143–172. Reprinted in Y. Bar-Hillel. (1964). *Language and Information: Selected Essays on their Theory and Application*, Addison-Wesley 1964, 116–150.

Barton, Jr., G. E., Berwick, R. C., and Ristad, E. S. (1987). *Computational Complexity and Natural Language*. MIT Press.

Chi, Z. (1999). Statistical Properties of Probabilistic Context-Free Grammars. *Computational Linguistics*, *25*(1), 131–160.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, *2*(3), 113–124.

Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, *2*, 137–167.

Chomsky, N. and Halle, M. (1968). *The Sound Pattern of English*. Harper and Row.

Chomsky, N. and Miller, G. A. (1958). Finite-state languages. *Information and Control*, *1*, 91–112.

Chomsky, N. and Miller, G. A. (1963). Introduction to the formal analysis of natural languages. In Luce, R. D., Bush, R., and Galanter, E. (Eds.), *Handbook of Mathematical Psychology*, Vol. 2, pp. 269–322. Wiley.

Church, K. W. (1980). On memory limitations in natural language processing. Master's thesis, MIT. Distributed by the Indiana University Linguistics Club.

Cowper, E. A. (1976). *Constraints on Sentence Complexity: A Model for Syntactic Processing*. Ph.D. thesis, Brown University, Providence, RI†.

Culy, C. (1985). The complexity of the vocabulary of Bambara. *Linguistics and Philosophy*, *8*, 345–351.

Ford, M. (1983). A method for obtaining measures of local parsing complexity through sentences. *Journal of Verbal Learning and Verbal Behavior*, *22*, 203–218.

Gibson, E. (1998). Linguistic complexity: Locality of syntactic dependencies. *Cognition*, *68*, 1–76.

Gibson, E. (2003). Sentence comprehension, linguistic complexity in. In Nadel, L. (Ed.), *Encyclopedia of Cognitive Science*, pp. 1137–1141. Nature Publishing Group, New York, NY.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

Huybregts, R. (1984). The weak inadequacy of context-free phrase structure grammars. In de Haan, G., Trommele, M., and Zonneveld, W. (Eds.), *Van Periferie naar Kern*. Foris, Dordrecht†. Cited in Pullum (1991).

Johnson, C. D. (1972). *Formal Aspects of Phonological Description*. Mouton, The Hague. Monographs on Linguistic Analysis No. 3.

Joshi, A. K., Vijay-Shanker, K., and Weir, D. J. (1991). The convergence of mildly context-sensitive grammatical formalisms. In Sells, P., Shieber, S., and Wasow, T. (Eds.), *Foundational issues in natural language processing*, pp. 31–81. MIT Press.

Joshi, A. K. (1985). Tree adjoining grammars: how much context-sensitivity is required to provide reasonable structural descriptions?. In Dowty, D. R., Karttunen, L., and Zwicky, A. (Eds.), *Natural Language Parsing*, pp. 206–250. Cambridge University Press.

Juola, P. (1999). Measuring linguistic complexity. Presented at the 4th Conference on Conceptual Structure, Discourse, and Language (CSDL-4), Georgia.

Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Linguistics*, *43*, 365–392.

King, J. and Just, M. A. (1991). Individual differences in syntactic processing: The role of working memory. *Journal of Memory and Language*, *30*, 580–602.

Lewis, H. and Papadimitriou, C. (1988). *Elements of the Theory of Computation*. Prentice-Hall. Second edition.

MacWhinney, B. (1977). Starting points. *Language*, *53*, 152–168.

MacWhinney, B. (1982). Basic syntactic processes. In Kuczaj, S. (Ed.), *Language Acquisition: Volume 1, Syntax and Semantics*, pp. 73–136. Lawrence Erlbaum.

MacWhinney, B. and Csaba Pléh (1988). The processing of restrictive relative clauses in Hungarian. *Cognition*, *29*, 95–141.

Miller, G. A. and Chomsky, N. (1963). Finitary models of language users. In Luce, R. D., Bush, R. R., and Galanter, E. (Eds.), *Handbook of Mathematical Psychology*, Vol. II, pp. 419–491. John Wiley.

Mohri, M. and Sproat, R. (1998). On a common fallacy in computational linguistics. In Suominen, M., Arppe, A., Airola, A., Heinämäki, O., Miestamo, M., Määttä, U., Niemi, J., Pitkänen, K. K., and Sinnemäki, K. (Eds.), *A Man of Measure: Festschrift in Honour of Fred Karlsson on this 60th Birthday*, pp. 432–439. SKY Journal of Linguistics, Volume 19, 2006.

Partee, B. H., ter Meulen, A., and Wall, R. E. (1990). *Mathematical Methods in Linguistics*. Kluwer, Dordrecht.

Pollard, C. (1984). *Generalized phrase structure grammars, head grammars, and natural language*. Ph.D. thesis, Stanford University.

Pullum, G. K. and Gazdar, G. (1982). Natural languages and context-free languages. *Linguistics and Philosophy*, *4*, 471–504.

Pullum, G. K. (1991). *The Great Eskimo Vocabulary Hoax*. University of Chicago, Chicago, IL.

Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, *8*, 333–343.

Smith, N. A. and Johnson, M. (2007). Weighted and probabilistic context-free grammars are equally expressive. *Computational Linguistics*. To appear.

Stabler, E. (1997). Derivational minimalism. In Retoré, C. (Ed.), *Logical Aspects of Computational Linguistics*, pp. 68–95. Springer.

Steedman, M. (1996). *Surface Structure and Interpretation*. MIT Press. Linguistic Inquiry Monograph, 30.

Steedman, M. (2000). *The Syntactic Process*. The MIT Press.

Wanner, E. and Maratsos, M. (1978). An ATN approach to comprehension. In Halle, M., Bresnan, J., and Miller, G. A. (Eds.), *Linguistic Theory and Psychological Reality*, pp. 119–161. MIT Press.

Yngve, V. H. (1960). A model and an hypothesis for language structure. *Proceedings of the American Philosophical Society*, *104*, 444–466.

16 FEATURES AND UNIFICATION

FRIAR FRANCIS: *If either of you know any inward impediment why you should not be conjoined, charge you, on your souls, to utter it.*

William Shakespeare, *Much Ado About Nothing*

From a reductionist perspective, the history of the natural sciences over the last few hundred years can be seen as an attempt to explain the behavior of larger structures by the combined action of smaller primitives. In biology, the properties of inheritance have been explained by the action of genes, and then again the properties of genes have been explained by the action of DNA. In physics, matter was reduced to atoms and then again to subatomic particles. The appeal of reductionism has not escaped computational linguistics. In this chapter we introduce the idea that grammatical categories like *VPto*, *Sthat*, *Non3sgAux*, or *3sgNP*, as well as the grammatical rules like *S* → *NP VP* that make use of them, should be thought of as *objects* that can have complex sets of *properties* associated with them. The information in these properties is represented by **constraints**, and so these kinds of models are often called **constraint-based formalisms**.

CONSTRAINT-BASED FORMALISMS

Why do we need a more fine-grained way of representing and placing constraints on grammatical categories? One problem arose in Ch. 12, where we saw that naive models of grammatical phenomena such as agreement and subcategorization can lead to overgeneration problems. For example, in order to avoid ungrammatical noun phrases such as *this flights* and verb phrases like *disappeared a flight*, we were forced to create a huge proliferation of primitive grammatical categories such as *Non3sgVPto*, *NPmass*, *3sgNP* and *Non3sgAux*. These new categories led, in turn, to an explosion in the number of grammar rules and a corresponding loss of generality in the grammar. A constraint-based representation scheme will allow us to represent fine-grained information about number and person, agreement, subcategorization, as well as semantic categories like mass/count.

Constraint-based formalisms have other advantages that we will not cover in this chapter, such as the ability to model more complex phenomena than context-free grammars, and the ability to efficiently and conveniently compute semantics for syntactic representations.

Consider briefly how this approach might work in the case of grammatical number. As we saw in Ch. 12, noun phrases like *this flight* and *those flights* can be distinguished based on whether they are singular or plural. This distinction can be captured if we associate a property called NUMBER that can have the value singular or plural, with appropriate members of the *NP* category. Given this ability, we can say that *this flight* is a member of the *NP* category and, in addition, has the value singular for its NUMBER property. This same property can be used in the same way to distinguish singular and plural members of the *VP* category such as *serves lunch* and *serve lunch*.

Of course, simply associating these properties with various words and phrases does not solve any of our overgeneration problems. To make these properties useful, we need the ability to perform simple operations, such as equality tests, on them. By pairing such tests with our core grammar rules, we can add various constraints to help ensure that only grammatical strings are generated by the grammar. For example, we might want to ask whether or not a given noun phrase and verb phrase have the same values for their respective number properties. Such a test is illustrated by the following kind of rule.

$S \rightarrow NP\ VP$

Only if the number of the NP is equal to the number of the VP.

The remainder of this chapter provides the details of one computational implementation of a constraint-based formalism, based on feature structures and unification. The next section describes feature structures, the representation used to capture the kind of grammatical properties we have in mind. Section 16.2 then introduces the unification operator that is used to implement basic operations over feature structures. Section 16.3 then covers the integration of these structures into a grammatical formalism. Section 16.4 then introduces the unification algorithm and its required data structures. Next, Section 16.5 describes how feature structures and the unification operator can be integrated into a parser. Finally, Section 16.6 discusses the most significant extension to this constraint-based formalism, the use of types and inheritance, as well as other extensions.

16.1 Feature Structures

FEATURE STRUCTURES

One of the simplest ways to encode the kind of properties that we have in mind is through the use of **feature structures**. These are simply sets of feature-value pairs, where features are unanalyzable atomic symbols drawn from some finite set, and values are either atomic symbols or feature structures themselves. Such feature structures are traditionally illustrated with the following kind of matrix-like diagram, called an **attribute-value matrix** or **AVM**:

ATTRIBUTE-VALUE MATRIX

AVM

$$\begin{bmatrix} \text{FEATURE}_1 & \textit{value}_1 \\ \text{FEATURE}_2 & \textit{value}_2 \\ \vdots & \\ \text{FEATURE}_n & \textit{value}_n \end{bmatrix}$$

To make this concrete, consider the number property discussed above. To capture this property, we will use the symbol NUMBER to designate this grammatical attribute, and the symbols *sg* and *pl* (introduced in Ch. 3) to designate the possible values it can take on in English. A simple feature structure consisting of this single feature would then be illustrated as follows:

$$\begin{bmatrix} \text{NUMBER} & \textit{sg} \end{bmatrix}$$

Adding an additional feature-value pair to capture the grammatical notion of person leads to the following feature structure:

$$\begin{bmatrix} \text{NUMBER} & \textit{sg} \\ \text{PERSON} & \textit{3rd} \end{bmatrix}$$

Next we can encode the grammatical category of the constituent that this structure corresponds to through the use of the CAT feature. For example, we can indicate that these features are associated with a noun phrase by using the following structure:

$$\begin{bmatrix} \text{CAT} & \textit{NP} \\ \text{NUMBER} & \textit{sg} \\ \text{PERSON} & \textit{3rd} \end{bmatrix}$$

This structure can be used to represent the *3sgNP* category introduced in Ch. 12 to capture a restricted subcategory of noun phrases. The corresponding plural version of this structure would be captured as follows:

$$\begin{bmatrix} \text{CAT} & \textit{NP} \\ \text{NUMBER} & \textit{pl} \\ \text{PERSON} & \textit{3rd} \end{bmatrix}$$

Note that the values of the CAT and PERSON features remain the same for these last two structures. This illustrates how the use of feature structures allows us to both preserve the core set of grammatical categories and draw distinctions among members of a single category.

As mentioned earlier, features are not limited to atomic symbols as their values; they can also have other feature structures as their values. This is particularly useful when we wish to bundle a set of feature-value pairs together for similar treatment. As an example of this, consider that the NUMBER and PERSON features are often lumped together since grammatical subjects must agree with their predicates in both their number and person properties. This lumping together can be captured by introducing an AGREEMENT feature that takes a feature structure consisting of the NUMBER and PERSON feature-value pairs as its value. Introducing this feature into our third person singular noun phrase yields the following kind of structure.

$$\begin{bmatrix} \text{CAT} & \textit{NP} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \textit{sg} \\ \text{PERSON} & \textit{3rd} \end{bmatrix} \end{bmatrix}$$

Given this kind of arrangement, we can test for the equality of the values for both the NUMBER and PERSON features of two constituents by testing for the equality of their AGREEMENT features.

FEATURE PATH

This ability to use feature structures as values leads fairly directly to the notion of a **feature path**. A feature path is nothing more than a sequence of features through a feature structure leading to a particular value. For example, in the last feature structure, we can say that the ⟨AGREEMENT NUMBER⟩ path leads to the value *sg*, while the ⟨AGREEMENT PERSON⟩ path leads to the value *3rd*. This notion of a path leads naturally to an alternative graphical way of illustrating feature structures, shown in Figure 16.1, which as we will see in Section 16.4 is suggestive of how they will be implemented. In these diagrams, feature structures are depicted as directed graphs where features appear as labeled edges and values as nodes.

REENTRANT

Although this notion of paths will prove useful in a number of settings, we introduce it here to help explain an additional important kind of feature structure: those that contain features that actually share some feature structure as a value. Such feature structures will be referred to as **reentrant** structures. What we have in mind here is not the simple idea that two features might have equal values, but rather that they share precisely the same feature structure (or node in the graph). These two cases can be distinguished clearly if we think in terms of paths through a graph. In the case of simple equality, two paths lead to distinct nodes in the graph that anchor identical, but distinct structures. In the case of a reentrant structure, two feature paths actually lead to the same node in the structure.

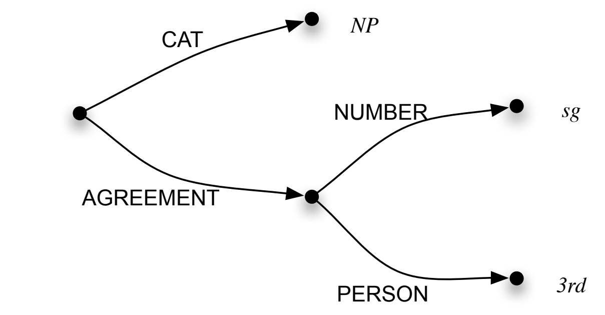


Figure 16.1 A feature structure with shared values. The location (value) found by following the ⟨HEAD SUBJECT AGREEMENT⟩ path is the same as that found via the ⟨HEAD AGREEMENT⟩ path.

Figure 16.2 illustrates a simple example of reentrancy. In this structure, the ⟨HEAD SUBJECT AGREEMENT⟩ path and the ⟨HEAD AGREEMENT⟩ path lead to the same location. Shared structures like this will be denoted in our AVM diagrams by adding numerical indexes that signal the values to be shared. The AVM version of the feature structure from Figure 16.2 would be denoted as follows, using the notation of the PATR-II system (Shieber, 1986), based on Kay (1979):

$$\begin{bmatrix} \text{CAT} & S \\ \text{HEAD} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \begin{bmatrix} \text{NUMBER} & sg \\ \text{PERSON} & 3rd \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

As we will see, these simple structures give us the ability to express linguistic generalizations in surprisingly compact and elegant ways.

16.2 Unification of Feature Structures

As noted earlier, feature structures would be of little use without our being able to perform reasonably efficient and powerful operations on them. As we will show, the two principal operations we need to perform are merging the information content of two structures and rejecting the merger of structures that are incompatible. Fortunately, a single computational technique, called **unification**, suffices for both of these purposes. The bulk of this section will illustrate through a series of examples how unification instantiates these notions of merger and compatibility.

UNIFICATION

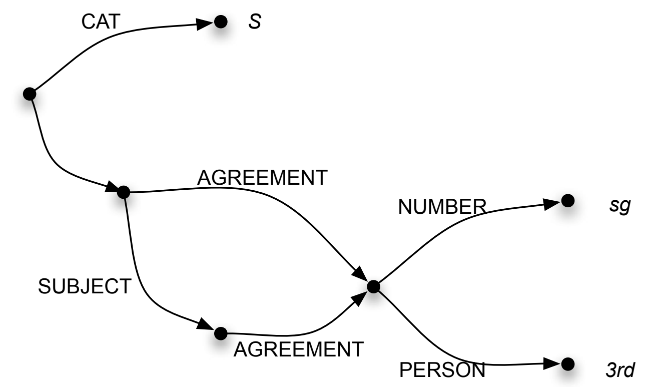


Figure 16.2 A feature structure with shared values. The location (value) found by following the ⟨HEAD SUBJECT AGREEMENT⟩ path is the same as that found via the ⟨HEAD AGREEMENT⟩ path.

Discussion of the unification algorithm and its implementation will be deferred to Section 16.4.

We begin with the following simple application of the unification operator.

$$\begin{bmatrix}\textsc{number} & \textit{sg}\end{bmatrix} \sqcup \begin{bmatrix}\textsc{number} & \textit{sg}\end{bmatrix} = \begin{bmatrix}\textsc{number} & \textit{sg}\end{bmatrix}$$

As this equation illustrates, unification is a binary operation (represented here as ⊔) that accepts two feature structures as arguments and returns a feature structure when it succeeds. In this example, unification is being used to perform a simple equality check. The unification succeeds because the corresponding NUMBER features in each structure agree as to their values. In this case, since the original structures are identical, the output is the same as the input. The following similar kind of check fails since the NUMBER features in the two structures have incompatible values.

$$\begin{bmatrix}\textsc{number} & \textit{sg}\end{bmatrix} \sqcup \begin{bmatrix}\textsc{number} & \textit{pl}\end{bmatrix} \textit{Fails!}$$

This next unification illustrates an important aspect of the notion of compatibility in unification.

$$\begin{bmatrix}\textsc{number} & \textit{sg}\end{bmatrix} \sqcup \begin{bmatrix}\textsc{number} & []\end{bmatrix} = \begin{bmatrix}\textsc{number} & \textit{sg}\end{bmatrix}$$

In this situation, these features structures are taken to be compatible, and are hence capable of being merged, despite the fact that the given values for the respective NUMBER features are different. The [] value in the second structure indicates that the value has been left unspecified. A feature with such a [] value can be successfully matched to any value in a corresponding feature in another structure.

Therefore, in this case, the value *sg* from the first structure can match the [] value from the second, and as is indicated by the output shown, the result of this type of unification is a structure with the value provided by the more specific, non-null, value.

The next example illustrates another of the merger aspects of unification.

$$\begin{bmatrix}\text{NUMBER} & \mathit{sg}\end{bmatrix} \sqcup \begin{bmatrix}\text{PERSON} & \mathit{3rd}\end{bmatrix} = \begin{bmatrix}\text{NUMBER} & \mathit{sg}\\ \text{PERSON} & \mathit{3rd}\end{bmatrix}$$

Here the result of the unification is a merger of the original two structures into one larger structure. This larger structure contains the union of all the information stored in each of the original structures. Although this is a simple example, it is important to understand why these structures are judged to be compatible: they are compatible because they contain no features that are explicitly incompatible. The fact that they each contain a feature-value pair that the other does not is not a reason for the unification to fail.

We will now consider a series of cases involving the unification of somewhat more complex reentrant structures. The following example illustrates an equality check complicated by the presence of a reentrant structure in the first argument.

$$\begin{bmatrix}\text{AGREEMENT} & \boxed{1}\begin{bmatrix}\text{NUMBER} & \mathit{sg}\\ \text{PERSON} & \mathit{3rd}\end{bmatrix}\\ \text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \boxed{1}\end{bmatrix}\end{bmatrix}$$

$$\sqcup \begin{bmatrix}\text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \begin{bmatrix}\text{PERSON} & \mathit{3rd}\\ \text{NUMBER} & \mathit{sg}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

$$= \begin{bmatrix}\text{AGREEMENT} & \boxed{1}\begin{bmatrix}\text{NUMBER} & \mathit{sg}\\ \text{PERSON} & \mathit{3rd}\end{bmatrix}\\ \text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \boxed{1}\end{bmatrix}\end{bmatrix}$$

The important elements in this example are the SUBJECT features in the two input structures. The unification of these features succeeds because the values found in the first argument by following the $\boxed{1}$ numerical index, match those that are directly present in the second argument. Note that, by itself, the value of the AGREEMENT feature in the first argument would have no bearing on the success of unification since the second argument lacks an AGREEMENT feature at the top level. It only becomes relevant because the value of the AGREEMENT feature is shared with the SUBJECT feature.

The following example illustrates the copying capabilities of unification.

(16.1)

$$\begin{aligned}
&\begin{bmatrix} \text{AGREEMENT} & \boxed{1} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix} \\
\sqcup\ &\begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & \textit{3rd} \\ \text{NUMBER} & \textit{sg} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\
=\ &\begin{bmatrix} \text{AGREEMENT} & \boxed{1} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \begin{bmatrix} \text{PERSON} & \textit{3rd} \\ \text{NUMBER} & \textit{sg} \end{bmatrix} \end{bmatrix} \end{bmatrix}
\end{aligned}$$

Here the value found via the second argument's ⟨SUBJECT AGREEMENT⟩ path is copied over to the corresponding place in the first argument. In addition, the AGREEMENT feature of the first argument receives a value as a side-effect of the index linking it to the value at the end of the ⟨SUBJECT AGREEMENT⟩ path.

The next example demonstrates the important difference between features that actually share values versus those that merely have identical looking values.

(16.2)

$$\begin{aligned}
&\begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \textit{sg} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \textit{sg} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\
\sqcup\ &\begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 3 \\ \text{NUMBER} & \textit{sg} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\
=\ &\begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \textit{sg} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \textit{sg} \\ \text{PERSON} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}
\end{aligned}$$

The values at the end of the ⟨SUBJECT AGREEMENT⟩ path and the ⟨AGREEMENT⟩ path are the same, but not shared, in the first argument. The unification of the SUBJECT features of the two arguments adds the PERSON information from the second argument to the result. However, since there is no index linking the AGREEMENT feature to the ⟨SUBJECT AGREEMENT⟩ path, this information is not added to the value of the AGREEMENT feature.

Finally, consider the following example of a failure to unify.

$$\begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix}\text{NUMBER} & sg\\ \text{PERSON} & 3\end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \boxed{1}\end{bmatrix}\end{bmatrix}$$

$$\sqcup \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix}\text{NUMBER} & sg\\ \text{PERSON} & 3\end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \begin{bmatrix}\text{NUMBER} & \text{PL}\\ \text{PERSON} & 3\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Fails!

Proceeding through the features in order, we first find that the AGREEMENT features in these examples successfully match. However, when we move on to the SUBJECT features, we find that the values found at the respective $\langle$ SUBJECT AGREEMENT NUMBER $\rangle$ paths differ, causing a unification failure.

Feature structures are a way of representing partial information about some linguistic object or placing informational constraints on what the object can be. Unification can be seen as a way of merging the information in each feature structure, or describing objects which satisfy both sets of constraints. Intuitively, unifying two feature structures produces a new feature structure which is more specific (has more information) than, or is identical to, either of the input feature structures.

SUBSUMES

We say that a less specific (more abstract) feature structure **subsumes** an equally or more specific one. Subsumption is represented by the operator $\sqsubseteq$. A feature structure F subsumes a feature structure G ($F \sqsubseteq G$) if and only if:

1. For every feature x in F, $F(x) \sqsubseteq G(x)$ (where $F(x)$ means "the value of the feature x of feature structure F").
2. For all paths p and q in F such that $F(p) = F(q)$, it is also the case that $G(p) = G(q)$.

For example, consider these feature structures:

$$\begin{bmatrix}\text{NUMBER} & sg\end{bmatrix} \tag{16.3}$$

$$\begin{bmatrix}\text{PERSON} & 3\end{bmatrix} \tag{16.4}$$

$$\begin{bmatrix}\text{NUMBER} & sg\\ \text{PERSON} & 3\end{bmatrix} \tag{16.5}$$

$$\begin{bmatrix}\text{CAT} & \text{VP}\\ \text{AGREEMENT} & \boxed{1}\\ \text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \boxed{1}\end{bmatrix}\end{bmatrix} \tag{16.6}$$

(16.7)
$$\begin{bmatrix} \text{CAT} & \text{VP} \\ \text{AGREEMENT} & \boxed{1} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \begin{bmatrix} \text{PERSON} & 3 \\ \text{NUMBER} & sg \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The following subsumption relations hold among them:

$16.3 \sqsubseteq 16.5$

$16.4 \sqsubseteq 16.5$

$16.6 \sqsubseteq 16.7$

Subsumption is a partial ordering; there are pairs of feature structures that neither subsume nor are subsumed by each other:

$16.3 \not\sqsubseteq 16.4$

$16.4 \not\sqsubseteq 16.3$

SEMILATTICE

MONOTONIC

Since every feature structure is subsumed by the empty structure [], the relation among feature structures can be defined as a **semilattice**. Unification can be defined in terms of the subsumption semilattice. Given two feature structures F and G, $F \sqcup G$ is defined as the most general feature structure H such that $F \sqsubseteq H$ and $G \sqsubseteq H$. Since the information ordering defined by subsumption is a semilattice, the unification operation is **monotonic** (Pereira and Shieber, 1984; Rounds and Kasper, 1986; Moshier, 1988). This means that if some description is true of a feature structure, unifying it with another feature structure results in a feature structure that still satisfies the original description. The unification operation is therefore associative; given a finite set of feature structures to unify, we can check them in any order and get the same result.

To summarize, unification is a way of implementing the integration of knowledge from different constraints. Given two compatible feature structures as input, it produces the most general feature structure which nonetheless contains all the information in the inputs. Given two incompatible feature structures, it fails.

16.3 Feature Structures in the Grammar

Our primary purpose in introducing feature structures and unification has been to provide a way to elegantly express syntactic constraints that would be difficult to express using the mechanisms of context-free grammars alone. Our next step, therefore, is to specify a way to integrate feature structures and unification operations into the specification of a grammar. This can be accomplished by *augmenting* the rules of ordinary context-free grammars with attachments that specify feature structures for the constituents of the rules, along with appropriate unification operations that express constraints on those constituents. From a grammatical point of view, these attachments will be used to accomplish the following goals:

- to associate complex feature structures with both lexical items and instances of grammatical categories
- to guide the composition of feature structures for larger grammatical constituents based on the feature structures of their component parts
- to enforce compatibility constraints between specified parts of grammatical constructions

We will use the following notation to denote the grammar augmentations that will allow us to accomplish all of these goals, based on the PATR-II system described in Shieber (1986):

$$\beta_0 \rightarrow \beta_1 \cdots \beta_n$$
$$\{set\ of\ constraints\}$$

The specified constraints have one of the following forms.

$$\langle \beta_i\ feature\ path \rangle = Atomic\ value$$
$$\langle \beta_i\ feature\ path \rangle = \langle \beta_j\ feature\ path \rangle$$

The notation $\langle \beta_i$ *feature path*$\rangle$ denotes a feature path through the feature structure associated with the β_i component of the context-free part of the rule. The first style of constraint specifies that the value found at the end of the given path must unify with the specified atomic value. The second form specifies that the values found at the end of the two given paths must be unifiable.

To illustrate the use of these constraints, let us return to the informal solution to the number agreement problem proposed at the beginning of this chapter.

$S \rightarrow NP\ VP$

Only if the number of the NP is equal to the number of the VP.

Using the new notation, this rule can now be expressed as follows.

$S \rightarrow NP\ VP$

$\langle NP$ NUMBER$\rangle$ = $\langle VP$ NUMBER$\rangle$

Note that in cases where there are two or more constituents of the same syntactic category in a rule, we will subscript the constituents to keep them straight, as in $VP \rightarrow V\ NP_1\ NP_2$.

Taking a step back from the notation, it is important to note that in this approach the simple generative nature of context-free rules has been fundamentally changed by this augmentation. Ordinary context-free rules are based on the simple notion of concatenation; an *NP* followed by a *VP* is an *S*, or generatively, to produce an *S* all we need to do is concatenate an *NP* to a *VP*. In the new scheme, this concatenation must be accompanied by a successful unification operation. This leads naturally to questions about the computational complexity of the unification operation and its effect on the generative power of this new grammar. These issues will be discussed in Ch. 15.

To review, there are two fundamental components to this approach.

- The elements of context-free grammar rules will have feature-based constraints associated with them. This reflects a shift from atomic grammatical categories to more complex categories with properties.
- The constraints associated with individual rules can make reference to the feature structures associated with the parts of the rule to which they are attached.

The following sections present applications of unification constraints to four interesting linguistic phenomena: agreement, grammatical heads, subcategorization, and long-distance dependencies.

16.3.1 Agreement

As discussed in Ch. 12, agreement phenomena show up in a number of different places in English. This section illustrates how unification can be used to capture the two main types of English agreement phenomena: subject-verb agreement and determiner-nominal agreement. We will use the following ATIS sentences as examples throughout this discussion to illustrate these phenomena.

(16.8) This flight serves breakfast.

(16.9) Does this flight serve breakfast?

(16.10) Do these flights serve breakfast?

Notice that the constraint used to enforce SUBJECT-VERB agreement given above is deficient in that it ignores the PERSON feature. The following constraint which makes use of the AGREEMENT feature takes care of this problem.

$S \rightarrow NP\ VP$

$\langle NP\ \text{AGREEMENT}\rangle = \langle VP\ \text{AGREEMENT}\rangle$

Examples 16.9 and 16.10 illustrate a minor variation on SUBJECT-VERB agreement. In these yes-no-questions, the subject *NP* must agree with the auxiliary verb, rather than the main verb of the sentence, which appears in a non-finite form. This agreement constraint can be handled by the following rule.

$S \rightarrow Aux\ NP\ VP$

$\langle Aux\ \text{AGREEMENT}\rangle = \langle NP\ \text{AGREEMENT}\rangle$

Agreement between determiners and nominals in noun phrases is handled in a similar fashion. The basic task is to permit the expressions given above, but block the unwanted **this flights* and **those flight* expressions where the determiners and nominals clash in their NUMBER feature. Again, the logical place to enforce this constraint is in the grammar rule that brings the parts together.

NP → *Det Nominal*
⟨*Det* AGREEMENT⟩ = ⟨*Nominal* AGREEMENT⟩
⟨*NP* AGREEMENT⟩ = ⟨*Nominal* AGREEMENT⟩

This rule states that the AGREEMENT feature of the *Det* must unify with the AGREEMENT feature of the *Nominal*, and moreover, that the AGREEMENT feature of the *NP* must also unify with the *Nominal*.

Having expressed the constraints needed to enforce subject-verb and determiner-nominal agreement, we must now fill in the rest of the machinery needed to make these constraints work. Specifically, we must consider how the various constituents that take part in these constraints (the *Aux*, *VP*, *NP*, *Det*, and *Nominal*) acquire values for their various agreement features.

We can begin by noting that our constraints involve both lexical and non-lexical constituents. The simpler lexical constituents, *Aux* and *Det*, receive values for their respective agreement features directly from the lexicon as in the following rules.

Aux → *do*
⟨*Aux* AGREEMENT NUMBER⟩ = *pl*
⟨*Aux* AGREEMENT PERSON⟩ = *3rd*

Aux → *does*
⟨*Aux* AGREEMENT NUMBER⟩ = *sg*
⟨*Aux* AGREEMENT PERSON⟩ = *3rd*

Det → *this*
⟨*Det* AGREEMENT NUMBER⟩ = *sg*

Det → *these*
⟨*Det* AGREEMENT NUMBER⟩ = *pl*

Returning to our first *S* rule, let us first consider the AGREEMENT feature for the *VP* constituent. The constituent structure for this *VP* is specified by the following rule.

VP → *Verb NP*

It seems clear that the agreement constraint for this constituent must be based on its constituent verb. This verb, as with the previous lexical entries, can acquire its agreement feature values directly from lexicon as in the following rules.

Verb → *serve*
⟨*Verb* AGREEMENT NUMBER⟩ = *pl*

Verb → *serves*
⟨*Verb* AGREEMENT NUMBER⟩ = *sg*
⟨*Verb* AGREEMENT PERSON⟩ = *3rd*

All that remains is to stipulate that the agreement feature of the parent *VP* is constrained to be the same as its verb constituent.

VP → *Verb NP*
⟨*VP* AGREEMENT⟩ = ⟨*Verb* AGREEMENT⟩

In other words, non-lexical grammatical constituents can acquire values for at least some of their features from their component constituents.

The same technique works for the remaining *NP* and *Nominal* categories. The values for the agreement features for these categories are derived from the nouns *flight* and *flights*.

Noun → *flight*
⟨*Noun* AGREEMENT NUMBER⟩ = *sg*

Noun → *flights*
⟨*Noun* AGREEMENT NUMBER⟩ = *pl*

Nominal features can be constrained to have the same values as their constituent nouns.

Nominal → *Noun*
⟨*Nominal* AGREEMENT⟩ = ⟨*Noun* AGREEMENT⟩

Note that this section has only scratched the surface of the English agreement system, and that the agreement system of other languages can be considerably more complex than English.

16.3.2 Head Features

To account for the way that compositional grammatical constituents such as noun phrases, nominals, and verb phrases come to have agreement features, the preceding section introduced the notion of copying feature structures from phrase structure children to their parents. This turns out to be a specific instance of a much more general phenomenon in constraint-based grammars. Specifically, the features for most grammatical categories are copied from *one* of the children to the parent. The child that provides the features is called the **head of the phrase**, and the features copied are referred to as **head features**.

HEAD OF THE PHRASE

HEAD FEATURES

This idea of heads, first introduced in Sec. **??**, plays an important role in constraint-based grammars. Consider the following three rules from the last section.

$$VP \longrightarrow Verb\ NP$$
$$\langle VP\ \textsc{agreement}\rangle = \langle Verb\ \textsc{agreement}\rangle$$

$$NP \longrightarrow Det\ Nominal$$
$$\langle Det\ \textsc{agreement}\rangle = \langle Nominal\ \textsc{agreement}\rangle$$
$$\langle NP\ \textsc{agreement}\rangle = \langle Nominal\ \textsc{agreement}\rangle$$

$$Nominal \longrightarrow Noun$$
$$\langle Nominal\ \textsc{agreement}\rangle = \langle Noun\ \textsc{agreement}\rangle$$

In each of these rules, the constituent providing the agreement feature structure to its parent is the head of the phrase. More specifically, the verb is the head of the verb phrase, the nominal is the head of the noun phrase, and the noun is the head of the nominal. As a result, we can say that the agreement feature structure is a head feature. We can rewrite our rules to reflect these generalizations by placing the agreement feature structure under a HEAD feature and then copying that feature upward as in the following constraints.

(16.11)
$$VP \longrightarrow Verb\ NP$$
$$\langle VP\ \textsc{head}\rangle = \langle Verb\ \textsc{head}\rangle$$

(16.12)
$$NP \longrightarrow Det\ Nominal$$
$$\langle NP\ \textsc{head}\rangle = \langle Nominal\ \textsc{head}\rangle$$
$$\langle Det\ \textsc{head agreement}\rangle = \langle Nominal\ \textsc{head agreement}\rangle$$

(16.13)
$$Nominal \longrightarrow Noun$$
$$\langle Nominal\ \textsc{head}\rangle = \langle Noun\ \textsc{head}\rangle$$

Similarly, the lexical entries that introduce these features must now reflect this HEAD notion, as in the following.

$$Noun \longrightarrow flights$$
$$\langle Noun\ \textsc{head agreement number}\rangle = pl$$

$\textit{Verb} \rightarrow \textit{serves}$

$\langle \textit{Verb}\ \textsc{head agreement number} \rangle = \textit{sg}$

$\langle \textit{Verb}\ \textsc{head agreement person} \rangle = \textit{3rd}$

16.3.3 Subcategorization

Recall that subcategorization is the notion that verbs can be picky about the patterns of arguments they will allow themselves to appear with. In Ch. 12, to prevent the generation of ungrammatical sentences with verbs and verb phrases that do not match, we were forced to split the category of verb into multiple sub-categories. These more specific verb categories were then used in the definition of the specific verb phrases that they were allowed to occur with, as in the following.

$\textit{Verb-with-S-comp} \rightarrow \textit{think}$

$\textit{VP} \rightarrow \textit{Verb-with-S-comp S}$

Clearly, this approach introduces exactly the same undesirable proliferation of categories that we saw with the similar approach to solving the number problem. The proper way to avoid this proliferation is to introduce feature structures to distinguish among the various members of the verb category. This goal can be accomplished by associating an atomic feature called SUBCAT, with an appropriate value, with each of the verbs in the lexicon. For example, the transitive version of *serves* could be assigned the following feature structure in the lexicon.

$\textit{Verb} \rightarrow \textit{serves}$

$\langle \textit{Verb}\ \textsc{head agreement number} \rangle = \textit{sg}$

$\langle \textit{Verb}\ \textsc{head subcat} \rangle = \textit{trans}$

The SUBCAT feature is a signal to the rest of the grammar that this verb should only appear in verb phrases with a single noun phrase argument. This constraint is enforced by adding corresponding constraints to all the verb phrase rules in the grammar, as in the following.

$\textit{VP} \rightarrow \textit{Verb}$

$\langle \textit{VP}\ \textsc{head} \rangle = \langle \textit{Verb}\ \textsc{head} \rangle$

$\langle \textit{VP}\ \textsc{head subcat} \rangle = \textit{intrans}$

$\textit{VP} \rightarrow \textit{Verb NP}$

$\langle \textit{VP}\ \textsc{head} \rangle = \langle \textit{Verb}\ \textsc{head} \rangle$

$\langle \textit{VP}\ \textsc{head subcat} \rangle = \textit{trans}$

$\textit{VP} \rightarrow \textit{Verb NP NP}$

⟨*VP* HEAD⟩ = ⟨*Verb* HEAD⟩

⟨*VP* HEAD SUBCAT⟩ = *ditrans*

The first unification constraint in these rules states that the verb phrase receives its HEAD features from its verb constituent, while the second constraint specifies what the value of that SUBCAT feature must be. Any attempt to use a verb with an inappropriate verb phrase will fail since the value of the SUBCAT feature of the *VP* will fail to unify with the atomic symbol given in the second constraint. Note that this approach requires unique symbols for each of the 50–100 verb phrase frames in English.

This is a somewhat clumsy approach since these unanalyzable SUBCAT symbols do not directly encode either the number or type of the arguments that the verb expects to take. To see this, note that one can not simply examine a verb's entry in the lexicon and know what its subcategorization frame is. Rather, you must use the value of the SUBCAT feature indirectly as a pointer to those verb phrase rules in the grammar that can accept the verb in question.

A more elegant solution, which makes better use of the expressive power of feature structures, allows the verb entries to directly specify the order and category type of the arguments they require. The following entry for *serves* is an example of one such approach, in which the verb's subcategory feature expresses a **list** of its objects and complements.

Verb → *serves*

⟨*Verb* HEAD AGREEMENT NUMBER⟩ = *sg*

⟨*Verb* HEAD SUBCAT FIRST CAT⟩ = *NP*

⟨*Verb* HEAD SUBCAT SECOND⟩ = *end*

This entry uses the FIRST feature to state that the first post-verbal argument must be an *NP*; the value of the SECOND feature indicates that this verb expects only one argument. A verb like *leave Boston in the morning*, with two arguments, would have the following kind of entry.

Verb → *leaves*

⟨*Verb* HEAD AGREEMENT NUMBER⟩ = *sg*

⟨*Verb* HEAD SUBCAT FIRST CAT⟩ = *NP*

⟨*Verb* HEAD SUBCAT SECOND CAT⟩ = *PP*

⟨*Verb* HEAD SUBCAT THIRD⟩ = *end*

This scheme is, of course, a rather baroque way of encoding a list; it is also possible to use the idea of **types** defined in Sec. 16.6 to define a list type more cleanly.

The individual verb phrase rules must now check for the presence of exactly the elements specified by their verb, as in the following transitive rule.

(16.14)
$$\begin{aligned}
&VP \;\rightarrow\; Verb\ NP\\
&\quad\langle VP\ \text{HEAD}\rangle = \langle Verb\ \text{HEAD}\rangle\\
&\quad\langle VP\ \text{HEAD SUBCAT FIRST CAT}\ \rangle = \langle NP\ \text{CAT}\ \rangle\\
&\quad\langle VP\ \text{HEAD SUBCAT SECOND}\rangle = \textit{end}
\end{aligned}$$

The second constraint in this rule's constraints states that the category of the first element of the verb's SUBCAT list must match the category of the constituent immediately following the verb. The third constraint goes on to state that this verb phrase rule expects only a single argument.

SUBCATEGORIZATION FRAME

Our previous examples have shown rather simple subcategorization structures for verbs. In fact, verbs can subcategorize for quite complex **subcategorization frames**, (e.g., *NP PP*, *NP NP*, or *NP S*) and these frames can be composed of many different phrasal types. In order to come up with a list of possible subcategorization frames for English verbs, we first need to have a list of possible phrase types that can make up these frames. Fig. 16.3 shows one short list of possible phrase types for making up subcategorization frames for verbs; this list is modified from one used to create verb subcategorization frames in the FrameNet project (Johnson, 1999; Baker et al., 1998), and includes phrase types for special subjects of verbs like *there* and *it*, as well as for objects and complements.

To use the phrase types in Fig. 16.3 in a unification grammar, each phrase type could be described using features. For example the form **VPto**, which is subcategorized for by *want* might be expressed as:

$$\begin{aligned}
&Verb \rightarrow\; want\\
&\quad\langle Verb\ \text{HEAD SUBCAT FIRST CAT}\rangle = VP\\
&\quad\langle Verb\ \text{HEAD SUBCAT FIRST FORM}\rangle = \textit{infinitive}
\end{aligned}$$

Each of the 50 to 100 possible verb subcategorization frames in English would be described as a list drawn from these phrase types. For example, here is an example of the two-complement *want*. We can use this example to demonstrate two different notational possibilities. First, lists can be represented via an angle brackets notation ⟨ and ⟩. Second, instead of using a rewrite-rule annotated with path equations, we can represent the lexical entry as a single feature structure:

$$\begin{bmatrix}
\text{ORTH} & \textit{want}\\
\text{CAT} & \textit{Verb}\\
\text{HEAD} & \begin{bmatrix}\text{SUBCAT} & \left\langle \begin{bmatrix}\text{CAT}\ \textit{NP}\end{bmatrix}, \begin{bmatrix}\text{CAT}\ \textit{VP}\\ \text{HEAD}\ \begin{bmatrix}\text{VFORM}\ \textit{infinitival}\end{bmatrix}\end{bmatrix}\right\rangle\end{bmatrix}
\end{bmatrix}$$

Noun Phrase Types		
There	nonreferential there	**There** *is still much to learn*
It	nonreferential it	**It** *was evident that my ideas*
NP	noun phrase	*As he was relating* **his story**
Preposition Phrase Types		
PP	preposition phrase	*couch their message* **in terms**
PPing	gerundive PP	*censured him* **for not having intervened**
PPpart	particle	*turn it* **off**
Verb Phrase Types		
VPbrst	bare stem VP	*she could* **discuss it**
VPto	to-marked infin. VP	*Why do you want* **to know***?*
VPwh	wh-VP	*it is worth considering* **how to write**
VPing	gerundive VP	*I would consider* **using it**
Complement Clause types		
Finite Clause		
Sfin	finite clause	*maintain* **that the situation was unsatisfactory**
Swh	wh-clause	*it tells us* **where we are**
Sif	whether/if clause	*ask* **whether Aristophanes is depicting a**
Nonfinite Clause		
Sing	gerundive clause	*see* **some attention being given**
Sto	to-marked clause	*know* **themselves to be relatively unhealthy**
Sforto	for-to clause	*She was waiting* **for him to make some reply**
Sbrst	bare stem clause	*commanded* **that his sermons be published**
Other Types		
AjP	adjective phrase	*thought it* **possible**
Quo	quotes	*asked* **"What was it like?"**

Figure 16.3 A small set of potential phrase types which can be combined to create a set of potential subcategorization frames for verbs. Modified from the FrameNet tagset (Johnson, 1999; Baker et al., 1998). The sample sentence fragments are from the British National Corpus.

Combining even a limited set of phrase types results in a very large set of possible subcategorization frames. Furthermore, each verb allows many different subcategorization frames. For example, here are just some of the subcategorization patterns for the verb *ask*, with examples from the BNC:

Subcat	**Example**
Quo	asked [$_{Quo}$ “What was it like?”]
NP	asking [$_{NP}$ a question]
Swh	asked [$_{Swh}$ what trades you’re interested in]
Sto	ask [$_{Sto}$ him to tell you]
PP	that means asking [$_{PP}$ at home]
Vto	asked [$_{Vto}$ to see a girl called Evelyn]
NP Sif	asked [$_{NP}$ him] [$_{Sif}$ whether he could make]
NP NP	asked [$_{NP}$ myself] [$_{NP}$ a question]
NP Swh	asked [$_{NP}$ him] [$_{Swh}$ why he took time off]

CONTROL

A number of comprehensive subcategorization-frame tagsets exist, such as the COMLEX set (Macleod et al., 1998), which includes subcategorization frames for verbs, adjectives, and nouns, and the ACQUILEX tagset of verb subcategorization frames (Sanfilippo, 1993). Many subcategorization-frame tagsets add other information about the complements, such as specifying the identity of the implicit subject in a lower verb phrase that has no overt subject; this is called **control** information. For example *Temmy promised Ruth to go* (at least in some dialects) implies that Temmy will do the going, while *Temmy persuaded Ruth to go* implies that Ruth will do the going. Some of the multiple possible subcategorization frames for a verb can be partially predicted by the semantics of the verb; for example many verbs of transfer (like *give*, *send*, *carry*) predictably take the two subcategorization frames *NP NP* and *NP PP*:

NP NP	sent FAA Administrator James Busey a letter
NP PP	sent a letter to the chairman of the Armed Services Committee

ALTERNATIONS

These relationships between subcategorization frames across classes of verbs are called argument-structure **alternations**, and will be discussed in Ch. 19 when we discuss the semantics of verbal argument structure. Ch. 14 will introduce probabilities for modeling the fact that verbs generally have preferences even among the different subcategorization frames they allow.

Subcategorization in Other Parts of Speech

VALENCE

Although the notion of subcategorization, or **valence** as it is often called, was originally designed for verbs, more recent work has focused on the fact that many other kinds of words exhibit forms of valence-like behavior. Consider the following contrasting uses of the prepositions *while* and *during*.

(16.15) Keep your seatbelt fastened while *we are taking off*.

(16.16) *Keep your seatbelt fastened while *takeoff*.

(16.17) Keep your seatbelt fastened during *takeoff*.

(16.18) *Keep your seatbelt fastened during *we are taking off*.

Despite the apparent similarities between these words, they make quite different demands on their arguments. Representing these differences is left as Exercise 16.5 for the reader.

Many adjectives and nouns also have subcategorization frames. Here are some examples using the adjectives *apparent*, *aware*, and *unimportant* and the nouns *assumption* and *question*:

It was **apparent** [$_{Sfin}$ that the kitchen was the only room. . .]
It was **apparent** [$_{PP}$ from the way she rested her hand over his]
aware [$_{Sfin}$ he may have caused offense]
it is **unimportant** [$_{Swheth}$ whether only a little bit is accepted]
the **assumption** [$_{Sfin}$ that wasteful methods have been employed]
the **question** [$_{Swheth}$ whether the authorities might have decided]

See Macleod et al. (1998) and Johnson (1999) for descriptions of subcategorization frames for nouns and adjectives.

Verbs express subcategorization constraints on their subjects as well as their complements. For example, we need to represent the lexical fact that the verb *seem* can take an **Sfin** as its subject (*That she was affected seems obvious*), while the verb *paint* cannot. The SUBJECT feature can be used to express these constraints.

16.3.4 Long-Distance Dependencies

The model of subcategorization we have developed so far has two components. Each head word has a SUBCAT feature which contains a list of the complements it expects. Then phrasal rules like the *VP* rule in (16.15) match up each expected complement in the SUBCAT list with an actual constituent. This mechanism works fine when the complements of a verb are in fact to be found in the verb phrase.

Sometimes, however, a constituent subcategorized for by the verb is not locally instantiated, but stands in a **long-distance** relationship with its predicate. Here are some examples of such **long-distance dependencies**:

LONG-DISTANCE DEPENDENCIES

What cities does Continental service?
What flights do you have from Boston to Baltimore?
What time does that flight leave Atlanta?

In the first example, the constituent *what cities* is subcategorized for by the verb *service*, but because the sentence is an example of a **wh-non-subject-question**, the object is located at the front of the sentence. Recall from Ch. 12 that a (simple) phrase-structure rule for a **wh-non-subject-question** is something like the following:

$$S \rightarrow \textit{Wh-NP Aux NP VP}$$

Now that we have features, we can augment this phrase-structure rule to require the *Aux* and the *NP* to agree (since the *NP* is the subject). But we also need some way to augment the rule to tell it that the *Wh-NP* should fill some subcategorization slot in the *VP*. The representation of such long-distance dependencies is a quite difficult problem, because the verb whose subcategorization requirement is being filled can be quite distant from the filler. In the following (made-up) sentence, for example, the *wh*-phrase *which flight* must fill the subcategorization requirements of the verb *book*, despite the fact that there are two other verbs (*want* and *have*) in between:

Which flight do you want me to have the travel agent book?

GAP LIST

FILLER

Many solutions to representing long-distance dependencies in unification grammars involve keeping a list, often called a **gap list**, implemented as a feature GAP, which is passed up from phrase to phrase in the parse tree. The **filler** (for example *which flight* above) is put on the gap list, and must eventually be unified with the subcategorization frame of some verb. See Sag and Wasow (1999) for an explanation of such a strategy, together with a discussion of the many other complications that must be modeled in long-distance dependencies.

16.4 Implementing Unification

As discussed, the unification operator takes two feature structures as input and returns a single merged feature structure if successful, or a failure signal if the two inputs are not compatible. The input feature structures are represented as directed acyclic graphs (DAGs), where features are depicted as labels on directed edges, and feature values are either atomic symbols or DAGs. As we will see, the implementation of the operator is a relatively straightforward recursive graph matching algorithm, suitably tailored to accommodate the various requirements of unification. Roughly speaking, the algorithm loops through the features in one input and attempts to find a corresponding feature in the other. If all of the respective feature values match, then the unification is successful. If there is a mismatch then the unification fails. Not surprisingly, the recursion is motivated by the need to correctly match those features that have feature structures as their values.

A notable aspect of this algorithm is that rather than constructing a new feature structure with the unified information from the two arguments, it destructively alters the arguments so that in the end they point to exactly the same information. Thus, the result of a successful call to the unification operator consists of suitably altered versions of the arguments. As is discussed in the next section, the destructive nature of this algorithm necessitates certain minor extensions to the simple graph version of feature structures as DAGs we have been assuming.

16.4.1 Unification Data Structures

To facilitate the destructive merger aspect of the algorithm, we add a small complication to the DAGs used to represent the input feature structures; feature structures are represented using DAGs with additional edges, or fields. Specifically, each feature structure consists of two fields: a content field and a pointer field. The content field may be null or contain an ordinary feature structure. Similarly, the pointer field may be null or contain a pointer to another feature structure. If the pointer field of the DAG is null, then the content field of the DAG contains the actual feature structure to be processed. If, on the other hand, the pointer field is non-null, then the destination of the pointer represents the actual feature structure to be processed. The merger aspects of unification will be achieved by altering the pointer field of DAGs during processing.

To see how this works, let's consider the extended DAG representation for the following familiar feature structure.

(16.19) $$\begin{bmatrix} \text{NUMBER} & \textit{sg} \\ \text{PERSON} & \textit{3rd} \end{bmatrix}$$

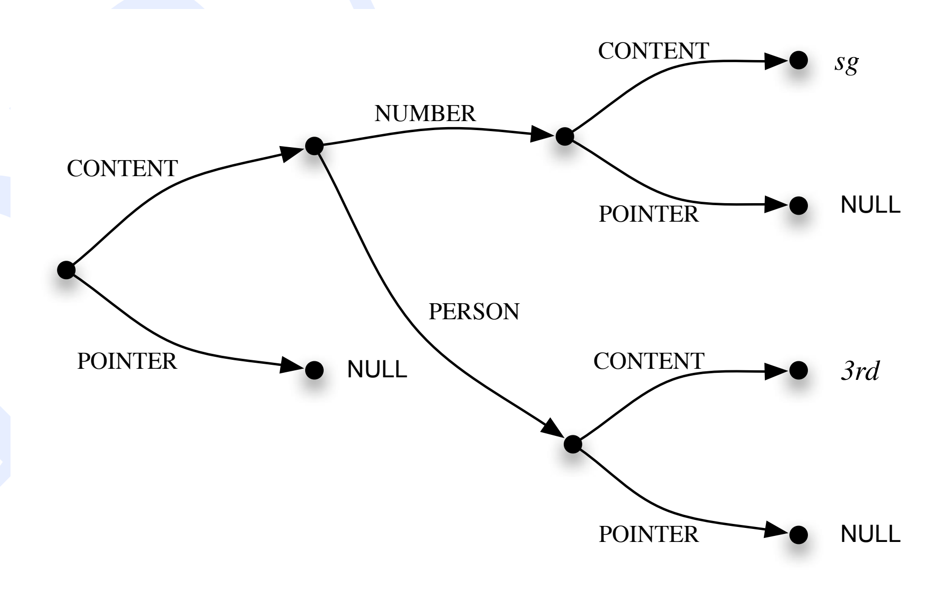


Figure 16.4 An extended DAG notation for Example 16.19.

Figure 16.4 shows this extended representation in its graphical form. Note that the extended representation contains content and pointer links both for the top-level layer of features, as well as for each of the embedded feature structures all the way down to the atomic values.

Before presenting the details of the unification algorithm, let's illustrate the use of this extended DAG representation with the following simple example. The original extended representation of the arguments to this unification are shown in Figure 16.5.

(16.20) $$\begin{bmatrix}\textsc{number} & \textit{sg}\end{bmatrix} \sqcup \begin{bmatrix}\textsc{person} & \textit{3rd}\end{bmatrix} = \begin{bmatrix}\textsc{number} & \textit{sg}\\ \textsc{person} & \textit{3rd}\end{bmatrix}$$

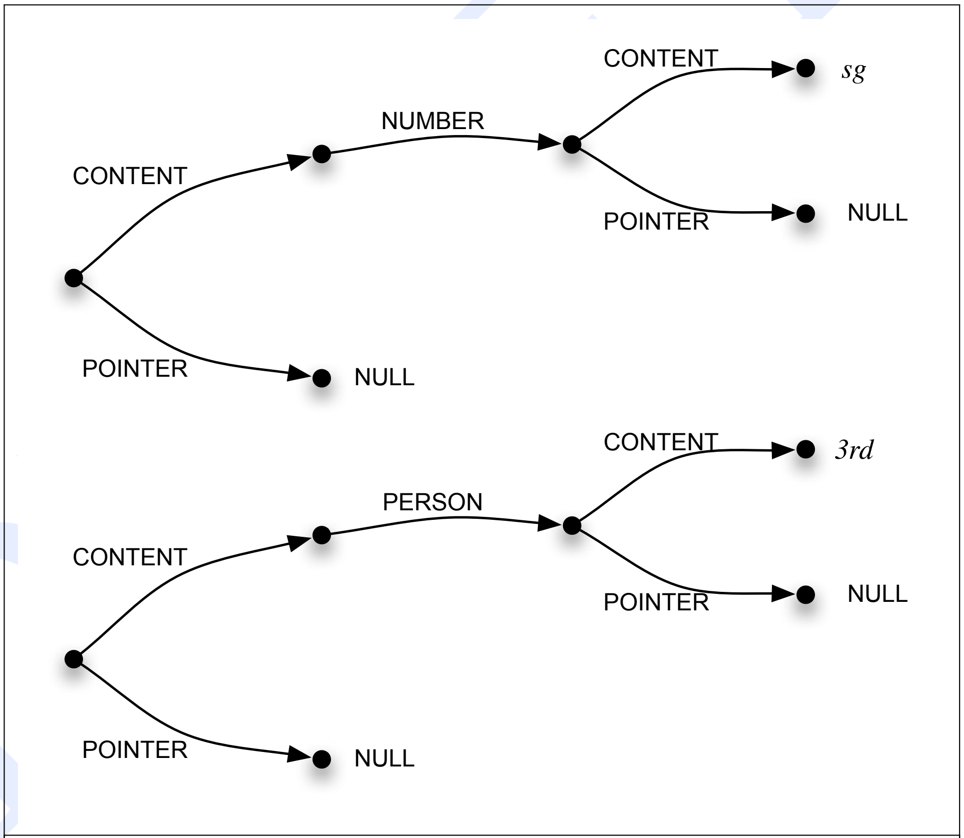


Figure 16.5 The original arguments to Example 16.20.

At a basic level, this unification results in the creation of a new structure containing the union of the information from the two original arguments. With the extended notation, we can see how the unification is accomplished by making some additions to the original arguments and changing some of the pointers from one structure to the other so that in the end they contain the same content. In this example, this is accomplished by first adding a PERSON feature to the first argument, and assigning it a value by filling its POINTER field with a pointer to the appropriate location in the second argument, as shown in Figure 16.6.

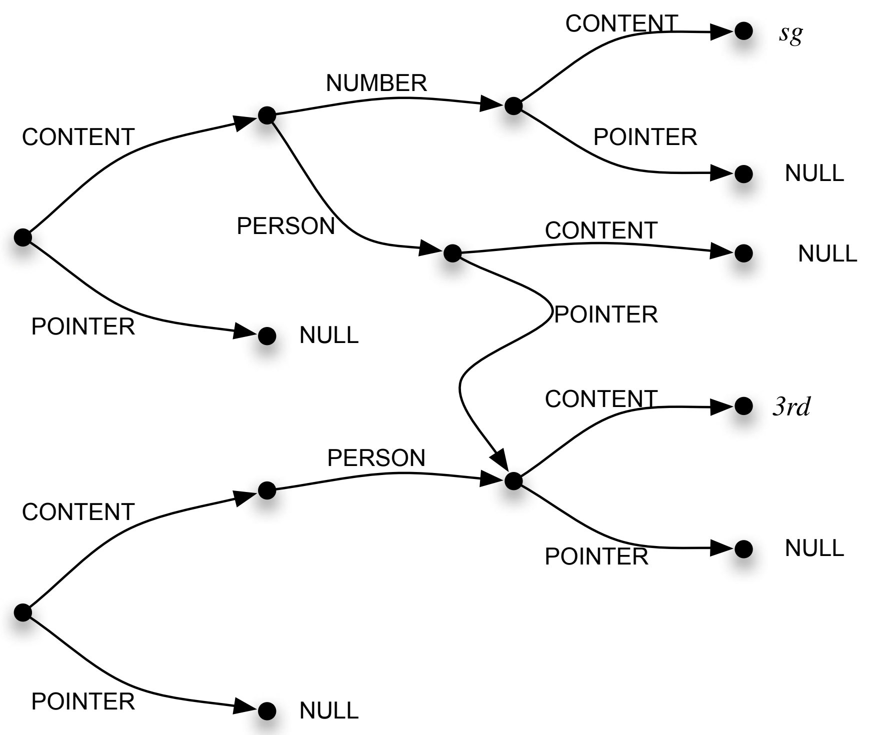


Figure 16.6 The arguments after assigning the first argument's new PERSON feature to the appropriate value in the second argument.

The process is, however, not yet complete. While it is clear from Figure 16.6 that the first argument now contains all the correct information, the second one does not; it lacks a NUMBER feature. We could, of course, add a NUMBER feature to this argument with a pointer to the appropriate place in the first one. This change would result in the two arguments having all the correct information from this unification. Unfortunately, this solution is inadequate since it does not meet our requirement that the two arguments be truly unified. Since the two arguments are not completely unified at the top level, future unifications involving one of the arguments would not show up in the other. The solution to this problem is to simply set the POINTER field of the second argument to point at the first one. When this is done any future change to either argument will be immediately reflected in both. The result of this final change is shown in Figure 16.7.

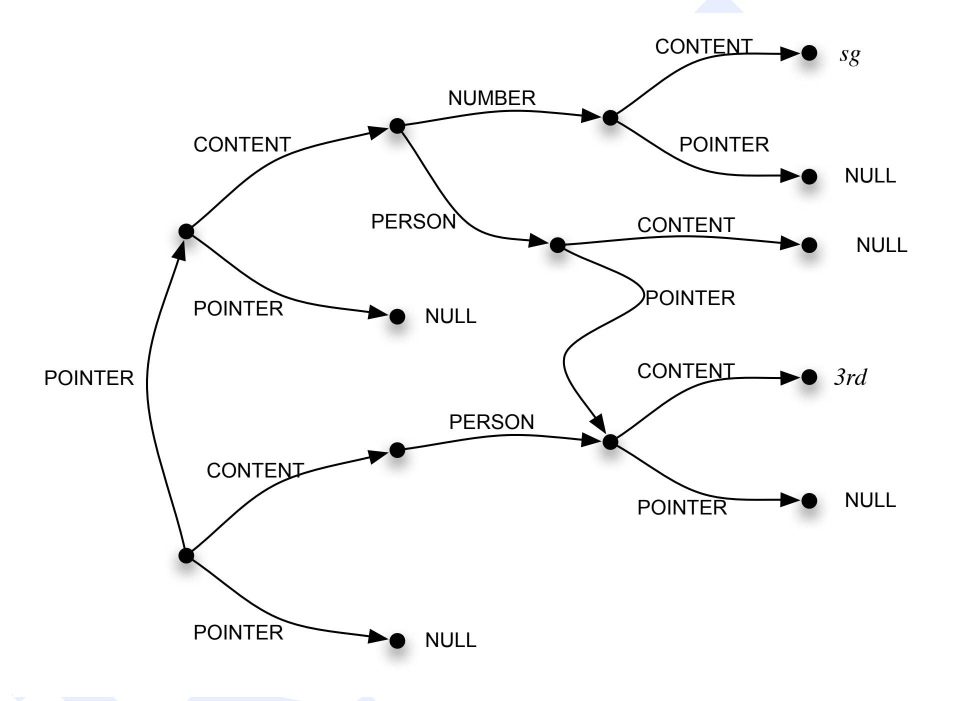


Figure 16.7 The final result of unifying F1 and F2.

16.4.2 The Unification Algorithm

The unification algorithm that we have been leading up to is shown in Figure 16.8. This algorithm accepts two feature structures represented using the extended DAG representation and returns as its value a modified version of one of the arguments, or a failure signal in the event that the feature structures are incompatible.

DEREFERENCING

The first step in this algorithm is to acquire the true contents of both of the arguments. Recall that if the pointer field of an extended feature structure is non-null, then the real content of that structure is found by following the pointer found in pointer field. The variables *f1* and *f2* are the result of this pointer following process, often referred to as **dereferencing**.

As with all recursive algorithms, the next step is to test for the various base

```
function UNIFY(f1-orig, f2-orig) returns f-structure or failure

 f1 ← Dereferenced contents of f1-orig
 f2 ← Dereferenced contents of f2-orig

 if f1 and f2 are identical then
   f1.pointer ← f2
   return f2
 else if f1 is null then
   f1.pointer ← f2
   return f2
 else if f2 is null then
   f2.pointer ← f1
   return f1
 else if both f1 and f2 are complex feature structures then
   f2.pointer ← f1
   for each f2-feature in f2 do
     f1-feature ← Find or create a corresponding feature in f1
     if UNIFY(f1-feature.value, f2-feature.value) returns failure then
       return failure
   return f1
 else return failure
```

Figure 16.8 The unification algorithm.

cases of the recursion before proceeding on to a recursive call involving some part of the original arguments. In this case, there are three possible base cases:

- The arguments are identical
- One or both of the arguments has a null value
- The arguments are non-null and non-identical

If the structures are identical, then the pointer of the first is set to the second and the second is returned. It is important to understand why this pointer change is done in this case. After all, since the arguments are identical, returning either one would appear to suffice. This might be true for a single unification but recall that we want the two arguments to the unification operator to be truly unified. The pointer change is necessary since we want the arguments to be truly identical, so that any subsequent unification that adds information to one will add it to both.

In the case where either of the arguments is null, the pointer field for the null argument is changed to point to the other argument, which is then returned. The result is that both structures now point at the same value.

If neither of the preceding tests is true then there are two possibilities: they are non-identical atomic values, or they are non-identical complex structures. The former case signals an incompatibility in the arguments that leads the algorithm to return a failure signal. In the latter case, a recursive call is needed to ensure that the component parts of these complex structures are compatible. In this implementation, the key to the recursion is a loop over all the features of the *second* argument, *f2*. This loop attempts to unify the value of each feature in *f2* with the corresponding feature in *f1*. In this loop, if a feature is encountered in *f2* that is missing from *f1*, a feature is added to *f1* and given the value NULL. Processing then continues as if the feature had been there to begin with. If *every* one of these unifications succeeds, then the pointer field of *f2* is set to *f1* completing the unification of the structures and *f1* is returned as the value of the unification.

An Example

To illustrate this algorithm, let's walk through the following example.

(16.21)
$$\begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix}\text{NUMBER} & \textit{sg}\end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \boxed{1}\end{bmatrix}\end{bmatrix} \sqcup \begin{bmatrix}\text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \begin{bmatrix}\text{PERSON} & \textit{3rd}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 16.9 shows the extended representations for the arguments to this unification. These original arguments are neither identical, nor null, nor atomic, so the main loop is entered. Looping over the features of *f2*, the algorithm is led to a recursive attempt to unify the values of the corresponding SUBJECT features of *f1* and *f2*.

$$\begin{bmatrix}\text{AGREEMENT} & \boxed{1}\end{bmatrix} \sqcup \begin{bmatrix}\text{AGREEMENT} & \begin{bmatrix}\text{PERSON} & \textit{3rd}\end{bmatrix}\end{bmatrix}$$

These arguments are also non-identical, non-null, and non-atomic so the loop is entered again leading to a recursive check of the values of the AGREEMENT features.

$$\begin{bmatrix}\text{NUMBER} & \textit{sg}\end{bmatrix} \sqcup \begin{bmatrix}\text{PERSON} & \textit{3rd}\end{bmatrix}$$

In looping over the features of the second argument, the fact that the first argument lacks a PERSON feature is discovered. A PERSON feature initialized with a NULL value is, therefore, added to the first argument. This, in effect, changes the previous unification to the following.

$$\begin{bmatrix}\text{NUMBER} & \textit{sg} \\ \text{PERSON} & \textit{null}\end{bmatrix} \sqcup \begin{bmatrix}\text{PERSON} & \textit{3rd}\end{bmatrix}$$

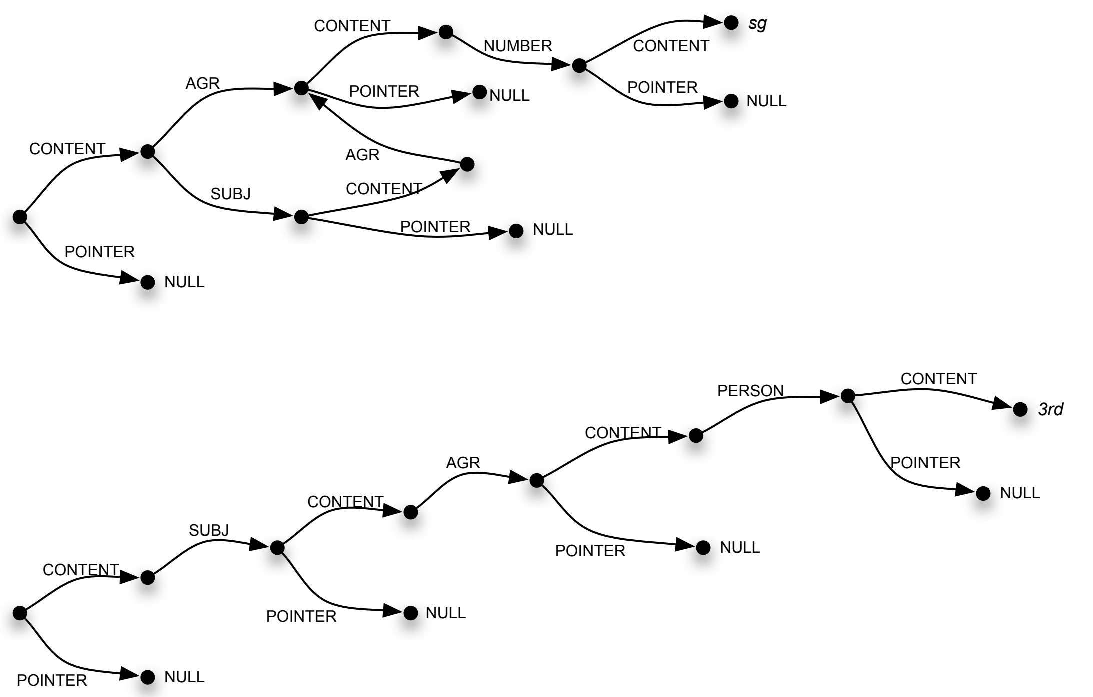


Figure 16.9 The initial arguments *f1* and *f2* to Example 16.21.

After creating this new PERSON feature, the next recursive call leads to the unification of the NULL value of the new feature in the first argument with the *3rd* value of the second argument. Since there are no further features to check in the *f2* argument at any level of recursion, each of the recursive calls to UNIFY returns. The result is shown in Figure 16.10.

16.5 Parsing with Unification Constraints

We now have all the pieces necessary to integrate feature structures and unification into a parser. Fortunately, the order-independent nature of unification allows us to largely ignore the actual search strategy used in the parser. Once we have associated unification constraints with the context-free rules of the grammar, and feature structures with the states of the search, any of the standard search algorithms described in Ch. 13 can be used.

Of course, this leaves a fairly large range of possible implementation strate-

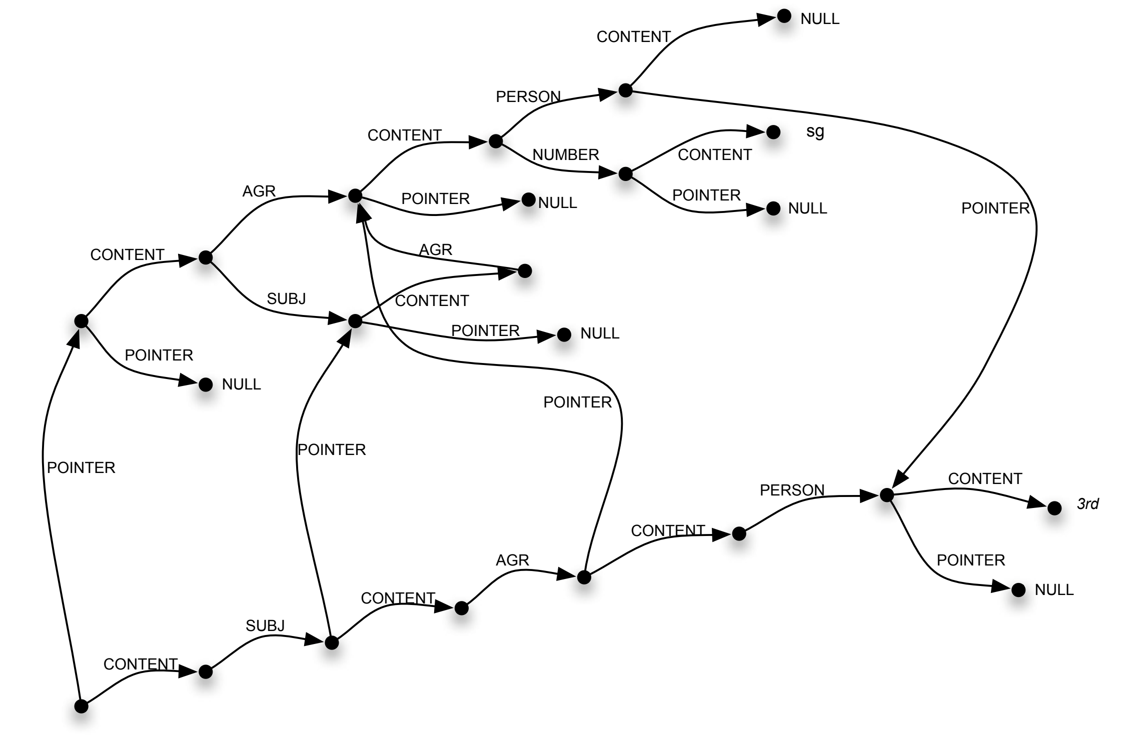


Figure 16.10 The final structures of *f1* and *f2* at the end.

gies. We could, for example, simply parse as we did before using the context-free components of the rules, and then build the feature structures for the resulting trees after the fact, filtering out those parses that contain unification failures. Although such an approach would result in only well-formed structures in the end, it fails to use the power of unification to reduce the size of the parser's search space during parsing.

The next section describes an approach that makes better use of the power of unification by integrating unification constraints directly into the Earley parsing process, allowing ill-formed structures to be eliminated as soon as they are proposed. As we will see, this approach requires only minimal changes to the basic Earley algorithm. We then move on to briefly consider an approach to unification-based parsing that moves even further away from standard context-free methods.

16.5.1 Integrating Unification into an Earley Parser

We have two goals in integrating feature structures and unification into the Earley algorithm: to use feature structures to provide a richer representation for the constituents of the parse, and to block entry into the chart of ill-formed constituents that violate unification constraints. As we will see, these goals can be accomplished by fairly minimal changes to the original Earley scheme given on page **??**.

The first change involves the various representations used in the original code. Recall that the Earley algorithm operates by using a set of unadorned context-free grammar rules to fill in a data-structure called a chart with a set of states. At the end of the parse, the states that make up this chart represent all possible parses of the input. Therefore, we begin our changes by altering the representations of both the context-free grammar rules, and the states in the chart.

The rules are altered so that in addition to their current components, they also include a feature structure derived from their unification constraints. More specifically, we will use the constraints listed with a rule to build a feature structure, represented as a DAG, for use with that rule during parsing.

Consider the following context-free rule with unification constraints.

$$S \rightarrow NP\ VP$$

$$\langle NP\ \textsc{head agreement}\rangle = \langle VP\ \textsc{head agreement}\rangle$$

$$\langle S\ \textsc{head}\rangle = \langle VP\ \textsc{head}\rangle$$

Converting these constraints into a feature structure results in the following structure:

$$\begin{bmatrix} \textsc{s} & \begin{bmatrix}\textsc{head} & \boxed{1}\end{bmatrix} \\ \textsc{np} & \begin{bmatrix}\textsc{head} & \begin{bmatrix}\textsc{agreement} & \boxed{2}\end{bmatrix}\end{bmatrix} \\ \textsc{vp} & \begin{bmatrix}\textsc{head} & \boxed{1}\begin{bmatrix}\textsc{agreement} & \boxed{2}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

In this derivation, we combined the various constraints into a single structure by first creating top-level features for each of the parts of the context-free rule, S, NP, and VP in this case. We then add further components to this structure by following the path equations in the constraints. Note that this is a purely notational conversion; the DAGs and the constraint equations contain the same information. However, tying the constraints together in a single feature structure puts it in a form that can be passed directly to our unification algorithm.

The second change involves the states used to represent partial parses in the Earley chart. The original states contain fields for the context-free rule being used, the position of the dot representing how much of the rule has been completed, the positions of the beginning and end of the state, and a list of other states that represent the completed sub-parts of the state. To this set of fields, we simply add an additional field to contain the DAG representing the feature structure corresponding to the state. Note that when a rule is first used by PREDICTOR to create a state, the DAG associated with the state will simply consist of the DAG retrieved from the rule. For example, when PREDICTOR uses the above *S* rule to enter a state into the chart, the DAG given above will be its initial DAG. We'll denote states like this as follows, where *Dag* denotes the feature structure given above.

$$S \rightarrow \bullet\, NP\ VP,\ [0,0], [], Dag$$

Given these representational additions, we can move on to altering the algorithm itself. The most important change concerns the actions that take place when a new state is created by the extension of an existing state, which takes place in the COMPLETER routine. Recall that COMPLETER is called when a completed constituent has been added to the chart. Its task is to attempt to find, and extend, existing states in the chart that are looking for constituents that are compatible with the newly completed constituent. COMPLETER is, therefore, a function that creates new states by *combining* the information from two other states, and as such is a likely place to apply the unification operation.

To be more specific, COMPLETER adds a new state into the chart by finding an existing state whose • can be advanced by the newly completed state. A • can be advanced when the category of the constituent immediately following it matches the category of the newly completed constituent. To accommodate the use of feature structures, we can alter this scheme by unifying the feature structure associated with the newly completed state with the appropriate part of the feature structure being advanced. If this unification succeeds, then the DAG of the new state receives the unified structure and is entered into the chart. If it fails, then no new state is entered into the chart. The appropriate alterations to COMPLETER are shown in Figure 16.11.

Consider this process in the context of parsing the phrase *That flight*, where the *That* has already been seen, as is captured by the following state.

$$NP \rightarrow Det \bullet Nominal[0,1], [S_{Det}], Dag_1$$

$$Dag_1 \begin{bmatrix} \text{NP} & \begin{bmatrix} \text{HEAD} & \boxed{1} \end{bmatrix} \\ \text{DET} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{AGREEMENT} \boxed{2} \begin{bmatrix} \text{NUMBER} & \text{SG} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{NOMINAL} & \begin{bmatrix} \text{HEAD} & \boxed{1} \begin{bmatrix} \text{AGREEMENT} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Now consider the later situation where the parser has processed *flight* and has subsequently produced the following state.

$$Nominal \rightarrow Noun \bullet, [1,2], [S_{Noun}], Dag_2$$

$$Dag_2 \begin{bmatrix} \text{NOMINAL} & \begin{bmatrix} \text{HEAD} & \boxed{1} \end{bmatrix} \\ \text{NOUN} & \begin{bmatrix} \text{HEAD} & \boxed{1} \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SG} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

To advance the *NP* rule, the parser unifies the feature structure found under the NOMINAL feature of Dag_2, with the feature structure found under the NOMINAL feature of the *NP*'s Dag_1. As in the original algorithm, a new state is created to represent the fact that an existing state has been advanced. This new state's DAG is given the DAG that resulted from this unification.

The final change to the original algorithm concerns the check for states already contained in the chart. In the original algorithm, the ENQUEUE function refused to enter into the chart any state that was *identical* to one already present in the chart. "Identical" meant the same rule, with the same start and finish positions, and the same position of the •. It is this check that allows the algorithm to, among other things, avoid the infinite recursion problems associated with left-recursive rules.

The problem, of course, is that our states are now more complex since they have complex feature structures associated with them. States that appeared identical under the original criteria might in fact now be different since their associated DAGs may differ. One solution to this problem is to extend the identity check to include the DAGs associated with the states, but it turns out that we can improve on this solution.

The motivation for the improvement lies in the motivation for the identity check. Its purpose is to prevent the wasteful addition of a state into the chart whose effect on the parse would be accomplished by an already existing state. Put another way, we want to prevent the entry into the chart of any state that would duplicate the work that will eventually be done by other states. Of course, this will clearly be the case with identical states, but it turns out it is also the case for states in the chart that are *more general* than new states being considered.

Consider the situation where the chart contains the following state, where the *Dag* places no constraints on the *Det*.

$$NP \rightarrow \bullet Det\ NP, [i,i], [], Dag$$

Such a state simply says that it is expecting a *Det* at position *i*, and that any *Det* will do.

Now consider the situation where the parser wants to insert a new state into the chart that is identical to this one, with the exception that its DAG restricts the

```
function Earley-Parse(words, grammar) returns chart

  AddToChart(($\gamma \rightarrow \bullet S,\ [0,0],\ dag_{\gamma}$), chart[0])
  for i ← from 0 to Length(words) do
   for each state in chart[i] do
     if Incomplete?(state) and
              Next-Cat(state) is not a part of speech then
        Predictor(state)
     elseif Incomplete?(state) and
              Next-Cat(state) is a part of speech then
        Scanner(state)
     else
        Completer(state)
   end
  end
  return(chart)

 procedure Predictor(($A \rightarrow \alpha \bullet B\,\beta,\ [i,j],\ dag_A$))
    for each ($B \rightarrow \gamma$) in Grammar-Rules-For(B, grammar) do
        AddToChart(($B \rightarrow \bullet\,\gamma,\ [j,j],\ dag_B$), chart[j])
    end

 procedure Scanner(($A \rightarrow \alpha \bullet B\,\beta,\ [i,j],\ dag_A$))
    if B ∈ Parts-of-Speech(word[j]) then
       AddToChart(($B \rightarrow word[j]\bullet,\ [j,j+1],\ dag_B$), chart[j+1])

 procedure Completer(($B \rightarrow \gamma\ \bullet,\ [j,k],\ dag_B$))
    for each ($A \rightarrow \alpha \bullet B\,\beta,\ [i,j],\ dag_A$) in chart[j] do
      if new-dag ← Unify-States($dag_B, dag_A, B$) $\neq$ Fails!
        AddToChart(($A \rightarrow \alpha\ B \bullet \beta,\ [i,k]$, new-dag), chart[k])
    end

 procedure Unify-States(dag1, dag2, cat)
   dag1-cp ← CopyDag(dag1)
   dag2-cp ← CopyDag(dag2)
   Unify(Follow-Path(cat, dag1-cp), Follow-Path(cat, dag2-cp))

 procedure AddToChart(state, chart-entry)
    if state is not subsumed by a state in chart-entry then
       Push-On-End(state, chart-entry)
    end
```

Figure 16.11 Modifications to the Earley algorithm to include unification.

Det to be singular. In this case, although the states in question are not identical, the addition of the new state to the chart would accomplish nothing and should therefore be prevented.

To see this let's consider all the cases. If the new state is added, then a subsequent singular *Det* will match both rules and advance both. Due to the unification of features, both will have DAGs indicating that their *Det*s are singular, with the net result being duplicate states in the chart. If on the other hand, a plural *Det* is encountered, the new state will reject it and not advance, while the old rule will advance, entering a single new state into the chart. On the other hand, if the new state is not placed in the chart, a subsequent plural or singular *Det* will match the more general state and advance it, leading to the addition of one new state into the chart. Note that this leaves us in exactly the same situation as if the new state had been entered into the chart, with the exception that the duplication is avoided. In sum, nothing worthwhile is accomplished by entering into the chart a state that is more specific than a state already in the chart.

Fortunately, the notion of **subsumption** introduced earlier gives us a formal way to talk about the generalization and specialization relations among feature structures. This suggests that the proper way to alter ENQUEUE is to check whether a newly created state is *subsumed* by any existing states in the chart. If it is, then it will not be allowed into the chart. More specifically, a new state that is identical in terms of its rule, start and finish positions, subparts, and • position, to an existing state, will be not be entered into the chart if its DAG is subsumed by the DAG of an existing state (ie. if $Dag_{old} \sqsubseteq Dag_{new}$). The necessary change to the original Earley ENQUEUE procedure is shown in Figure 16.11.

The Need for Copying

The calls to COPYDAG within the UNIFY-STATE procedure require some elaboration. Recall that one of the strengths of the Earley algorithm (and of the dynamic programming approach in general) is that once states have been entered into the chart they may be used again and again as part of different derivations, including ones that in the end do not lead to successful parses. This ability is the motivation for the fact that states already in the chart are not updated to reflect the progress of their •, but instead are copied and then updated, leaving the original states intact so that they can be used again in further derivations.

The call to COPYDAG in UNIFY-STATE is required to preserve this behavior because of the destructive nature of our unification algorithm. If we simply unified the DAGs associated with the existing states, those states would be altered by the unification, and hence would not be available in the same form for subsequent uses by the COMPLETER function. Note that this has negative consequences regardless of whether the unification succeeds or fails, since in either case the original states are altered.

Let's consider what would happen if the call to COPYDAG was absent in the following example where an early unification attempt fails.

(16.22) Show me morning flights.

Let's assume that our parser has the following entry for the ditransitive version of the verb *show*, as well as the following transitive and ditransitive verb phrase rules.

$$
\begin{array}{l}
\mathit{Verb} \rightarrow \mathit{show} \\
\quad \langle \mathit{Verb}\ \textsc{head subcat first cat} \rangle = \mathit{NP} \\
\quad \langle \mathit{Verb}\ \textsc{head subcat second cat} \rangle = \mathit{NP} \\
\quad \langle \mathit{Verb}\ \textsc{head subcat third} \rangle = \mathit{END}
\end{array}
$$

$$
\begin{array}{l}
\mathit{VP} \rightarrow \mathit{Verb}\ \mathit{NP} \\
\quad \langle \mathit{VP}\ \textsc{head} \rangle = \langle \mathit{Verb}\ \textsc{head} \rangle \\
\quad \langle \mathit{VP}\ \textsc{head subcat first cat} \rangle = \langle \mathit{NP}\ \textsc{cat} \rangle \\
\quad \langle \mathit{VP}\ \textsc{head subcat second} \rangle = \mathit{END}
\end{array}
$$

$$
\begin{array}{l}
\mathit{VP} \rightarrow \mathit{Verb}\ \mathit{NP}\ \mathit{NP} \\
\quad \langle \mathit{VP}\ \textsc{head} \rangle = \langle \mathit{Verb}\ \textsc{head} \rangle \\
\quad \langle \mathit{VP}\ \textsc{head subcat first cat} \rangle = \langle \mathit{NP}_1\ \textsc{cat} \rangle \\
\quad \langle \mathit{VP}\ \textsc{head subcat second cat} \rangle = \langle \mathit{NP}_2\ \textsc{cat} \rangle \\
\quad \langle \mathit{VP}\ \textsc{head subcat third} \rangle = \mathit{END}
\end{array}
$$

When the word *me* is read, the state representing the transitive verb phrase will be completed since its dot has moved to the end. COMPLETER will, therefore, call UNIFY-STATES before attempting to enter this complete state into the chart. This will fail since the SUBCAT structures of these two rules can not be unified. This is, of course, exactly what we want since this version of *show* is ditransitive. Unfortunately, because of the destructive nature of our unification algorithm we have already altered the DAG attached to the state representing *show*, as well as the one attached to the *VP* thereby ruining them for use with the correct verb phrase rule later on. Thus, to make sure that states can be used again and again with multiple derivations, copies are made of the dags associated with states before attempting any unifications involving them.

All of this copying can be quite expensive. As a result, a number of alternative techniques have been developed that attempt to minimize this cost (Pereira, 1985; Karttunen and Kay, 1985; Tomabechi, 1991; Kogure, 1990). Kiefer et al. (1999b) and Penn and Munteanu (2003) describe a set of related techniques used to speed up a large unification-based parsing system.

16.5.2 Unification-Based Parsing

A more radical approach to using unification in parsing can be motivated by looking at an alternative way of denoting our augmented grammar rules. Consider the following *S* rule that we have been using throughout this chapter.

$$S \rightarrow NP\ VP$$
$$\langle NP\ \textsc{head agreement}\rangle = \langle VP\ \textsc{head agreement}\rangle$$
$$\langle S\ \textsc{head}\rangle = \langle VP\ \textsc{head}\rangle$$

An interesting way to alter the context-free part of this rule is to change the way its grammatical categories are specified. In particular, we can place the categorical information about the parts of the rule inside the feature structure, rather than inside the context-free part of the rule. A typical instantiation of this approach would give us the following rule (Shieber, 1986).

$$X_0 \rightarrow X_1\ X_2$$
$$\langle X_0\ \textsc{cat}\rangle = S$$
$$\langle X_1\ \textsc{cat}\rangle = NP$$
$$\langle X_2\ \textsc{cat}\rangle = VP$$
$$\langle X_1\ \textsc{head agreement}\rangle = \langle X_2\ \textsc{head agreement}\rangle$$
$$\langle\ X_0\ \textsc{head}\rangle = \langle X_2\ \textsc{head}\rangle$$

Focusing solely on the context-free component of the rule, this rule now simply states that the X_0 constituent consists of two components, and that the X_1 constituent is immediately to the left of the X_2 constituent. The information about the actual categories of these components is placed inside the rule's feature structure; in this case, indicating that X_0 is an *S*, X_1 is an *NP*, and X_2 is a *VP*. Altering the Earley algorithm to deal with this notational change is trivial. Instead of seeking the categories of constituents in the context-free components of the rule, it simply needs to look at the CAT feature in the DAG associated with a rule.

Of course, since it is the case that these two rules contain precisely the same information, it isn't clear that there is any benefit to this change. To see the potential benefit of this change, consider the following rules.

$$X_0 \rightarrow X_1\ X_2$$
$$\langle X_0\ \textsc{cat}\rangle = \langle\ X_1\ \textsc{cat}\rangle$$
$$\langle X_2\ \textsc{cat}\rangle = PP$$

$$X_0 \rightarrow X_1\ and\ X_2$$
$$\langle X_1\ \textsc{cat}\rangle = \langle\ X_2\ \textsc{cat}\rangle$$
$$\langle X_0\ \textsc{cat}\rangle = \langle\ X_1\ \textsc{cat}\rangle$$

The first rule is an attempt to generalize over various rules that we have already seen, such as *NP* → *NP PP* and *VP* → *VP PP*. It simply states that any category can be followed by a prepositional phrase, and that the resulting constituent has the same category as the original. Similarly, the second rule is an attempt to generalize over rules such as *S* → *S and S*, *NP* → *NP and NP*, and so on.[1] It states that any constituent can be conjoined with a constituent of the same category to yield a new category of the same kind. What these rules have in common is their use of phrase structure rules that contain constituents with constrained, but unspecified, categories, something that can not be accomplished with our old rule format.

Of course, since these rules rely on the use the CAT feature, their effect could be approximated in the old format by simply enumerating all the various instantiations of the rule. A more compelling case for the new approach is motivated by the existence of grammatical rules, or constructions, that contain constituents that are not easily characterized using any existing syntactic category.

Consider the following examples of the English HOW-MANY construction from the WSJ (Jurafsky, 1992).

(16.23) **How early** does it open?

(16.24) **How deep** is her Greenness?

(16.25) **How papery** are your profits?

(16.26) **How quickly** we forget.

(16.27) **How many of you** can name three famous sporting Blanchards?

As is illustrated in these examples, the HOW-MANY construction has two components: the lexical item *how*, and a lexical item or phrase that is rather hard to characterize syntactically. It is this second element that is of interest to us here. As these examples show, it can be an adjective, adverb, or some kind of quantified phrase (although not all members of these categories yield grammatical results). Clearly, a better way to describe this second element is as a *scalar* concept, a constraint can be captured using feature structures, as in the following rule.

$$X_0 \rightarrow X_1\ X_2$$
$$\langle X_1\ \text{ORTH}\rangle = \langle \textit{how}\rangle$$
$$\langle X_2\ \text{SEM}\rangle = \langle\ \text{SCALAR}\rangle$$

A complete account of rules like this involves semantics and will therefore have to wait for Ch. 17. The key point here is that by using feature structures a grammatical rule can place constraints on its constituents in a manner that does not make any use of the notion of a syntactic category.

[1] These rules should not be mistaken for correct, or complete, accounts of the phenomena in question.

Of course, dealing this kind of rule requires some changes to our parsing scheme. All of the parsing approaches we have considered thus far are driven by the syntactic category of the various constituents in the input. More specifically, they are based on simple atomic matches between the categories that have been predicted, and categories that have been found. Consider, for example, the operation of the COMPLETER function shown in Figure 16.11. This function searches the chart for states that can be advanced by a newly completed state. It accomplishes this by matching the category of the newly completed state against the category of the constituent following the • in the existing state. Clearly this approach will run into trouble when there are no such categories to consult.

The remedy for this problem with COMPLETER is to search the chart for states whose DAGs *unify* with the DAG of the newly completed state. This eliminates any requirement that states or rules have a category. The PREDICTOR can be changed in a similar fashion by having it add states to the chart states whose X_0 DAG component can unify with the constituent following the • of the predicting state. Exercise 16.6 asks you to make the necessary changes to the pseudo-code in Figure 16.11 to effect this style of parsing. Exercise 16.7 asks you to consider some of the implications of these alterations, particularly with respect to prediction.

16.6 TYPES AND INHERITANCE

I am surprised that ancient and modern writers have not attributed greater importance to the laws of inheritance. . .

Alexis de Tocqueville, *Democracy in America*, 1840

The basic feature structures we have presented so far have two problems that have led to extensions to the formalism. The first problem is that there is no way to place a constraint on what can be the value of a feature. For example, we have implicitly assumed that the NUMBER attribute can take only *sg* and *pl* as values. But in our current system, there is nothing, for example, to stop NUMBER from have the value *3rd* or *feminine* as values:

$$\begin{bmatrix}\textsc{number} & \textsc{feminine}\end{bmatrix}$$

This problem has caused many unification-based grammatical theories to add various mechanisms to try to constrain the possible values of a feature. Formalisms like Functional Unification Grammar (FUG) (Kay, 1979, 1984, 1985) and Lexical Functional Grammar (LFG) (Bresnan, 1982), for example, focused on ways to keep intransitive verb like *sneeze* from unifying with a direct object (*Marin sneezed Toby*). This was addressed in FUG by adding a special atom **none** which is not allowed to unify with anything, and in LFG by adding **coherence** condi-

NONE

tions which specified when a feature should not be filled. The Generalized Phrase Structure Grammar (GPSG) (Gazdar et al., 1985, 1988) added a class of **feature co-occurrence restrictions**, to prevent, for example, nouns from having some verbal properties.

The second problem with simple feature structures is that there is no way to capture generalizations across them. For example, the many types of English verb phrases described in the Subcategorization section on page 16 share many features, as do the many kinds of subcategorization frames for verbs. Syntacticians were looking for ways to express these generalities.

TYPES

A general solution to both of these problems is the use of **types**. Type systems for unification grammars have the following characteristics:

1. Each feature structure is labeled by a type.
2. Conversely, each type has **appropriateness conditions** expressing which features are appropriate for it and what types of values then can take.
3. The types are organized into a **type hierarchy**, in which more specific types inherit properties of more abstract ones.
4. The unification operation is modified to unify the types of feature structures in addition to unifying the attributes and values.

APPROPRIATENESS

TYPE HIERARCHY

TYPED FEATURE STRUCTURE

SIMPLE TYPES

COMPLEX TYPES

In such **typed feature structure** systems, types are a new class of objects, just like attributes and values were for standard feature structures. Types come in two kinds: **simple types** (also called **atomic types**), and **complex types**. Let's begin with simple types. A simple type is an atomic symbol like **sg** or **pl** (we will use **boldface** for all types), and replaces the simple atomic values used in standard feature structures. All types are organized into a multiple-inheritance **type hierarchy** (a kind of **partial order** called a **lattice**). Fig. 16.12 shows the type hierarchy for the new type **agreement**, which will be the type of the kind of atomic object that can be the value of an AGREE feature.

SUBTYPE

In the hierarchy in Fig. 16.12, **3rd** is a **subtype** of **agr**, and **3-sg** is a **subtype** of both **3rd** and **sg**. Types can be unified in the type hierarchy; the unification of any two types is the most-general type that is more specific than the two input types. Thus:

$$\mathbf{3rd} \sqcup \mathbf{sg} = \mathbf{3sg}$$
$$\mathbf{1st} \sqcup \mathbf{pl} = \mathbf{1pl}$$
$$\mathbf{1st} \sqcup \mathbf{agr} = \mathbf{1st}$$
$$\mathbf{3rd} \sqcup \mathbf{1st} = \textit{undefined}$$

FAIL TYPE

The unification of two types which do not have a defined unifier is undefined, although it is also possible to explicitly represent this **fail type** using the symbol ⊥ (Aït-Kaci, 1984).

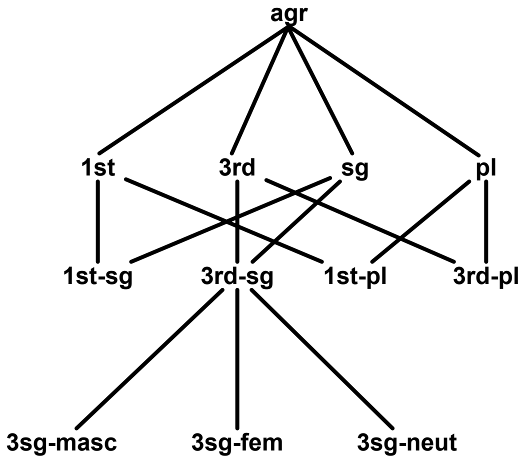


Figure 16.12 A simple type hierarchy for the subtypes of type **agr** which can be the value of the AGREE attribute. After Carpenter (1992).

The second kind of types are complex types, which specify:

- a set of features that are appropriate for that type
- restrictions on the values of those features (expressed in terms of types)
- equality constraints between the values

Consider a simplified representation of the complex type **verb**, which just represents agreement and verbal morphology information. A definition of **verb** would define the two appropriate features, AGREE and VFORM, and would also define the type of the values of the two features. Let's suppose that the AGREE feature takes values of type **agr** defined in Fig. 16.12 above, and the VFORM feature takes values of type **vform** where **vform** subsumes the seven subtypes **finite**, **infinitive**, **gerund**, **base**, **present-participle**, **past-participle**, and **passive-participle**. Thus **verb** would be defined as follows (where the convention is to indicate the type either at the top of the AVM or just to the lower left of the left bracket):

$$\begin{bmatrix} \textbf{verb} & \\ \textsc{agree} & \textbf{agr} \\ \textsc{vform} & \textbf{vform} \end{bmatrix}$$

By contrast, the type **noun** might be defined with the AGREE feature, but without the VFORM feature:

$$\begin{bmatrix} \textbf{noun} & \\ \textsc{agree} & \textbf{agr} \end{bmatrix}$$

The unification operation is augmented for typed feature structures just by requiring that the types of the two structures unify in addition to the values of the component features unifying.

$$\begin{bmatrix}\textbf{verb} & \\ \text{AGREE} & \textbf{1st} \\ \text{VFORM} & \textbf{gerund}\end{bmatrix} \sqcup \begin{bmatrix}\textbf{verb} & \\ \text{AGREE} & \textbf{sg} \\ \text{VFORM} & \textbf{gerund}\end{bmatrix} = \begin{bmatrix}\textbf{verb} & \\ \text{AGREE} & \textbf{1-sg} \\ \text{VFORM} & \textbf{gerund}\end{bmatrix}$$

Complex types are also part of the type hierarchy. Subtypes of complex types inherit all the features of their parents, together with the constraints on their values. Sanfilippo (1993), for example, uses a type hierarchy to encode the hierarchical structure of the lexicon. Fig. 16.13 shows a small part of this hierarchy, the part that models the various subcategories of verbs which take sentential complements; these are divided into the transitive ones (which take direct objects: (*ask yourself whether you have become better informed*) and the intransitive ones (*Monsieur asked whether I wanted to ride*). The type **trans-comp-cat** would introduce the required direct object, constraining it to be of type **noun-phrase**, while types like **sbase-comp-cat** would introduce the baseform (bare stem) complement and constrain its vform to be the baseform.

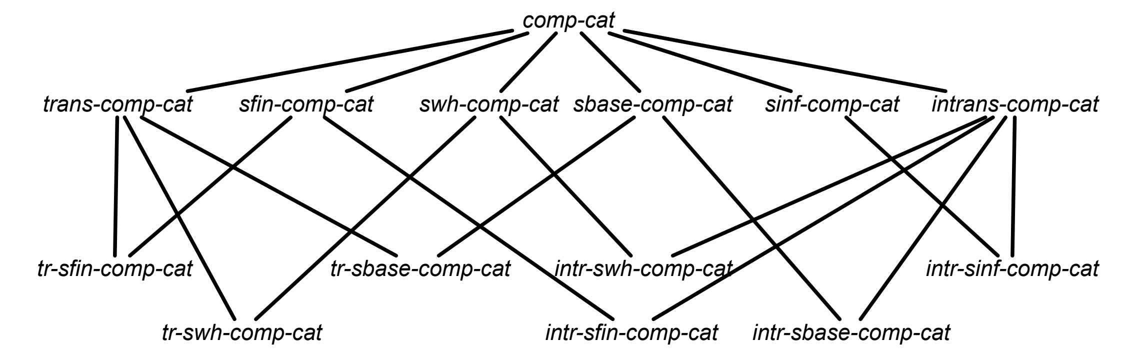


Figure 16.13 Part of the type hierarchy for the verb type **verb-cat**, showing the subtypes of the **comp-cat** type. These are all subcategories of verbs which take sentential complements. After Sanfilippo (1993).

16.6.1 Advanced: Extensions to Typing

DEFAULTS

Typed feature structures can be extended by allowing for inheritance with **defaults**. Default systems have mainly been used in lexical type hierarchies of the sort described in the previous section, in order to encode generalizations and subregular exceptions to them. In early versions of default unification the operation was order-

PRIORITY UNION

dependent, based on the **priority union** operation (Kaplan, 1987). More recent architectures are order-independent (Lascarides and Copestake, 1997; Young and Rounds, 1993), related to Reiter's default logic (Reiter, 1980).

LEXICAL RULE

Many unification-based theories of grammar, including HPSG (Pollard and Sag, 1987, 1994) and LFG (Bresnan, 1982) use an additional mechanism besides inheritance for capturing lexical generalizations: the **lexical rule**. Lexical rules (Jackendoff, 1975) express lexical generalizations by allowing a reduced, hence more redundancy-free lexicon to be automatically expanded by the rules. See Pollard and Sag (1994) for examples, Carpenter (1991) on complexity issues, and Meurers and Minnen (1997) on efficient implementation. Conversely, see Krieger and Nerbonne (1993) on using the type hierarchy to replace lexical rules.

Types can also be used to represent constituency. Rules like (16.12) on page 15 used a normal phrase structure rule template and added the features via path equations. Instead, it's possible to represent the whole phrase structure rule as a type. In order to do this, we need a way to represent constituents as features. One way to do this, following Sag and Wasow (1999), is to take a type **phrase** which has a feature called DTRS ("daughters"), whose value is a list of **phrases**. For example the phrase *I love New York* could have the following representation, (showing only the DTRS feature):

$$\begin{bmatrix}\textbf{phrase} \\ \textsc{dtrs}\ \left\langle \begin{bmatrix}\textsc{cat}\ \mathit{PRO}\\ \textsc{orth}\ \mathit{I}\end{bmatrix}, \begin{bmatrix}\textsc{cat}\ \mathit{VP}\\ \textsc{dtrs}\ \left\langle \begin{bmatrix}\textsc{cat}\ \mathit{Verb}\\ \textsc{orth}\ \mathit{love}\end{bmatrix}, \begin{bmatrix}\textsc{cat}\ \mathit{NP}\\ \textsc{orth}\ \mathit{New\ York}\end{bmatrix}\right\rangle\end{bmatrix}\right\rangle\end{bmatrix}$$

16.6.2 Other Extensions to Unification

PATH INEQUATIONS

NEGATION

SET-VALUED FEATURES

DISJUNCTION

There are many other extensions to unification besides typing, including **path inequations** (Moshier, 1988; Carpenter, 1992; Carpenter and Penn, 1994), **negation** (Johnson, 1988, 1990), **set-valued features** (Pollard and Moshier, 1990), and **disjunction** (Kay, 1979; Kasper and Rounds, 1986). In some unification systems these operations are incorporated into feature structures. Kasper and Rounds (1986) and others, by contrast, implement them in a separate metalanguage which is used to *describe* feature structures. This idea derives from the work of Pereira and Shieber (1984), and even earlier work by Kaplan and Bresnan (1982), all of whom distinguished between a metalanguage for describing feature structures and the actual feature structures themselves. The descriptions may thus use negation and disjunction to describe a set of feature structures (i.e., a certain feature must not contain a certain value, or may contain any of a set of values) but an actual instance of a feature structure that meets the description would not have negated or disjoint values.

The unification grammars as described so far have no mechanism for disambiguation. Much recent work in unification grammars has focused on this disambiguation problem, particular via the use of probabilistic augmentations. See the History section for important references.

16.7 Summary

This chapter introduced feature structures and the unification operation which is used to combine them.

- A feature structure is a set of features-value pairs, where features are unanalyzable atomic symbols drawn from some finite set, and values are either atomic symbols or feature structures. They are represented either as **attribute-value matrices** (**AVMs**) or as directed acyclic graphs (**DAGs**), where features are directed labeled edges and feature values are nodes in the graph.
- **Unification** is the operation for both combining information (merging the information content of two feature structures) and comparing information (rejecting the merger of incompatible features).
- A phrase-structure rule can be augmented with feature structures, and with feature constraints expressing relations among the feature structures of the constituents of the rule. **Subcategorization** constraints can be represented as feature structures on head verbs (or other predicates). The elements which are subcategorized for by a verb may appear in the verb phrase or may be realized apart from the verb, as a **long-distance dependency**.
- Feature structures can be **typed**. The resulting **typed feature structures** place constraints on which type of values a given feature can take, and can also be organized into a **type hierarchy** to capture generalizations across types.

Bibliographical and Historical Notes

The use of features in linguistic theory comes originally from phonology. Anderson (1985) credits Jakobson (1939) with being the first to use features (called **distinctive features**) as an ontological type in a theory, drawing on previous uses of features by Trubetskoi (1939) and others. The semantic use of features followed soon after; see Ch. 19 for the history of componential analysis in semantics. Features in syntax were well established by the 1950s and were popularized by Chomsky (1965).

The unification operation in linguistics was developed independently by Kay (1979) (feature structure unification) and Colmerauer (1970, 1975) (term unification) (see page **??**). Both were working in machine translation and looking for a formalism for combining linguistic information which would be reversible. Colmerauer's original Q-system was a bottom-up parser based on a series of rewrite rules which contained logical variables, designed for a English to French machine translation system. The rewrite rules were reversible to allow them to work for both parsing and generation. Colmerauer, Fernand Didier, Robert Pasero, Philippe Roussel, and Jean Trudel designed the Prolog language based on extended Q-systems to full unification based on the resolution principle of Robinson (1965), and implemented a French analyzer based on it (Colmerauer and Roussel, 1996). The modern use of Prolog and term unification for natural language via **Definite Clause Grammars** was based on Colmerauer's (1975) metamorphosis grammars, and was developed and named by Pereira and Warren (1980). Meanwhile Martin Kay and Ron Kaplan had been working with Augmented Transition Network (**ATN**) grammars. An ATN is a Recursive Transition Network (RTN) in which the nodes are augmented with feature registers. In an ATN analysis of a passive, the first NP would be assigned to the subject register, then when the passive verb was encountered, the value would be moved into the object register. In order to make this process reversible, they restricted assignments to registers so that certain registers could only be filled once, that is, couldn't be overwritten once written. They thus moved toward the concepts of logical variables without realizing it. Kay's original unification algorithm was designed for feature structures rather than terms (Kay, 1979). The integration of unification into an Earley-style approach given in Section 16.5 is based on Shieber (1985).

DEFINITE CLAUSE GRAMMARS

ATN

See Shieber (1986) for a clear introduction to unification, and Knight (1989) for a multidisciplinary survey of unification.

Inheritance and appropriateness conditions were first proposed for linguistic knowledge by Bobrow and Webber (1980) in the context of an extension of the KL-ONE knowledge representation system (Brachman and Schmolze, 1985). Simple inheritance without appropriateness conditions was taken up by number of researchers; early users include Jacobs (1985, 1987). Aït-Kaci (1984) borrowed the notion of inheritance in unification from the logic programming community. Typing of feature structures, including both inheritance and appropriateness conditions, was independently proposed by Calder (1987), Pollard and Sag (1987), and Elhadad (1990). Typed feature structures were formalized by King (1989) and Carpenter (1992). There is an extensive literature on the use of type hierarchies in linguistics, particularly for capturing lexical generalizations; besides the papers previously discussed, the interested reader should consult Evans and Gazdar (1996) for a description of the DATR language, designed for defining inheritance networks for linguistic knowledge representation, Fraser and Hudson (1992) for the use of inheritance in a dependency grammar, and Daelemans et al. (1992) for a general overview. Formalisms and systems for the implementation of constraint-based grammars via typed feature structures include the PAGE system using the TDL language (Krieger and Schäfer, 1994), ALE (Carpenter and Penn, 1994), ConTroll (Götz et al., 1997) and LKB (Copestake, 2002).

Efficiency issues in unification parsing are discussed by Kiefer et al. (1999a), Malouf et al. (2000), and Munteanu and Penn (2004).

Grammatical theories based on unification include Lexical Functional Grammar (LFG) (Bresnan, 1982), Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1987, 1994), Construction Grammar (Kay and Fillmore, 1999), and Unification Categorial Grammar (Uszkoreit, 1986).

Much recent computational work on unification grammars has focused on probabilistic augmentations for disambiguation. Key relevant papers include Abney (1997), Goodman (1997), Johnson et al. (1999), Riezler et al. (2000), Geman and Johnson (2002), Riezler et al. (2002, 2003), Kaplan et al. (2004), Miyao and Tsujii (2005), Toutanova et al. (2005), Ninomiya et al. (2006) and Blunsom and Baldwin (2006).

Exercises

16.1 Draw the DAGs corresponding to the AVMs given in Examples 16.1–16.2.

16.2 Consider the following BERP examples, focusing on their use of pronouns.

I want to spend lots of money.
Tell me about Chez-Panisse.
I'd like to take her to dinner.
She doesn't like Italian.

Assuming that these pronouns all belong to the category *Pro*, write lexical and grammatical entries with unification constraints that block the following examples.

*Me want to spend lots of money.
*Tell I about Chez-Panisse.
*I would like to take she to dinner.
*Her doesn't like Italian.

16.3 Draw a picture of the subsumption semilattice corresponding to the feature structures in Examples 16.3 to 16.7. Be sure to include the most general feature structure [].

16.4 Consider the following examples.

The sheep are baaaaing.
The sheep is baaaaing.

Create appropriate lexical entries for the words *the*, *sheep*, and *baaaaing*. Show that your entries permit the correct assignment of a value to the NUMBER feature for the subjects of these examples, as well as their various parts.

16.5 Create feature structures expressing the different SUBCAT frames for *while* and *during* shown on page 21.

16.6 Alter the pseudocode shown in Figure 16.11 so that it performs the more radical kind of unification-based parsing described on page 37.

16.7 Consider the following problematic grammar suggested by Shieber (1985).

$$S \rightarrow T$$
$$\langle T\ \textsc{f} \rangle = \mathrm{a}$$

$$T_1 \rightarrow T_2\ A$$
$$\langle T_1\ \textsc{f} \rangle = \langle T_2\ \textsc{f}\ \textsc{f} \rangle$$

$$S \rightarrow A$$

$$A \rightarrow a$$

Show the first S state entered into the chart using your modified PREDICTOR from the previous exercise, then describe any problematic behavior displayed by PREDICTOR on subsequent iterations. Discuss the cause of the problem and how in might be remedied.

16.8 Using the list approach to representing a verb's subcategorization frame, show how a grammar could handle any number of verb subcategorization frames with only the following two *VP* rules. More specifically, show the constraints that would have to be added to these rules to make this work.

$$VP \rightarrow Verb$$

$$VP \rightarrow VP\ X$$

The solution to this problem involves thinking about a recursive walk down a verb's subcategorization frame. This is a hard problem; you might consult Shieber (1986) if you get stuck.

16.9 Page 43 showed how to use typed feature structures to represent constituency. Use that notation to represent rules 16.12, 16.13, and 16.14 shown on page 15.

Abney, S. P. (1997). Stochastic attribute-value grammars. *Computational Linguistics*, *23*(4), 597–618.

Aït-Kaci, H. (1984). *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially Ordered Types*. Ph.D. thesis, University of Pennsylvania.

Anderson, S. R. (1985). *Phonology in the Twentieth Century*. Cambridge University Press.

Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The Berkeley FrameNet project. In *COLING/ACL-98*, pp. 86–90.

Blunsom, P. and Baldwin, T. (2006). Multilingual deep lexical acquisition for hpsgs via supertagging. In *EMNLP 2006*.

Bobrow, R. J. and Webber, B. L. (1980). Knowledge representation for syntactic/semantic processing. In *AAAI-80*, Stanford, CA, pp. 316–323. Morgan Kaufmann.

Brachman, R. J. and Schmolze, J. G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, *9*(2), 171–216.

Bresnan, J. (Ed.). (1982). *The Mental Representation of Grammatical Relations*. MIT Press.

Calder, J. (1987). Typed unification for natural language processing. In Kahn, G., MacQueen, D., and Plotkin, G. (Eds.), *Categories, Polymorphism, and Unification*. Centre for Cognitive Science, University of Edinburgh, Edinburgh, Scotland†.

Carpenter, B. (1991). The generative power of categorial grammars and head-driven phrase structure grammars with lexical rules. *Computational Linguistics*, *17*(3), 301–313.

Carpenter, B. (1992). *The Logic of Typed Feature Structures*. Cambridge University Press.

Carpenter, B. and Penn, G. (1994). The Attribute Logic Engine Users's Guide Version 2.0.1. Tech. rep., Carnegie Mellon University.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. MIT Press.

Colmerauer, A. (1970). Les systèmes-q ou un formalisme pour analyser et synthétiser des phrase sur ordinateur. Internal publication 43, Département d'informatique de l'Université de Montréal†.

Colmerauer, A. (1975). Les grammaires de métamorphose GIA. Internal publication, Groupe Intelligence artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, Nov 1975. English version, Metamorphosis grammars. In L. Bolc, (Ed.), *Natural Language Communication with Computers, Lecture Notes in Computer Science 63*, Springer Verlag, Berlin, 1978, pp. 133–189.

Colmerauer, A. and Roussel, P. (1996). The birth of Prolog. In Bergin Jr., T. J. and Gibson, Jr., R. G. (Eds.), *History of Programming Languages – II*, pp. 331–352. ACM Press/Addison-Wesley, New York.

Copestake, A. (2002). *Implementing Typed Feature Structure Grammars*. CSLI, Stanford, CA.

Daelemans, W., Smedt, K. D., and Gazdar, G. (1992). Inheritance in natural language processing. *Computational Linguistics*, *18*(2), 205–218.

de Tocqueville, A. (1840). *Democracy in America*. Doubleday, New York. The 1966 translation by George Lawrence.

Elhadad, M. (1990). Types in functional unification grammars. In *Proceedings of the 28th ACL*, Pittsburgh, PA, pp. 157–164. ACL.

Evans, R. and Gazdar, G. (1996). DATR: A language for lexical knowledge representation. *Computational Linguistics*, *22*(2), 167–216.

Fraser, N. M. and Hudson, R. A. (1992). Inheritance in word grammar. *Computational Linguistics*, *18*(2), 133–158.

Gazdar, G., Klein, E., Pullum, G. K., and Sag, I. A. (1985). *Generalized Phrase Structure Grammar*. Basil Blackwell, Oxford.

Gazdar, G., Pullum, G. K., Carpenter, B., Klein, E., Hukari, T. E., and Levine, R. D. (1988). Category structures. *Computational Linguistics*, *14*(1), 1–19.

Geman, S. and Johnson, M. (2002). Dynamic programming for parsing and estimation of stochastic unification-based grammars.. In *ACL-02*, pp. 279–286.

Goodman, J. (1997). Probabilistic feature grammars. In *Proceedings of the International Workshop on Parsing Technology*.

Götz, T., Meurers, W. D., and Gerdemann, D. (1997). The ConTroll manual. Tech. rep., Seminar für Sprachwissenschaft, Universität Tübingen.

Jackendoff, R. (1975). Morphological and semantic regularities in the lexicon. *Language*, *51*(3), 639–671.

Jacobs, P. (1985). *A Knowledge-Based Approach to Language Generation*. Ph.D. thesis, University of California, Berkeley, CA. Available as University of California at Berkeley Computer Science Division Tech. rep. #86/254.

Jacobs, P. (1987). Knowledge-based natural language generation. *Artificial Intelligence*, *33*, 325–378.

Jakobson, R. (1939). Observations sur le classement phonologique des consonnes. In Blancquaert, E. and Pée, W. (Eds.), *Proceedings of the Third International Congress of Phonetic Sciences*, Ghent, pp. 34–41.

Johnson, C. (1999). Syntactic and semantic principles of FrameNet annotation, version 1. Tech. rep. TR-99-018, ICSI, Berkeley, CA.

Johnson, M. (1988). *Attribute-Value Logic and the Theory of Grammar*. CSLI Lecture Notes. Chicago University Press, Chicago.

Johnson, M. (1990). Expressing disjunctive and negative feature constraints with classical first-order logic. In *Proceedings of the 28th ACL*, Pittsburgh, PA, pp. 173–179. ACL.

Johnson, M., Geman, S., Canon, S., Chi, Z., and Riezler, S. (1999). Estimators for stochastic "unification-based" grammars. In *ACL-99*, pp. 535–541.

Jurafsky, D. (1992). *An On-line Computational Model of Human Sentence Interpretation: A Theory of the Representation and Use of Linguistic Knowledge*. Ph.D. thesis, University of California, Berkeley, CA. Available as University of California at Berkeley Computer Science Division Tech. rep. #92/676.

Kaplan, R. M., Riezler, S., King, T. H., Maxwell, J. T., Vasserman, A., and Crouch, R. (2004). Speed and accuracy in shallow and deep stochastic parsing. In *HLT-NAACL-04*.

Kaplan, R. M. (1987). Three seductions of computational psycholinguistics. In Whitelock, P., Wood, M. M., Somers, H. L., Johnson, R., and Bennett, P. (Eds.), *Linguistic Theory and Computer Applications*, pp. 149–188. Academic Press, London.

Kaplan, R. M. and Bresnan, J. (1982). Lexical-functional grammar: A formal system for grammatical representation. In Bresnan, J. (Ed.), *The Mental Representation of Grammatical Relations*, pp. 173–281. MIT Press.

Karttunen, L. and Kay, M. (1985). Structure sharing with binary trees. In *ACL-85*, Chicago, pp. 133–136. ACL.

Kasper, R. T. and Rounds, W. C. (1986). A logical semantics for feature structures. In *ACL-86*, New York, pp. 257–266. ACL.

Kay, M. (1979). Functional grammar. In *BLS-79*, Berkeley, CA, pp. 142–158.

Kay, M. (1984). Functional unification grammar: A formalism for machine translation. In *COLING-84*, Stanford, CA, pp. 75–78.

Kay, M. (1985). Parsing in functional unification grammar. In Dowty, D. R., Karttunen, L., and Zwicky, A. (Eds.), *Natural Language Parsing*, pp. 251–278. Cambridge University Press.

Kay, P. and Fillmore, C. J. (1999). Grammatical constructions and linguistic generalizations: The What's X Doing Y? construction. *Language*, *75*(1), 1–33.

Kiefer, B., Krieger, H. U., Carroll, J., and Malouf, R. (1999a). A bag of useful techniques for efficient and robust parsing. In *ACL-99*, pp. 535–541.

Kiefer, B., Krieger, H.-U., Carroll, J., and Malouf, R. (1999b). A bag of useful techniques for efficient and robust parsing. In *ACL-99*, College Park, MD, pp. 473–480.

King, P. (1989). *A Logical Formalism for Head-Driven Phrase Structure Grammar*. Ph.D. thesis, University of Manchester†. Cited in Carpenter (1992)).

Knight, K. (1989). Unification: A multidisciplinary survey. *ACM Computing Surveys*, *21*(1), 93–124.

Kogure, K. (1990). Strategic lazy incremental copy graph unification. In *COLING-90*, Helsinki, pp. 223–228.

Krieger, H.-U. and Nerbonne, J. (1993). Feature-based inheritance networks for computational lexicons. In Briscoe, T., de Paiva, V., and Copestake, A. (Eds.), *Inheritance, Defaults, and the Lexicon*, pp. 90–136. Cambridge University Press.

Krieger, H.-U. and Schäfer, U. (1994). TDL — A type description language for HPSG. Part 1: Overview. Tech. rep. RR-94-37, DFKI, Saarbrücken.

Lascarides, A. and Copestake, A. (1997). Default representation in constraint-based frameworks. *Computational Linguistics*, *25*(1), 55–106.

Macleod, C., Grishman, R., and Meyers, A. (1998). COMLEX Syntax Reference Manual Version 3.0. Linguistic Data Consortium.

Malouf, R., Carrol, J., , and Copestake, A. (2000). Efficient feature structure operations without compilation. *Natural Language Engineering*, *6*(1).

Meurers, W. D. and Minnen, G. (1997). A computational treatment of lexical rules in HPSG as covariation in lexical entries. *Computational Linguistics*, *23*(4), 543–568.

Miyao, Y. and Tsujii, J. (2005). Probabilistic disambiguation models for wide-coverage hpsg parsing. In *ACL-05*, pp. 83–90.

Moshier, M. A. (1988). *Extensions to Unification Grammar for the Description of Programming Languages*. Ph.D. thesis, University of Michigan, Ann Arbor, MI.

Munteanu, C. and Penn, G. (2004). Optimizing typed feature structure grammar parsing through non-statistical indexing. In *ACL-04*, Barcelona, Spain, pp. 223–230.

Ninomiya, T., Tsuruoka, Y., Miyao, Y., Taura, K., and Tsujii, J. (2006). Fast and scalable hpsg parsing. *Traitement automatique des langues (TAL)*, *46*(2).

Penn, G. and Munteanu, C. (2003). A tabulation-based parsing method that reduces copying. In *Proceedings of the 41st Annual Meeting of the ACL*, Sapporo, Japan.

Pereira, F. C. N. (1985). A structure-sharing representation for unification-based grammar formalisms. In *ACL-85*, Chicago, pp. 137–144.

Pereira, F. C. N. and Shieber, S. M. (1984). The semantics of grammar formalisms seen as computer languages. In *COLING-84*, Stanford, CA, pp. 123–129.

Pereira, F. C. N. and Warren, D. H. D. (1980). Definite clause grammars for language analysis— a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, *13*(3), 231–278.

Pollard, C. and Moshier, M. A. (1990). Unifying partial descriptions of sets. In Hanson, P. P. (Ed.), *Information, Language, and Cognition*, pp. 285–322. University of British Columbia Press, Vancouver.

Pollard, C. and Sag, I. A. (1987). *Information-Based Syntax and Semantics: Volume 1: Fundamentals*. University of Chicago Press, Chicago.

Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, *13*, 81–132.

Riezler, S., King, T. H., Crouch, R., , and Zaenen, A. (2003). Statistical sentence condensation using ambiguity packing and stochastic disambiguation methods for Lexical-Functional Grammar. In *HLT-NAACL-03*, Edmonton, Canada.

Riezler, S., Prescher, D., Kuhn, J., and Johnson, M. (2000). Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and em training. In *ACL-00*, Hong Kong.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., III, J. T. M., and Johnson, M. (2002). Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In *ACL-02*, Philadelphia, PA.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, *12*, 23–41.

Rounds, W. C. and Kasper, R. T. (1986). A complete logical calculus for record structures representing linguistic information. In *Proceedings of the 1st Annual IEEE Symposium on Logic in Computer Science*, pp. 38–43.

Sag, I. A. and Wasow, T. (Eds.). (1999). *Syntactic Theory: A Formal Introduction*. CSLI Publications, Stanford, CA.

Sanfilippo, A. (1993). LKB encoding of lexical knowledge. In Briscoe, T., de Paiva, V., and Copestake, A. (Eds.), *Inheritance, Defaults, and the Lexicon*, pp. 190–222. Cambridge University Press.

Shieber, S. M. (1985). Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *ACL-85*, Chicago, pp. 145–152.

Shieber, S. M. (1986). *An Introduction to Unification-Based Approaches to Grammar*. Center for the Study of Language and Information, Stanford University, Stanford, CA.

Tomabechi, H. (1991). Quasi-destructive graph unification. In *Proceedings of the 29th ACL*, Berkeley, CA, pp. 315–322.

Toutanova, K., Manning, C. D., Flickinger, D., and Oepen, S. (2005). Stochastic HPSG Parse Disambiguation using the Redwoods Corpus. *Research on Language & Computation*, *3*(1), 83–105.

Trubetskoi, N. S. (1939). *Grundzüge der Phonologie*, Vol. 7 of *Travaux du cercle linguistique de Prague*. Available in 1969 English translation by Christiane A. M. Baltaxe as *Principles of Phonology*, University of California Press.

Uszkoreit, H. (1986). Categorial unification grammars. In *COLING-86*, Bonn, pp. 187–194.

Young, M. and Rounds, W. C. (1993). A logical semantics for nonmonotonic sorts. In *Proceedings of the 31st ACL*, Columbus, OH, pp. 209–215. ACL.

17 REPRESENTING MEANING

Ishmael: *Surely all this is not without meaning.*
Herman Melville, *Moby Dick*

MEANING REPRESENTATIONS

MEANING REPRESENTATION LANGUAGES

The approach to semantics introduced here, and elaborated on in the next four chapters, is based on the notion that the meaning of linguistic utterances can be captured in formal structures, which we will call **meaning representations**. Correspondingly, the frameworks that are used to specify the syntax and semantics of these representations will be called **meaning representation languages**. These meaning representations play a role analogous to that of the phonological, morphological, and syntactic representations introduced in earlier chapters.

The need for meaning representations arises when neither the raw linguistic inputs, nor any of the structures derivable from them by any of the transducers we have studied thus far, facilitate the kind of semantic processing that is desired. More specifically, what we need are representations that bridge the gap from linguistic inputs to the non-linguistic knowledge of the world needed to perform tasks involving the meaning of linguistic inputs. To illustrate this notion, consider the following everyday language tasks that require some form of semantic processing of natural language:

- Answering an essay question on an exam;
- Deciding what to order at a restaurant by reading a menu;
- Learning to use a new piece of software by reading the manual;
- Realizing that you've been insulted; and
- Following a recipe.

Simply having access to the phonological, morphological, and syntactic representations that we have discussed thus far will not get us very far on accomplishing any of these tasks. These tasks require access to representations that link the linguistic elements involved in the task to the non-linguistic *knowledge of the world* needed to successfully accomplish them. For example, some of the world knowledge needed to perform the above tasks would include the following:

- Answering and grading essay questions requires background knowledge about the topic of the question, the desired knowledge level of the students, and how such questions are *normally* answered.
- Reading a menu and deciding what to order, giving advice about where to go to dinner, following a recipe, and generating new recipes all require knowledge about food, its preparation, what people like to eat and what restaurants are like.
- Learning to use a piece of software by reading a manual, or giving advice about how to do the same, requires knowledge about current computers, the specific software in question, similar software applications, and knowledge about users in general.

In the representational approach explored here, we take linguistic inputs and construct meaning representations that are made up of the *same kind of stuff* that is used to represent this kind of everyday commonsense knowledge of the world. The process whereby such representations are created and assigned to linguistic inputs is called **semantic analysis**.

SEMANTIC ANALYSIS

To make this notion a bit more concrete, consider Fig. 17.1, which shows sample meaning representations for the sentence *I have a car* using four representative meaning representation languages. The first row illustrates a sentence in **First-Order Logic**, which will be covered in detail in Section 17.4; the graph in the center illustrates a **Semantic Network**, which will be discussed further in Section 17.6; the third row contains a **Conceptual Dependency** diagram, discussed in more detail in Ch. 19, and finally a **Frame-Based** representation, also covered in Section 17.6.

While there are non-trivial differences among these approaches, at an abstract level they all share as a common foundation the notion that a meaning representation consists of structures composed from a set of symbols, or representational vocabulary. When appropriately arranged, these symbol structures are taken to *correspond* to the objects, properties of objects and relations among objects in some state of affairs being represented. In this case, all four representations make use of symbols corresponding to the speaker, a car, and relations denoting the possession of one by the other.

It is important to note that these representations can be viewed from at least two distinct perspectives in all four of these approaches: as representations of the meaning of the particular linguistic input *I have a car*, and as representations of

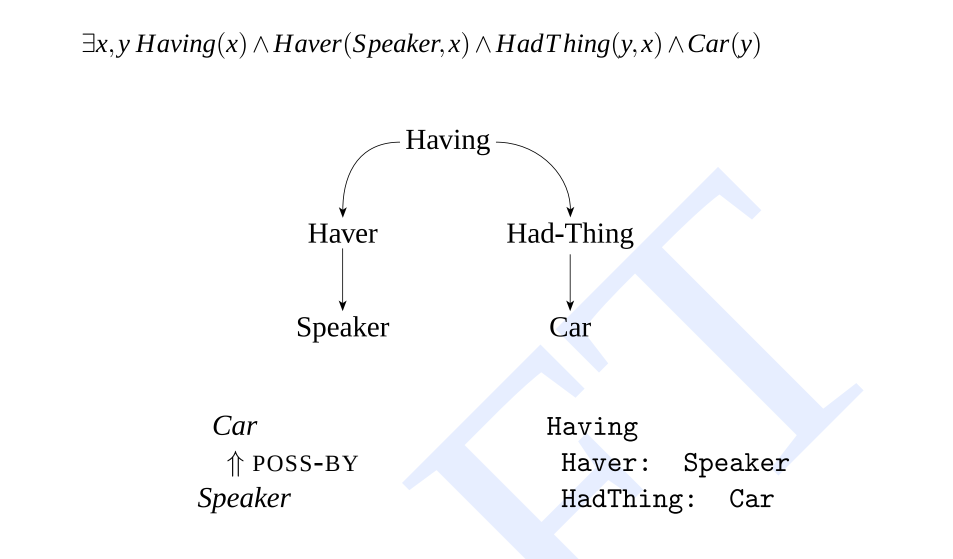


Figure 17.1 A list of symbols, two directed graphs, and a record structure: a sampler of meaning representations for *I have a car.*

the state of affairs in some world. It is this dual perspective that allows these representations to be used to link linguistic inputs to the world and to our knowledge of it.

The structure of this part of the book parallels that of the previous parts. We will alternate discussions of the nature of meaning representations with discussions of the computational processes that can produce them. More specifically, this chapter introduces the basics of what is needed in a meaning representation, while Ch. 18 introduces a number of techniques for assigning meanings to linguistic inputs. Ch. 19 explores a range of complex representational issues related to the meanings of words. Ch. 20 then explores some robust computational methods designed to exploit these lexical representations.

LITERAL MEANING

Since the focus of this chapter is on some of the basic requirements for meaning representations, we will defer a number of extremely important issues to later chapters. In particular, the focus of this chapter is on representing what is sometimes called the **literal meaning** of sentences. By this, we have in mind representations that are closely tied to the conventional meanings of the words that are used to create them, and that do not reflect much of the context in which they occur. The shortcomings of such representations with respect to phenomena such as idioms and metaphor will be discussed in the next two chapters, while the role of context in ascertaining the deeper meaning of sentences will be covered in Chs. 20 and 23.

There are four major parts to this chapter. Section 17.1 explores some of the key computational requirements for what we need in a meaning representation language. Section 17.2 then discusses some of the ways that languages are structured to convey meaning. Section 17.3 describes how we can more formally specify the meanings of our meaning representations. Section 17.4 then provides an introduction to First Order Logic, which has historically been the primary technique used to investigate issues in natural language semantics.

17.1 Computational Desiderata for Representations

We begin by considering the issue of why meaning representations are needed and what they should do for us. To focus this discussion, we will consider in more detail the task of giving advice about restaurants to tourists. In this discussion, we will assume that we have a computer system that accepts spoken language queries from tourists and construct appropriate responses by using a knowledge base of relevant domain knowledge. A series of examples will serve to introduce some of the basic requirements that a meaning representation must fulfill, and some of the complications that inevitably arise in the process of designing such meaning representations. In each of these examples, we will examine the role that the representation of the meaning of the request must play in the process of satisfying it.

17.1.1 Verifiability

Let us begin by considering the following simple question:

(17.1) Does Maharani serve vegetarian food?

This example illustrates the most basic requirement for a meaning representation: it must be possible to use the representation to determine the relationship between the meaning of a sentence and the world as we know it. In other words, we need to be able to determine the truth of our representations. The most straightforward way to implement this notion is make it possible for a system to compare, or *match*, the representation of the meaning of an input against the representations in its **knowledge base**, its store of information about its world.

KNOWLEDGE BASE

In this example, let us assume that the meaning of this question contains, as a component, the meaning underlying the proposition *Maharani serves vegetarian food*. For now, we will simply gloss this representation as:

$$Serves(Maharani, VegetarianFood)$$

It is this representation of the input that will be matched against the knowledge base of facts about a set of restaurants. If the system finds a representation matching the input proposition in its knowledge base, it can return an affirmative answer. Otherwise, it must either say *No*, if its knowledge of local restaurants is complete, or say that it does not know if there is reason to believe that its knowledge is incomplete.

VERIFIABILITY

This notion is known as **verifiability**, and concerns a system's ability to compare the state of affairs described by a representation to the state of affairs in some world as modeled in a knowledge base.

17.1.2 Unambiguous Representations

The domain of semantics, like all the other domains we have studied, is subject to ambiguity. Specifically, single linguistic inputs can legitimately have different meaning representations assigned to them based on the circumstances in which they occur.

Consider the following example from the BERP corpus:

(17.2) I wanna eat someplace that's close to ICSI.

Given the allowable argument structures for the verb *eat*, this sentence can either mean that the speaker wants to eat *at* some nearby location, or under a Godzilla as speaker interpretation, the speaker may want to devour some nearby location. The answer generated by the system for this request will depend on which interpretation is chosen as the correct one.

Since ambiguities such as this abound in all genres of all languages, some means of determining that certain interpretations are preferable (or alternatively less preferable) than others is needed. The various linguistic phenomena that give rise to such ambiguities, and the techniques that can be employed to deal with them, will be discussed in detail in the next four chapters.

Our concern in this chapter, however, is with the status of our meaning representations with respect to ambiguity, and not with the means by which we might arrive at correct interpretations. Since we reason about, and act upon, the semantic content of linguistic inputs, the final representation of an input's meaning should be free from any ambiguity. Therefore, regardless of any ambiguity in the raw input, it is critical that a meaning representation language support representations that have a single unambiguous interpretation[1].

VAGUENESS

A concept closely related to ambiguity is **vagueness**. Like ambiguity, vague-

[1] This does not preclude the use of intermediate semantic representations that maintain some level of ambiguity on the way to a single unambiguous form. Examples of such representations will be discussed in Ch. 18.

ness can make it difficult to determine what to do with a particular input based on its meaning representation. Vagueness, however, does not give rise to multiple representations.

Consider the following request as an example:

(17.3) I want to eat Italian food.

While the use of the phrase *Italian food* may provide enough information for a restaurant advisor to provide reasonable recommendations, it is nevertheless quite *vague* as to what the user really wants to eat. Therefore, a vague representation of the meaning of this phrase may be appropriate for some purposes, while a more specific representation may be needed for other purposes. It will, therefore, be advantageous for a meaning representation language to support representations that maintain a certain level of vagueness. Note that it is not always easy to distinguish ambiguity from vagueness. Zwicky and Sadock (1975) provide a useful set of tests that can be used as diagnostics.

17.1.3 Canonical Form

The notion that single sentences can be assigned multiple meanings leads to the related phenomenon of distinct inputs that should be assigned the same meaning representation. Consider the following alternative ways of expressing example (17.1):

(17.4) Does Maharani have vegetarian dishes?

(17.5) Do they have vegetarian food at Maharani?

(17.6) Are vegetarian dishes served at Maharani?

(17.7) Does Maharani serve vegetarian fare?

Given that these alternatives use different words and have widely varying syntactic analyses, it would not be unreasonable to expect them to have substantially different meaning representations. Such a situation would, however, have undesirable consequences for our matching approach to determining the truth of our representations. If the system's knowledge base contains only a single representation of the fact in question, then the representations underlying all but one of our alternatives will fail to produce a match. We could, of course, store all possible alternative representations of the same fact in the knowledge base, but this would lead to an enormous number of problems related to keeping such a knowledge base consistent.

The way out of this dilemma is motivated by the fact that since the answers given for each of these alternatives should be the same in all situations, we might say that they all mean the same thing, at least for the purposes of giving restaurant recommendations. In other words, at least in this domain, we can legitimately consider assigning the same meaning representation to the propositions underlying each of these requests. Taking such an approach would guarantee that our matching scheme for answering Yes-No questions will still work.

CANONICAL FORM

The notion that inputs that mean the same thing should have the same meaning representation is known as the doctrine of **canonical form**. This approach greatly simplifies various reasoning tasks since systems need only deal with a single meaning representation for a potentially wide range of expressions.

Canonical form does, of course, complicate the task of semantic analysis. To see this, note that the alternatives given above use completely different words and syntax to refer to vegetarian fare and to what restaurants do with it. More specifically, to assign the same representation to all of these requests our system will have to conclude that *vegetarian fare*, *vegetarian dishes* and *vegetarian food* refer to the same thing in this context, that the use here of *having* and *serving* are similarly equivalent, and that the different syntactic parses underlying these requests are all compatible with the same meaning representation.

Being able to assign the same representation to such diverse inputs is a tall order. Fortunately there are some systematic meaning relationships among word senses and among grammatical constructions that can be exploited to make this task tractable. Consider the issue of the meanings of the words *food*, *dish* and *fare* in these examples. A little introspection, or a glance at a dictionary, reveals that these words have a fair number of distinct uses. Fortunately, it also reveals that there is at least one sense that is shared among them all. If a system has the ability to choose that shared sense, then an identical meaning representation can be assigned to the phrases containing these words.

WORD SENSES

WORD SENSE DISAMBIGUATION

In general, we say that these words all have various **word senses** and that some of the senses are synonymous with one another. The process of choosing the right sense in context is called **word sense disambiguation**, or word sense tagging by analogy to part-of-speech tagging. The topics of synonymy, sense tagging, and a host of other topics related to word meanings will be covered in Chs. 17 and 18. Suffice it to say here that the fact that inputs may use different words does not preclude the assignment of identical meanings to them.

Just as there are systematic relationships among the meanings of different words, there are similar relationships related to the role that syntactic analyses play in assigning meanings to sentences. Specifically, alternative syntactic analyses often have meanings that are, if not identical, at least systematically related to one another. Consider the following pair of examples:

(17.8) Maharani serves vegetarian dishes.

(17.9) Vegetarian dishes are served by Maharani.

Despite the different placement of the arguments to *serve* in these examples, we can still assign *Maharani* and *vegetarian dishes* to the same roles in both of these examples because of our knowledge of the relationship between active and passive sentence constructions. In particular, we can use knowledge of where grammatical subjects and direct objects appear in these constructions to assign *Maharani*, to the role of the server, and *vegetarian dishes* to the role of thing being served in both of these examples, despite the fact that they appear in different surface locations. The precise role of the grammar in the construction of meaning representations will be covered in Ch. 18.

17.1.4 Inference and Variables

Continuing with the topic of the computational purposes that meaning representations should serve, we should consider more complex requests such as the following:

(17.10) Can vegetarians eat at Maharani?

Here, it would be a mistake to invoke canonical form to force our system to assign the same representation to this request as for the previous examples. The fact that this request results in the same answer as the others arises not because they mean the same thing, but because there is a commonsense connection between what vegetarians eat and what vegetarian restaurants serve. This is a fact about the world and not a fact about any particular kind of linguistic regularity. This implies that no approach based on canonical form and simple matching will give us an appropriate answer to this request. What is needed is a systematic way to connect the meaning representation of this request with the facts about the world as they are represented in a knowledge base.

INFERENCE

We will use the term **inference** to refer generically to a system's ability to draw valid conclusions based on the meaning representation of inputs and its store of background knowledge. It must be possible for the system to draw conclusions about the truth of propositions that are not explicitly represented in the knowledge base, but are nevertheless logically derivable from the propositions that are present.

Now consider the following somewhat more complex request:

(17.11) I'd like to find a restaurant where I can get vegetarian food.

Unlike our previous examples, this request does not make reference to any particular restaurant. The user is stating that they would like information about an unknown and unnamed entity that is a restaurant that serves vegetarian food. Since this request does not mention any particular restaurant, the kind of simple matching-based approach we have been advocating is not going to work. Rather, answering this request requires a more complex kind of matching that involves the use of variables. We can gloss a representation containing such variables as follows:

$$Serves(x, VegetarianFood)$$

Matching such a proposition succeeds only if the variable *x* can be replaced by some known object in the knowledge base in such a way that the entire proposition will then match. The concept that is substituted for the variable can then be used to fulfill the user's request. Of course, this simple example only hints at the issues involved in the use of such variables. Suffice it to say that linguistic inputs contain many instances of all kinds of indefinite references and it is therefore critical for any meaning representation language to be able to handle this kind of expression.

17.1.5 Expressiveness

Finally, to be useful a meaning representation scheme must be expressive enough to handle an extremely wide range of subject matter. The ideal situation, of course, would be to have a single meaning representation language that could adequately represent the meaning of any sensible natural language utterance. Although this is probably too much to expect from any single representational system, Section 17.4 will show that First-Order Logic is expressive enough to handle quite a lot of what needs to be represented.

17.2 Meaning Structure of Language

MEANING STRUCTURE OF LANGUAGE

The previous section focused on some of the purposes that meaning representations must serve, without saying much about what we will call the **meaning structure of language**. By this, we have in mind the various methods by which human languages convey meaning. These include a variety of conventional form-meaning associations, word-order regularities, tense systems, conjunctions and quantifiers, and a fundamental predicate-argument structure. The remainder of this section focuses exclusively on this last notion of a predicate-argument structure, which is the mechanism that has had the greatest practical influence on the nature of meaning representation languages. The remaining topics will be addressed in Ch. 18 where the primary focus will be on how they contribute to how meaning representations are assembled, rather than on the nature of the representations.

17.2.1 Predicate-Argument Structure

Human languages have a form of predicate-argument arrangement at the core of their semantic structure. To a first approximation, this predicate-argument structure asserts that specific relationships, or dependencies, hold among the various concepts underlying the constituent words and phrases that make up sentences. It is this underlying structure that permits the creation of a single composite meaning representation from the meanings of the various parts of an input. One of the most important jobs of a grammar is to help organize this predicate-argument structure. Correspondingly, it is critical that our meaning representation languages support the predicate-argument structures presented to us by language.

We have already seen the beginnings of this concept in our discussion of verb complements in Chs. 11 and 15. There we saw that verbs dictate specific constraints on the number, grammatical category, and location of the phrases that are expected to accompany them in syntactic structures. To briefly review this idea, consider the following examples:

(17.12) I want Italian food.

(17.13) I want to spend less than five dollars.

(17.14) I want it to be close by here.

These examples can be classified as having one of the following three syntactic argument frames:

NP want NP

NP want Inf-VP

NP want NP Inf-VP

These syntactic frames specify the number, position and syntactic category of the arguments that are expected to accompany a verb. For example, the frame for the variety of *want* that appears in example (17.12) specifies the following facts:

- There are two arguments to this predicate.
- Both arguments must be *NP*s.
- The first argument is pre-verbal and plays the role of the subject.
- The second argument is post-verbal and plays the role of the direct object.

As we have shown in previous chapters, this kind of information is quite valuable in capturing a variety of important facts about syntax. By analyzing easily observable semantic information associated with these frames, we can also gain considerable insight into our meaning representations. We will begin by considering two extensions of these frames into the semantic realm: semantic roles and semantic restrictions on these roles.

The notion of a semantic role can be understood by looking at the similarities among the arguments in examples (17.12) through (17.14). In each of these cases, the pre-verbal argument always plays the role of the entity doing the wanting, while the post-verbal argument plays the role of the concept that is *wanted*. By noticing these regularities and labeling them accordingly, we can associate the surface arguments of a verb with a set of discrete roles in its underlying semantics. More generally, we can say that verb subcategorization frames allow the **linking** of

LINKING

arguments in the surface structure with the semantic roles these arguments play in the underlying semantic representation of an input. The study of roles associated with specific verbs and across classes of verbs is usually referred to as **thematic role** or **case role** analysis and will be studied more fully in Ch. 19.

THEMATIC ROLE

CASE ROLE

The notion of semantic restrictions arises directly from these semantic roles. Returning to examples 17.12 through 17.14, we can see that it is not merely the case that each initial noun phrase argument will be the *wanter* but that only certain kinds, or *categories*, of concepts can play the role of wanter in any straightforward manner. Specifically, *want* restricts the constituents appearing as the first argument to those whose underlying concepts can actually partake in a wanting. Traditionally, this notion is referred to as a **selectional restriction**. Through the use of these selectional restrictions, verbs can specify semantic restrictions on their arguments.

SELECTIONAL RESTRICTION

Before leaving this topic, we should note that verbs are by no means the only objects in a grammar that can carry a predicate-argument structure. Consider the following phrases from the BERP corpus:

(17.15) an Italian restaurant under fifteen dollars

In this example, the meaning representation associated with the preposition *under* can be seen as having something like the following structure:

$$Under(ItalianRestaurant,\$15)$$

In other words, prepositions can be characterized as two-argument predicates where the first argument is an object that is being placed in some relation to the second argument.

Another non-verb based predicate-argument structure is illustrated in the following example:

(17.16) Make a reservation for this evening for a table for two persons at 8.

Here, the predicate-argument structure is based on the concept underlying the noun *reservation*, rather than *make*, the main verb in the phrase. This example gives rise to a four argument predicate structure like the following:

$$Reservation(Hearer,Today,8PM,2)$$

This discussion makes it clear that any useful meaning representation language must be organized in a way that supports the specification of semantic predicate-argument structures. Specifically, it must include support for the kind of semantic information that languages present:

- variable arity predicate-argument structures
- the semantic labeling of arguments to predicates
- the statement of semantic constraints on the fillers of argument roles

17.3 Model-Theoretic Semantics

The last two sections focused on various desiderata for meaning representations and on some of the ways in which natural languages convey meaning. We haven't said much formally about what it is about meaning representation languages that allows them to do all the things we want them to. In particular, we might like to have some kind of guarantee that these representations can do the work that we require of them: bridge the gap from merely formal representations to representations that tell us something about some state of affairs in the world.

MODEL

To see how we might provide such a guarantee, let's start with the basic notions shared by most meaning representation schemes. What they all have in common is the ability to represent **objects**, **properties of objects** and **relations among objects**. This point of view can be formalized via the notion of a **model**. The basic idea is that a model is a formal construct that stands for the particular state of affairs in the world that we're trying to represent. Expressions in a meaning representation language will then be mapped in a systematic way to the elements of the model. If the model accurately captures the facts we're interested in concerning some state of affairs in the world, then a systematic mapping between the meaning representation and model provides the necessary bridge between the meaning representation and world being considered. As we'll see, models provide a surprisingly simple and powerful way to ground the expressions in meaning representation languages.

NON-LOGICAL VOCABULARY

LOGICAL VOCABULARY

Before we start let's introduce some terminology. The vocabulary of a meaning representation consists of two parts: the non-logical vocabulary and the logical vocabulary. The **non-logical vocabulary** consists of the open-ended set of names for the objects, properties and relations that make up the world we're trying to represent. These appear in various schemes as predicates, nodes, labels on links, or labels in slots in frames, The **logical vocabulary** consists of the closed set of symbols, operators, quantifiers, links, etc. that provide the formal means for composing expressions in a given meaning representation language.

DOMAIN

We'll start by requiring that each element of the non-logical vocabulary of a meaning representation have a **denotation** in the model. By denotation, we simply mean that every element of the non-logical vocabulary corresponds to a fixed well-defined part of the model. Let's start with objects, the most basic notion in most representational schemes. The **domain** of a model is simply the set of objects that are part of the application, or state of affairs, being represented. Each distinct concept, category or individual in an application denotes a unique element in the domain. A domain is therefore formally a set. Note that it isn't the case that every element of the domain have a corresponding concept in our meaning representation; it's perfectly acceptable to have domain elements that aren't mentioned or conceived of in the meaning representation. Nor do we require that elements of the domain have a single denoting concept in the meaning representation; a given element in the domain might have several distinct representations denoting it, such as *Mary*, *WifeOf(Abe)*, or *MotherOf(Robert)*.

We can capture properties of objects in a model by denoting those domain elements that have the property in question; that is, properties denote sets. Similarly, relations among objects denote sets of ordered lists, or tuples, of domain elements that take part in the corresponding relations. This approach to properties and relations is thus an **extensional** one; the denotation of properties like *red* is the set of things we think are red, the denotation of a relation like *Married* is simply the set of pairs of domain elements that are married. To summarize:

- Objects denote *elements* of the domain
- Properties denote *sets of elements* of the domain
- Relations denote *sets of tuples of elements* of the domain

There is one additional element that we need to make this scheme work. We need a mapping that systematically gets us from our meaning representation to the corresponding denotations. More formally, we need a function that maps from the non-logical vocabulary of our meaning representation to the proper denotations in the model. We'll call such a mapping an **interpretation**.

INTERPRETATION

To make these notions more concrete, let's return to the realm of restaurants we introduced in Ch. 4. Assume that our application concerns a particular set of restaurant patrons and restaurants, various facts about the likes and dislikes of the patrons, and facts about the restaurants such as their cuisine, typical cost, and noise level.

To begin populating our domain, $\mathscr{D}$, let's assume that in the current state of affairs we're dealing with four patrons designated by the non-logical symbols *Matthew, Franco, Katie* and *Caroline*. These four symbols will denote 4 unique domain elements. We'll use the constants a, b, c and, d to stand for these domain elements. Note that we're deliberately using meaningless, non-mnemonic names for our domain elements to emphasize the fact that whatever it is that we know about these entities has to come from the formal properties of the model and not from the names of the symbols. Continuing, let's assume that our application includes three restaurants, designated as *Frasca*, *Med* and *Rio* in our meaning representation, that denote the domain elements e, f and g. Finally, let's assume that we're dealing with the three cuisines *Italian*, *Mexican*, and *Eclectic*, denoting i, j, and k in our model.

Having populated the domain, let's move on to the properties and relations we believe to be true in this particular state of affairs. Let's assume that in our application we need to represent some properties of restaurants such as the fact that

Domain	$\mathscr{D} = \{a,b,c,d,e,f,g,h,i,j\}$
Matthew, Franco, Katie and Caroline	a,b,c,d
Frasca, Med, Rio	e,f,g
ItalianCuisine, MexicanCuisne, EclecticCuisine	h,i,j
Noisy	$Noisy = \{e,f,g\}$
Frasca, Med and Rio are noisy	
Likes	$Likes = \{\langle a,f\rangle, \langle c,f\rangle, \langle c,g\rangle, \langle b,e\rangle, \langle d,f,\rangle, \langle d,g\rangle\}$
Matthew likes the Med	
Katie likes the Med and Rio	
Franco likes Frasca	
Caroline likes the Med and Rio	
Serves	$Serves = \{\langle e,j\rangle, \langle f,i\rangle, \langle e,h\rangle\}$
Med serves eclectic	
Rio serves Mexican	
Frasca serves Italian	

Figure 17.2 A model of the restaurant world.

some are noisy or expensive. Properties like *Noisy* denote the subset of restaurants from our domain that are known to be noisy. Two-place relational notions, such as which restaurants individual patrons *Like*, denote ordered pairs, or tuples, of the objects from the domain. Similarly, since we decided to represent cuisines as objects in our model, we can also capture which restaurants *Serve* which cuisines as a set of tuples. One particular state of affairs using this scheme is given in Fig. 17.2.

Given this simple scheme, we can ground the meaning of pretty much any of the representations shown earlier in Fig. **??** by simply consulting the appropriate denotations in the corresponding model. A representation claiming, for example, that *Matthew likes the Rio* , or that the *The Med serves Italian* can be evaluated by mapping the objects in the meaning representations to their corresponding domain elements, and any links, predicates, or slots in the meaning representation to the appropriate relations in the model. More concretely, a representation asserting that *Matthew likes Frasca* can be verified by first using our interpretation function to map the symbol *Matthew* to its denotation a, *Frasca* to e, and the *Likes* relation to the appropriate set of tuples. We then simply check that set of tuples for the presence of the tuple $\langle a,e\rangle$. If, as it is in this case, the tuple is present in the model then we can conclude that *Matthew likes Frasca* is true, and if it isn't we can't.

This is all pretty much straightforward, we're simply using sets and operations on sets to ground the expressions in our meaning representations. Of course, the more interesting part comes when we consider more complex examples such as the following:

(17.17) Katie likes the Rio and Matthew likes the Med.

(17.18) Katie and Caroline like the same restaurants.

(17.19) Franco likes noisy, expensive restaurants.

(17.20) Not everybody likes Frasca.

Clearly, our simple scheme for grounding the meaning of representations is not adequate for examples such as these. Plausible meaning representations for these examples will not map directly to individual entities, properties or relations. Instead, they involve complications such as conjunctions, equality, quantified variables and negations. To assess whether or not these statements are consistent with our model we'll have to tear them apart, assess the parts and then determine the meaning of the whole from the meaning of the parts according to the details of how the whole is assembled.

Consider the first example given above. A typical meaning representation for examples like this will include two distinct propositions expressing the individual patron's preferences, conjoined with some kind of implicit or explicit conjunction operator. Obviously, our model doesn't have a relation that encodes the pairwise preferences for all of the patrons and restaurants in our model, nor does it need to. We know from our model that *Matthew likes the Med* and separately that *Katie likes the Rio* (that is, we know that the tuples $\langle a, f \rangle$ and $\langle c, g \rangle$ are members of the set denoted by the *Likes* relation.) All we really need to know is how to deal with the semantics of the conjunction operator. If we assume the simplest possible semantics for the English word *and*, the whole statement is true if it is the case each of the components is true in our model. In this case, both components are true since the appropriate tuples are present and therefore the sentence as a whole is true.

TRUTH-CONDITIONAL SEMANTICS

What we've done implicitly in this example is to provide what is called a **truth-conditional semantics** for the assumed conjunction operator in some meaning representation. That is, we've provided a method for determining the truth of a complex expression from the meanings of the parts (by consulting a model) and the meaning of an operator by essentially consulting a truth-table. The various representations that populate Fig. 17.1 are truth-conditional to the extent that they give a formal specification as to how we can assess the meaning of complex sentences from the meaning of their parts. In particular, we'll need to know the semantics of the entire **logical vocabulary** of the meaning representation scheme being used.

Note that although the details of how this happens is dependent on details of the particular meaning representation being used, it should be clear that assessing the truth conditions of examples like these involves nothing beyond the simple set operations we've been discussing. We'll return to these issues in the next section where we discuss them in the context of the semantics of First Order Logic.

17.4 First-Order Logic

First-Order Logic (FOL) is a flexible, well-understood, and computationally tractable approach to the representation of knowledge that satisfies many of the desiderata given in Sections 17.1 and 17.2 for a meaning representation language. Specifically, it provides a sound computational basis for the verifiability, inference, and expressiveness requirements, and as we'll see a sound model-theoretic semantics.

However, the most attractive feature of FOL is the fact that it makes very few specific commitments as to how things ought to be represented. As we will see, the specific commitments it does make are ones that are fairly easy to live with and are shared by many of the schemes mentioned earlier; the represented world consists of objects, properties of objects, and relations among objects.

The remainder of this section first provides an introduction to the basic syntax and semantics of FOPC, and then describes the application of FOPC to a number of linguistically relevant topics. Section 17.7 then discusses the connections between FOPC and some of the other representations shown earlier in Figure 17.1.

17.4.1 Elements of First Order Logic

We will explore FOL in a bottom-up fashion by first examining its various atomic elements and then showing how they can be composed to create larger meaning representations. Fig. 17.3, which provides a complete context-free grammar for the particular syntax of FOL that we will be using, will be our roadmap for this section.

TERM

Let's begin by examining the notion of a **Term**, the FOL device for representing objects. As can be seen from Figure 17.3, FOL provides three ways to represent these basic building blocks: constants, functions, and variables. Each of these devices can be thought of as a way of naming, or pointing to, an object in the world under consideration.

CONSTANTS

Constants in FOL refer to specific objects in the world being described. Such constants are conventionally depicted as either single capitalized letters such as *A* and *B* or single capitalized words that are often reminiscent of proper nouns such as *Maharani* and *Harry*. Like programming language constants, FOL constants refer

$$
\begin{array}{rcl}
\mathit{Formula} & \rightarrow & \mathit{AtomicFormula} \\
 & | & \mathit{Formula\ Connective\ Formula} \\
 & | & \mathit{Quantifier\ Variable}, \ldots\ \mathit{Formula} \\
 & | & \neg\, \mathit{Formula} \\
 & | & (\mathit{Formula}) \\
 & & \\
\mathit{AtomicFormula} & \rightarrow & \mathit{Predicate}(\mathit{Term}, \ldots) \\
 & & \\
\mathit{Term} & \rightarrow & \mathit{Function}(\mathit{Term}, \ldots) \\
 & | & \mathit{Constant} \\
 & | & \mathit{Variable} \\
 & & \\
\mathit{Connective} & \rightarrow & \wedge \mid \vee \mid\ \Rightarrow \\
\mathit{Quantifier} & \rightarrow & \forall \mid \exists \\
\mathit{Constant} & \rightarrow & A \mid \mathit{VegetarianFood} \mid \mathit{Maharani} \cdots \\
\mathit{Variable} & \rightarrow & x \mid y \mid \cdots \\
\mathit{Predicate} & \rightarrow & \mathit{Serves} \mid \mathit{Near} \mid \cdots \\
\mathit{Function} & \rightarrow & \mathit{LocationOf} \mid \mathit{CuisineOf} \mid \cdots
\end{array}
$$

Figure 17.3 A context-free grammar specification of the syntax of First Order Predicate Calculus representations. Adapted from Russell and Norvig (1995).

to exactly one object. Objects can, however, have multiple constants that refer to them.

FUNCTIONS

Functions in FOPC correspond to concepts that are often expressed in English as genitives such as *Frasca's location*. A FOL translation of such an expression might look like the following.

$$LocationOf(Frasca)$$

FOPC functions are syntactically the same as single argument predicates. It is important to remember, however, that while they have the appearance of predicates they are in fact *Terms* in that they refer to unique objects. Functions provide a convenient way to refer to specific objects without having to associate a named constant with them. This is particularly convenient in cases where many named objects, like restaurants, will have a unique concept such as a location associated with them.

VARIABLE

The notion of a **variable** is our final FOPC mechanism for referring to objects. Variables, which are normally depicted as single lower-case letters, give us the ability to make assertions and draw inferences about objects without having to make reference to any particular named object. This ability to make statements about anonymous objects comes in two flavors: making statements about a particular unknown object and making statements about all the objects in some arbitrary world of objects. We will return to the topic of variables after we have presented quantifiers, the elements of FOPC that will make them useful.

Now that we have the means to refer to objects, we can move on to the FOPC mechanisms that are used to state relations that hold among objects. As one might guess from its name, FOPC is organized around the notion of the predicate. Predicates are symbols that refer to, or name, the relations that hold among some fixed number of objects in a given domain. Returning to the example introduced informally in Section 17.1, a reasonable FOPC representation for *Maharani serves vegetarian food* might look like the following formula:

$$Serves(Maharani, VegetarianFood)$$

This FOPC sentence asserts that *Serves*, a two-place predicate, holds between the objects denoted by the constants *Maharani* and *VegetarianFood*.

A somewhat different use of predicates is illustrated by the following typical representation for a sentence like *Maharani is a restaurant*:

$$Restaurant(Maharani)$$

This is an example of a one-place predicate that is used, not to relate multiple objects, but rather to assert a property of a single object. In this case, it encodes the category membership of *Maharani*. We should note that while this is a commonplace way to deal with categories it is probably not the most useful. Section 17.5 will return to the topic of the representation of categories.

With the ability to refer to objects, to assert facts about objects, and to relate objects to one another, we have the ability to create rudimentary composite representations. These representations correspond to the atomic formula level in Figure 17.3. Recall that this ability to create composite meaning representations was one of the core components of the meaning structure of language described in Section 17.2.

LOGICAL CONNECTIVES

This ability to compose complex representations is not limited to the use of single predicates. Larger composite representations can also be put together through the use of **logical connectives**. As can be seen from Figure 17.3, logical connectives give us the ability to create larger representations by conjoining logical formulas using one of three operators. Consider, for example, the following BERP sentence and one possible representation for it:

(17.21) I only have five dollars and I don't have a lot of time.

$$Have(Speaker, FiveDollars) \wedge \neg Have(Speaker, LotOfTime)$$

The semantic representation for this example is built up in a straightforward way from semantics of the individual clauses through the use of the $\wedge$ and $\neg$ operators. Note that the recursive nature of the grammar in Figure 17.3 allows an infinite number of logical formulas to be created through the use of these connectives. Thus as with syntax, we have the ability to create an infinite number of representations using a finite device.

17.4.2 The Semantics of First Order Logic

The various objects, properties, and relations represented in a FOPC knowledge base acquire their meanings by virtue of their correspondence to objects, properties, and relations out in the external world being modeled by the knowledge base. FOPC sentences can, therefore, be assigned a value of *True* or *False* based on whether the propositions they encode are in accord with the world or not.

Consider the following example:

(17.22) Ay Caramba is near ICSI.

Capturing the meaning of this example in FOPC involves identifying the *Terms* and *Predicates* that correspond to the various grammatical elements in the sentence, and creating logical formulas that capture the relations implied by the words and syntax of the sentence. For this example, such an effort might yield something like the following:

$$Near(LocationOf(AyCaramba), LocationOf(ICSI))$$

The meaning of this logical formula then arises from the relationship between the terms $LocationOf(AyCaramba)$, $LocationOf(ICSI)$, the predicate *Near*, and the objects and relation they correspond to in the world being modeled. Specifically, this sentence can be assigned a value of *True* or *False* based on whether or not the real Ay Caramba is actually close to ICSI or not. Of course, since our computers rarely have direct access to the outside world we have to rely on some other means to determine the truth of formulas like this one.

For our current purposes, we will adopt what is known as a database semantics for determining the truth of our logical formulas. Operationally, atomic formulas are taken to be true if they are literally present in the knowledge base or if they can be inferred from other formula that are in the knowledge base. The interpretations of formulas involving logical connectives is based on the meaning of the components in the formulas combined with the meanings of the connectives they contain. Fig. 17.4 gives interpretations for each of the logical operators shown in Figure 17.3.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$
False	*False*	*True*	*False*	*False*	*True*
False	*True*	*True*	*False*	*True*	*True*
True	*False*	*False*	*False*	*True*	*False*
True	*True*	*False*	*True*	*True*	*True*

Figure 17.4 Truth table giving the semantics of the various logical connectives.

The semantics of the $\wedge$ (and), and $\neg$ (not) operators are fairly straightforward, and are correlated with at least some of the senses of their corresponding English terms. However, it is worth pointing out that the $\vee$ (or) operator is not disjunctive in the same way that the corresponding English word is, and that the $\Rightarrow$ (implies) operator is only loosely based on any commonsense notions of implication or causation. As we will see in more detail in Section 17.5, in most cases it is safest to rely directly on the entries in the truth table, rather than on intuitions arising from the names of the operators.

17.4.3 Variables and Quantifiers

QUANTIFIERS

We now have all the machinery necessary to return to our earlier discussion of variables. As noted above, variables are used in two ways in FOPC: to refer to particular anonymous objects and to refer generically to all objects in a collection. These two uses are made possible through the use of operators known as **quantifiers**. The two operators that are basic to FOPC are the existential quantifier, which is denoted $\exists$, and is pronounced as “there exists”, and the universal quantifier, which is denoted $\forall$, and is pronounced as “for all”.

The need for an existentially quantified variable is often signaled by the presence of an indefinite noun phrase in English. Consider the following example:

(17.23) a restaurant that serves Mexican food near ICSI.

Here reference is being made to an anonymous object of a specified category with particular properties. The following would be a reasonable representation of the meaning of such a phrase:

$$\begin{aligned}&\exists x Restaurant(x)\\&\qquad\qquad \wedge Serves(x, MexicanFood)\\&\qquad\qquad \wedge Near((LocationOf(x), LocationOf(ICSI))\end{aligned}$$

The existential quantifier at the head of this sentence instructs us on how to interpret the variable x in the context of this sentence. Informally, it says that for this sentence to be true there must be at least one object such that if we were to substitute it for the variable x, the resulting sentence would be true. For example, if *AyCaramba* is a Mexican restaurant near ICSI, then substituting *AyCaramba* for *x* results in the following logical formula:

$$\begin{aligned}&Restaurant(AyCaramba)\\&\quad\wedge Serves(AyCaramba, MexicanFood)\\&\quad\wedge Near((LocationOf(AyCaramba), LocationOf(ICSI))\end{aligned}$$

Based on the semantics of the $\wedge$ operator, this sentence will be true if all of its three component atomic formulas are true. These in turn will be true if they are either present in the system's knowledge base or can be inferred from other facts in the knowledge base.

The use of the universal quantifier also has an interpretation based on substitution of known objects for variables. The substitution semantics for the universal quantifier takes the expression *for all* quite literally; the $\forall$ operator states that for the logical formula in question to be true the substitution of *any* object in the knowledge base for the universally quantified variable should result in a true formula. This is in marked contrast to the $\exists$ operator which only insists on a single valid substitution for the sentence to be true.

Consider the following example:

(17.24) All vegetarian restaurants serve vegetarian food.

A reasonable representation for this sentence would be something like the following:

$$\forall x VegetarianRestaurant(x) \Rightarrow Serves(x, VegetarianFood)$$

For this sentence to be true, it must be the case that every substitution of a known object for *x* must result in a sentence that is true. We can divide up the set of all possible substitutions into the set of objects consisting of vegetarian restaurants and the set consisting of everything else. Let us first consider the case where the substituted object actually is a vegetarian restaurant; one such substitution would result in the following sentence:

$$\begin{aligned}&VegetarianRestaurant(Maharani)\\&\qquad\qquad\Rightarrow Serves(Maharani, VegetarianFood)\end{aligned}$$

If we assume that we know that the consequent clause,

$$Serves(Maharani, VegetarianFood)$$

is true then this sentence as a whole must be true. Both the antecedent and the consequent have the value *True* and, therefore, according to the first two rows of Fig. 17.4 the sentence itself can have the value *True*. This result will, of course, be the same for all possible substitutions of *Terms* representing vegetarian restaurants for *x*.

Remember, however, that for this sentence to be true it must be true for all possible substitutions. What happens when we consider a substitution from the set of objects that are not vegetarian restaurants? Consider the substitution of a non-vegetarian restaurant such as *Ay Caramba's* for the variable *x*:

$$
\begin{aligned}
&VegetarianRestaurant(AyCaramba)\\
&\qquad\qquad\qquad \Rightarrow Serves(AyCaramba, VegetarianFood)
\end{aligned}
$$

Since the antecedent of the implication is *False*, we can determine from Fig. 17.4 that the sentence is always *True*, again satisfying the $\forall$ constraint.

Note, that it may still be the case that *Ay Caramba* serves vegetarian food without actually being a vegetarian restaurant. Note also, that despite our choice of examples, there are no implied categorical restrictions on the objects that can be substituted for *x* by this kind of reasoning. In other words, there is no restriction of *x* to restaurants or concepts related to them. Consider the following substitution:

$$
\begin{aligned}
&VegetarianRestaurant(Carburetor)\\
&\qquad\qquad\qquad \Rightarrow Serves(Carburetor, VegetarianFood)
\end{aligned}
$$

Here the antecedent is still false and hence the rule remains true under this kind of irrelevant substitution.

To review, variables in logical formulas must be either existentially ($\exists$) or universally ($\forall$) quantified. To satisfy an existentially quantified variable, there must be at least one substitution that results in a true sentence. Sentences with universally quantified variables must be true under all possible substitutions.

17.4.4 Inference

One of the most important desiderata given in Section 17.1 for a meaning representation language is that it should support inference—the ability to add valid new propositions to a knowledge base, or to determine the truth of propositions not explicitly contained within a knowledge base. This section briefly discusses **modus ponens**, the most important inference method provided by FOPC. Applications of modus ponens will be discussed in Ch. 21.

MODUS PONENS

Modus ponens is a familiar form of inference that corresponds to what is informally known as *if-then* reasoning. We can abstractly define modus ponens as follows, where α and β should be taken as FOPC formulas:

$$
\begin{array}{c}
\alpha\\
\alpha \Rightarrow \beta\\
\hline
\beta
\end{array}
$$

In general, schemas like this indicate that the formula below the line can be inferred from the formulas above the line by some form of inference. Modus ponens simply states that if the left-hand side of an implication rule is present in the knowledge base, then the right-hand side of the rule can be inferred. In the following discussions, we will refer to the left-hand side of an implication as the antecedent, and the right-hand side as the consequent.

As an example of a typical use of modus ponens, consider the following example, which uses a rule from the last section:

$$\frac{\begin{array}{l} VegetarianRestaurant(Rudys) \\ \forall x VegetarianRestaurant(x) \Rightarrow Serves(x, VegetarianFood) \end{array}}{Serves(Rudys, VegetarianFood)} \tag{17.25}$$

Here, the formula $VegetarianRestaurant(Rudys)$ matches the antecedent of the rule, thus allowing us to use modus ponens to conclude $Serves(Rudys, VegetarianFood)$.

FORWARD CHAINING

Modus ponens is typically put to practical use in one of two ways: forward chaining and backward chaining. In **forward chaining** systems, modus ponens is used in precisely the manner just described. As individual facts are added to the knowledge base, modus ponens is used to fire all applicable implication rules. In this kind of arrangement, as soon as a new fact is added to the knowledge base, all applicable implication rules are found and applied, each resulting in the addition new facts to the knowledge base. These new propositions in turn can be used to fire implication rules applicable to them. The process continues until no further facts can be deduced.

PRODUCTION SYSTEMS

The forward chaining approach has the advantage that facts will be present in the knowledge base when needed, since in a sense all inference is performed in advance. This can substantially reduce the time needed to answer subsequent queries since they should all amount to simple lookups. The disadvantage of this approach is that facts may be inferred and stored that will never be needed. **Production systems**, which are heavily used in cognitive modeling work, are forward chaining inference systems augmented with additional control knowledge that governs which rules are to be fired.

BACKWARD CHAINING

In **backward chaining**, modus ponens is run in reverse to prove specific propositions, called queries. The first step is to see if the query formula is true by determining if it is present in the knowledge base. If it is not, then the next step is to search for applicable implication rules present in the knowledge base. An applicable rule is one where the consequent of the rule matches the query formula. If there are any such rules, then the query can be proved if the antecedent of any one them can be shown to be true. Not surprisingly, this can be performed recursively by backward chaining on the antecedent as a new query. The **Prolog** programming language is a backward chaining system that implements this strategy.

To see how this works, let's assume that we have been asked to verify the truth of the proposition $Serves(Rudys, VegetarianFood)$, assuming the facts given above the line in (17.25). Since it is not present in the knowledge base, a search for an applicable rule is initiated that results in the rule given above. After substituting, the constant *Rudys* for the variable *x*, our next task is to prove the antecedent of the rule, $VegetarianRestaurant(Rudys)$, which of course is one of the facts we are given.

Note that it is critical to distinguish between reasoning via backward chaining from queries to known facts, and reasoning backwards from known consequents to unknown antecedents. To be specific, by reasoning backwards we mean that if the consequent of a rule is known to be true, we assume that the antecedent will be as well. For example, let's assume that we know that $Serves(Rudys, VegetarianFood)$ is true. Since this fact matches the consequent of our rule, we might reason backwards to the conclusion that $VegetarianRestaurant(Rudys)$.

While backward chaining is a sound method of reasoning, reasoning backwards is an invalid, though frequently useful, form of *plausible reasoning*. Plausible reasoning from consequents to antecedents is known as **abduction**, and as we will see in Ch. 21 is often useful in accounting for many of the inferences people make while analyzing extended discourses.

ABDUCTION

COMPLETE

While forward and backward reasoning are sound, neither is **complete**. This means that there are valid inferences that can not be found by systems using these methods alone. Fortunately, there is an alternative inference technique called **resolution** that is sound and complete. Unfortunately, inference systems based on resolution are far more computationally expensive than forward or backward chaining systems. In practice, therefore, most systems use some form of chaining, and place a burden on knowledge base developers to encode the knowledge in a fashion that permits the necessary inferences to be drawn.

RESOLUTION

17.5 Some Linguistically Relevant Concepts

Entire lives have been spent studying the representation of various aspects of human knowledge. These efforts have ranged from tightly focused efforts to represent individual domains such as time, to monumental efforts to encode all of our commonsense knowledge of the world (Lenat and Guha, 1991). Our focus here is considerably more modest. This section provides a brief overview of the representation of a few important topics that have clear implications for language processing. Specifically, the following sections provide introductions to the meaning representations of categories, events, time, and beliefs.

17.5.1 Categories

As we noted in Section 17.2, words with predicate-like semantics often express preferences for the semantics of their arguments in the form of selectional restrictions. These restrictions are typically expressed in the form of semantically-based categories where all the members of a category share a set of relevant features.

The most common way to represent categories is to create a unary predicate for each category of interest. Such predicates can then be asserted for each member of that category. For example, in our restaurant discussions we have been using the unary predicate *VegetarianRestaurant* as in:

$$VegetarianRestaurant(Maharani)$$

Similar logical formulas would be included in our knowledge base for each known vegetarian restaurant.

Unfortunately, in this method categories are relations, rather than full-fledged objects. It is, therefore, difficult to make assertions about categories themselves, rather than about their individual members. For example, we might want to designate the most popular member of a given category as in the following expression:

$$MostPopular(Maharani, VegetarianRestaurant)$$

Unfortunately, this is not a legal FOPC formula since the arguments to predicates in FOPC must be *Terms*, not other predicates.

REIFICATION

One way to solve this problem is to represent all the concepts that we want to make statements about as full-fledged objects via a technique called **reification**. In this case, we can represent the category of *VegetarianRestaurant* as an object just as *Maharani* is. The notion of membership in such a category is then denoted via a membership relation as in the following:

$$ISA(Maharani, VegetarianRestaurant)$$

The relation denoted by *ISA* (is a) holds between objects and the categories in which they are members. This technique can be extended to create hierarchies of categories through the use of other similar relations, as in the following:

$$AKO(VegetarianRestaurant, Restaurant)$$

Here, the relation *AKO* (a kind of) holds between categories and denotes a category inclusion relationship. Of course, to truly give these predicates meaning they would have to be situated in a larger set of facts defining categories as sets.

Ch. 19 discusses the practical use of such relations in databases of lexical relations, in the representation of selectional restrictions, and in word sense disambiguation.

17.5.2 Events

The representations for events that we have used until now have consisted of single predicates with as many arguments as are needed to incorporate all the roles associated with a given example. For example, the representation for *making a reservation* discussed in Section 17.2 consisted of a single predicate with arguments for the person making the reservation, the restaurant, the day, the time, and the number of people in the party, as in the following:

$$Reservation(Hearer, Maharani, Today, 8PM, 2)$$

In the case of verbs, this approach simply assumes that the predicate representing the meaning of a verb has the same number of arguments as are present in the verb's syntactic subcategorization frame.

Unfortunately, there are four problems with this approach that make it awkward to apply in practice:

- Determining the correct number of roles for any given event.
- Representing facts about the roles associated with an event.
- Ensuring that all the correct inferences can be derived directly from the representation of an event.
- Ensuring that no incorrect inferences can be derived from the representation of an event.

We will explore these, and other related issues, by considering a series of representations for events. This discussion will focus on the following examples of the verb *eat*:

(17.26) I ate.
(17.27) I ate a turkey sandwich.
(17.28) I ate a turkey sandwich at my desk.
(17.29) I ate at my desk.
(17.30) I ate lunch.
(17.31) I ate a turkey sandwich for lunch.
(17.32) I ate a turkey sandwich for lunch at my desk.

Clearly, the variable number of arguments for a predicate-bearing verb like *eat* poses a tricky problem. While we would like to think that all of these examples denote the same kind of event, predicates in FOPC have fixed **arity**—they take a fixed number of arguments.

ARITY

One possible solution is suggested by the way that examples like these are handled syntactically. The solution given in Ch. 16 was to create one subcategorization frame for each of the configurations of arguments that a verb allows. The semantic analog to this approach is to create as many different *eating* predicates as are needed to handle all of the ways that *eat* behaves. Such an approach would yield the following kinds of representations for examples (17.26) through (17.26).

$$\begin{aligned}&Eating_1(Speaker)\\&Eating_2(Speaker, TurkeySandwich)\\&Eating_3(Speaker, TurkeySandwich, Desk)\\&Eating_4(Speaker, Desk)\\&Eating_5(Speaker, Lunch)\\&Eating_6(Speaker, TurkeySandwich, Lunch)\\&Eating_7(Speaker, TurkeySandwich, Lunch, Desk)\end{aligned}$$

This approach simply sidesteps the issue of how many arguments the $Eating$ predicate should have by creating distinct predicates for each of the subcategorization frames. Unfortunately, this approach comes at a rather high cost. Other than the suggestive names of the predicates, there is nothing to tie these events to one another even though there are obvious logical relations among them. Specifically, if example (17.32) is true then all of the other examples are true as well. Similarly, if example (17.31) is true then examples (17.26), (17.27), and (17.30) must also be true. Such logical connections can not be made on the basis of these predicates alone. Moreover, we would expect a commonsense knowledge base to contain logical connections between concepts like $Eating$ and related concepts like $Hunger$ and $Food$.

MEANING POSTULATES

One method to solve these problems involves the use of what are called **meaning postulates**. Consider the following example postulate:

$$\forall w, x, y, z\ Eating_7(w, x, y, z) \Rightarrow Eating_6(w, x, y)$$

This postulate explicitly ties together the semantics of two of our predicates. Other postulates could be created to handle the rest of the logical relations among the various $Eatings$ and the connections from them to other related concepts.

Although such an approach might be made to work in small domains, it clearly has scalability problems. A somewhat more sensible approach is to say that examples (17.26) through (17.32) all reference the same predicate with some of the arguments missing from some of the surface forms. Under this approach, as many arguments are included in the definition of the predicate as ever appear with it in an input. Adopting the structure of a predicate like $Eating_7$ as an example would give us a predicate with four arguments denoting the eater, thing eaten, meal being eaten and the location of the eating. The following formulas would then capture the semantics of our examples:

$$\exists w,x,y\ Eating(Speaker,w,x,y)$$
$$\exists w,x\ Eating(Speaker,TurkeySandwich,w,x)$$
$$\exists w\ Eating(Speaker,TurkeySandwich,w,Desk)$$
$$\exists w,x\ Eating(Speaker,w,x,Desk)$$
$$\exists w,x\ Eating(Speaker,w,Lunch,x)$$
$$\exists w\ Eating(Speaker,TurkeySandwich,Lunch,w)$$
$$Eating(Speaker,TurkeySandwich,Lunch,Desk)$$

This approach directly yields the obvious logical connections among these formulas without the use of meaning postulates. Specifically, all of the sentences with ground terms as arguments logically imply the truth of the formulas with existentially bound variables as arguments.

Unfortunately, this approach still has at least two glaring deficiencies: it makes too many commitments, and it does not let us individuate events. As an example of how it makes too many commitments, consider how we accommodated the *for lunch* complement in examples (17.30) through (17.32); a third argument, the meal being eaten, was added to the $Eating$ predicate. The presence of this argument implicitly makes it the case that all eating events are associated with a meal (i.e., breakfast, lunch, or dinner). More specifically, the existentially quantified variable for the meal argument in the above examples states that there is some formal meal associated with each of these eatings. This is clearly silly since one can certainly eat something independent of it being associated with a meal.

To see how this approach fails to properly individuate events, consider the following formulas.

$$\exists w,x\ Eating(Speaker,w,x,Desk)$$
$$\exists w,x\ Eating(Speaker,w,Lunch,x)$$
$$\exists w,x\ Eating(Speaker,w,Lunch,Desk)$$

If we knew that the first two formulas were referring to the same event, they could be combined to create the third representation. Unfortunately, with the current representation we have no way of telling if this is possible. The independent facts that *I ate at my desk* and *I ate lunch* do not permit us to conclude that *I ate lunch at my desk*. Clearly what is lacking is some way of referring to the events in question.

As with categories, we can solve these problems if we employ reification to elevate events to objects that can be quantified and related to other objects via sets of defined relations (Davidson, 1967; Parsons, 1990). Consider the representation of example (17.27) under this kind of approach.

$$\exists w\ ISA(w,Eating)$$
$$\wedge Eater(w,Speaker) \wedge Eaten(w,TurkeySandwich)$$

This representation states that there is an eating event where the *Speaker* is doing the eating and a *TurkeySandwich* is being eaten. The meaning representations for examples (17.26) and (17.31) can be constructed similarly.

$$\exists w\, ISA(w, Eating) \wedge Eater(w, Speaker)$$

$$\begin{aligned}\exists w\, ISA(w, Eating)&\\ &\wedge Eater(w, Speaker) \wedge Eaten(w, TurkeySandwich)\\ &\wedge MealEaten(w, Lunch)\end{aligned}$$

Under this reified-event approach:

- There is no need to specify a fixed number of arguments for a given surface predicate; rather as many roles and fillers can be glued on as appear in the input.
- No more roles are postulated than are mentioned in the input.
- The logical connections among closely related examples are satisfied without the need for meaning postulates.

17.5.3 Representing Time

In the preceding discussion of events, we did not address the issue of representing the time when the represented events are supposed to have occurred. The representation of such information in a useful form is the domain of **temporal logic**. This discussion will serve to introduce the most basic concerns of temporal logic along with a brief discussion of the means by which human languages convey temporal information, which among other things includes **tense logic**, the ways that verb tenses convey temporal information.

TEMPORAL LOGIC

TENSE LOGIC

The most straightforward theory of time hold that it flows inexorably forward, and that events are associated with either points or intervals in time, as on a timeline. Given these notions, an ordering can be imposed on distinct events by situating them on the timeline. More specifically, we can say that one event *precedes* another, if the flow of time leads from the first event to the second. Accompanying these notions in most theories is the idea of the current moment in time. Combining this notion with the idea of a temporal ordering relationship yields the familiar notions of past, present and future.

Not surprisingly, there are a large number of schemes for representing this kind of temporal information. The one presented here is a fairly simple one that stays within the FOPC framework of reified events that we have been pursuing. Consider the following examples:

(17.33) I arrived in New York.

(17.34) I am arriving in New York.

(17.35) I will arrive in New York.

These sentences all refer to the same kind of event and differ solely in the tense of the verb. In our current scheme for representing events, all three would share the following kind of representation, which lacks any temporal information:

$$\begin{array}{l} \exists w\ \mathit{ISA}(w, \mathit{Arriving}) \\ \qquad \wedge \mathit{Arriver}(w, \mathit{Speaker}) \wedge \mathit{Destination}(w, \mathit{NewYork}) \end{array}$$

The temporal information provided by the tense of the verbs can be exploited by predicating additional information about the event variable *w*. Specifically, we can add temporal variables representing the interval corresponding to the event, the end point of the event, and temporal predicates relating this end point to the current time as indicated by the tense of the verb. Such an approach yields the following representations for our *arriving* examples:

$$\begin{array}{l} \exists i, e, w, t\ \mathit{ISA}(w, \mathit{Arriving}) \\ \qquad \wedge \mathit{Arriver}(w, \mathit{Speaker}) \wedge \mathit{Destination}(w, \mathit{NewYork}) \\ \qquad \mathit{IntervalOf}(w, i) \wedge \mathit{EndPoint}(i, e) \wedge \mathit{Precedes}(e, \mathit{Now}) \end{array}$$

$$\begin{array}{l} \exists i, e, w, t\ \mathit{ISA}(w, \mathit{Arriving}) \\ \qquad \wedge \mathit{Arriver}(w, \mathit{Speaker}) \wedge \mathit{Destination}(w, \mathit{NewYork}) \\ \qquad \mathit{IntervalOf}(w, i) \wedge \mathit{MemberOf}(i, \mathit{Now}) \end{array}$$

$$\begin{array}{l} \exists i, e, w, t\ \mathit{ISA}(w, \mathit{Arriving}) \\ \qquad \wedge \mathit{Arriver}(w, \mathit{Speaker}) \wedge \mathit{Destination}(w, \mathit{NewYork}) \\ \qquad \mathit{IntervalOf}(w, i) \wedge \mathit{EndPoint}(i, e) \wedge \mathit{Precedes}(\mathit{Now}, e) \end{array}$$

This representation introduces a variable to stand for the interval of time associated with the event, and a variable that stands for the end of that interval. The two-place predicate *Precedes* represents the notion that the first time point argument precedes the second in time; the constant *Now* refers to the current time. For past events, the end point of the interval must precede the current time. Similarly, for future events the current time must precede the end of the event. For events happening in the present, the current time is contained within the event interval.

Unfortunately, the relation between simple verb tenses and points in time is by no means straightforward. Consider the following examples:

(17.36) Ok, we fly from San Francisco to Boston at 10.

(17.37) Flight 1390 will be at the gate an hour now.

In the first example, the present tense of the verb *fly* is used to refer to a future event, while in the second the future tense is used to refer to a past event.

More complications occur when we consider some of the other verb tenses. Consider the following examples:

(17.38) Flight 1902 arrived late.

(17.39) Flight 1902 had arrived late.

REFERENCE POINT

Although both refer to events in the past, representing them in the same way seems wrong. The second example seems to have another unnamed event lurking in the background (e.g., Flight 1902 had already arrived late *when* something else happened). To account for this phenomena, Reichenbach (1947) introduced the notion of a **reference point**. In our simple temporal scheme, the current moment in time is equated with the time of the utterance, and is used as a reference point for when the event occurred (before, at, or after). In Reichenbach's approach, the notion of the reference point is separated out from the utterance time and the event time. The following examples illustrate the basics of this approach:

(17.40) When Mary's flight departed, I ate lunch.

(17.41) When Mary's flight departed, I had eaten lunch.

In both of these examples, the eating event has happened in the past, i.e. prior to the utterance. However, the verb tense in the first example indicates that the eating event began when the flight departed, while the second example indicates that the eating was accomplished prior to the flight's departure. Therefore, in Reichenbach's terms the *departure* event specifies the reference point. These facts can be accommodated by asserting additional constraints relating the *eating* and *departure* events. In the first example, the reference point precedes the *eating* event, and in the second example, the eating precedes the reference point. Figure 17.5 illustrates Reichenbach's approach with the primary English tenses. Exercise 17.9 asks you to represent these examples in FOPC.

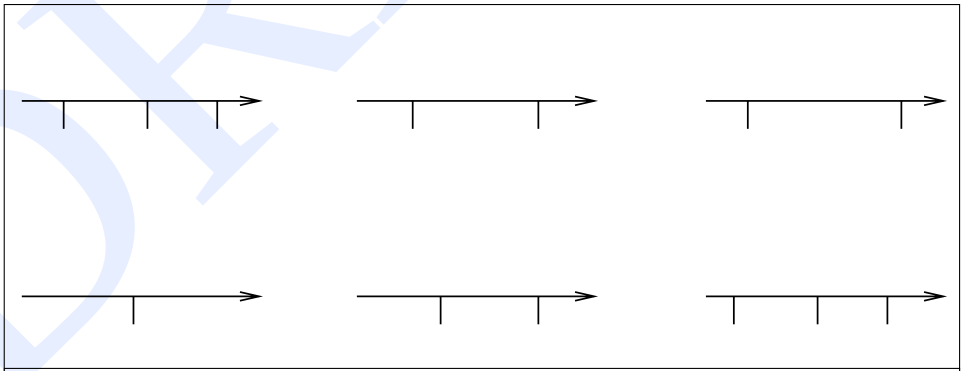


Figure 17.5 Reichenbach's approach applied to various English tenses. In these diagrams, time flows from left to right, an **E** denotes the time of the event, an **R** denotes the reference time, and an **U** denotes the time of the utterance.

This discussion has focused narrowly on the broad notions of past, present, and future and how they are signaled by verb tenses. Of course, languages also have many other more direct and more specific ways to convey temporal information, including the use of a wide variety of temporal expressions as in the following ATIS examples:

(17.42) I'd like to go at 6:45, in the morning.

(17.43) Somewhere around noon, please.

(17.44) Later in the afternoon, near 6PM.

As we will see in the next chapter, grammars for such temporal expressions are of considerable practical importance in information extraction and question-answering applications.

Finally, we should note that there is a systematic conceptual organization reflected in examples like these. In particular, temporal expressions in English are frequently expressed in spatial terms, as is illustrated by the various uses of *at*, *in*, *somewhere* and *near* in these examples (Lakoff and Johnson, 1980; Jackendoff, 1983) Metaphorical organizations such as these, where one domain is systematically expressed in terms of another, will be discussed in more detail in Ch. 19.

17.5.4 Aspect

In the last section, we discussed ways to represent the time of an event with respect to the time of an utterance describing it. In this section, we address the notion of **aspect**, which concerns a cluster of related topics, including whether an event has ended or is ongoing, whether it is conceptualized as happening at a point in time or over some interval, and whether or not any particular state in the world comes about because of it. Based on these and related notions, event expressions have traditionally been divided into four general classes: **statives**, **activities**, **accomplishments**, and **achievements**. The following examples provide prototypical instances of each class.

ASPECT

Stative: I know my departure gate.

Activity: John is flying.

Accomplishment: Sally booked her flight.

Achievement: She found her gate.

Although the earliest versions of this classification were discussed by Aristotle, the one presented here is due to Vendler (1967). In the following discussion, we'll present a brief characterization of each of the four classes, along with some diagnostic techniques suggested in Dowty (1979) for identifying examples of each kind.

STATIVE EXPRESSIONS

Stative expressions represent the notion of an event participant having a particular property, or being in a state, at a given point in time. As such, they can be thought of as capturing an aspect of a world at a single point in time. Consider the following ATIS examples.

(17.45) I like Flight 840 arriving at 10:06.

(17.46) I need the cheapest fare.

(17.47) I have a round trip ticket for $662.

(17.48) I want to go first class.

In examples like these, the event participant denoted by the subject can be seen as experiencing something at a specific point in time. Whether or not the experiencer was in the same state earlier, or will be in the future is left unspecified.

There are a number of diagnostic tests for identifying statives. As an example, stative verbs are distinctly odd when used in the progressive form.

(17.49) *I am needing the cheapest fare on this day.

(17.50) *I am wanting to go first class.

We should note that in these and subsequent examples, we are using an * to indicate a broadened notion of ill-formedness that may include both semantic and syntactic factors.

Statives are also odd when used as imperatives.

(17.51) *Need the cheapest fare!

Finally, statives are not easily modified by adverbs like *deliberately* and *carefully*.

(17.52) *I deliberately like Flight 840 arriving at 10:06.

(17.53) *I carefully like Flight 840 arriving at 10:06.

ACTIVITY EXPRESSIONS

Activity expressions describe events undertaken by a participant that have no particular end point. Unlike statives, activities are seen as occurring over some span of time, and are therefore not associated with single points in time. Consider the following examples:

(17.54) She drove a Mazda.

(17.55) I live in Brooklyn.

These examples both specify that the subject is engaged in, or has engaged in, the activity specified by the verb for some period of time.

Unlike statives, activity expressions are fine in both the progressive and imperative forms.

(17.56) She is living in Brooklyn.

(17.57) Drive a Mazda!

However, like statives, activity expressions are odd when temporally modified with temporal expressions using *in*.

(17.58) *I live in Brooklyn in a month.

(17.59) *She drove a Mazda in an hour.

They can, however, successfully be used with *for* temporal adverbials, as in the following examples:

(17.60) I live in Brooklyn for a month.

(17.61) She drove a Mazda for an hour.

ACCOMPLISHMENT EXPRESSIONS

Unlike activities, **accomplishment expressions** describe events that have a natural end point and result in a particular state. Consider the following examples:

(17.62) He booked me a reservation.

(17.63) United flew me to New York.

In these examples, there is an event that is seen as occurring over some period of time that ends when the intended state is accomplished.

A number of diagnostics can be used to distinguish accomplishment events from activities. Consider the following examples, which make use of the word *stop* as a test.

(17.64) I stopped living in Brooklyn.

(17.65) She stopped booking my flight.

In the first example, which is an activity, one can safely conclude that the statement *I lived in Brooklyn* even though this activity came to an end. However, from the second example, one can not conclude the statement *She booked her flight*, since the activity was stopped before the intended state was accomplished. Therefore, although stopping an activity entails that the activity took place, stopping an accomplishment event indicates that the event did not succeed.

Activities and accomplishments can also be distinguished by how they can be modified by various temporal adverbials. Consider the following examples:

(17.66) *I lived in Brooklyn in a year.

(17.67) She booked a flight in a minute.

In general, accomplishments can be modified by *in* temporal expressions, while simple activities can not.

ACHIEVEMENT EXPRESSIONS

The final aspectual class, **achievement expressions**, are similar to accomplishments in that they result in a state. Consider the following examples:

(17.68) She found her gate.

(17.69) I reached New York.

Unlike accomplishments, achievement events are thought of as happening in an instant, and are not equated with any particular activity leading up to the state. To be more specific, the events in these examples may have been preceded by extended *searching* or *traveling* events, but the events corresponding directly to *found* and *reach* are conceived of as points not intervals.

The point-like nature of these events has implications for how they can be temporally modified. In particular, consider the following examples:

(17.70) I lived in New York for a year.

(17.71) *I reached New York for a few minutes.

Unlike activity and accomplishment expressions, achievements can not be modified by *for* adverbials.

Achievements can also be distinguished from accomplishments by employing the word *stop*, as we did earlier. Consider the following examples:

(17.72) I stopped booking my flight.

(17.73) *I stopped reaching New York.

As we saw earlier, using *stop* with an accomplishment expression results in a failure to reach the intended state. Note, however, that the resulting expression is perfectly well-formed. On the other hand, using *stop* with an achievement example is unacceptable.

We should note that since both accomplishments and achievements are events that result in a state, they are sometimes characterized as sub-types of a single aspectual class. Members of this combined class are known as **telic eventualities**. TELIC EVENTUALITIES

Before moving on, we should make two points about this classification scheme. The first point is that event expressions can easily be shifted from one class to another. Consider the following examples:

(17.74) I flew.

(17.75) I flew to New York.

The first example is a simple activity; it has no natural end point and can not be temporally modified by *in* temporal expressions. On the other hand, the second example is clearly an accomplishment event since it has an end point, results in a particular state, and can be temporally modified in all the ways that accomplishments can. Clearly the classification of an event is not solely governed by the verb, but by the semantics of the entire expression in context.

The second point is that while classifications such as this one are often useful, they do not *explain* why it is that events expressed in natural languages fall into these particular classes. We will revisit this issue in Ch. 19 where we will sketch a representational approach due to Dowty (1979) that accounts for these classes.

17.5.5 Representing Beliefs

There are a fair number of words and expressions that have what might be called a *world creating* ability. By this, we mean that their meaning representations contain logical formulas that are not intended to be taken as true in the real world, but rather as part of some kind of hypothetical world. In addition, these meaning representations often denote a relation from the speaker, or some other entity, to this hypothetical world. Examples of words that have this ability are *believe*, *want*, *imagine* and *know*. World-creating words generally take various sentence-like constituents as arguments.

Consider the following example:

(17.76) I believe that Mary ate British food.

Applying our event-oriented approach we would say that there are two events underlying this sentence: a believing event relating the speaker to some specific belief, and an eating event that plays the role of the believed thing. Ignoring temporal information, a straightforward application of our reified event approach would produce the following kind of representation:

$$\begin{aligned}&\exists u,v\; ISA(u,Believing) \wedge ISA(v,Eating)\\ &\quad \wedge Believer(u,Speaker) \wedge BelievedProp(u,v)\\ &\quad \wedge Eater(v,Mary) \wedge Eaten(v,BritishFood)\end{aligned}$$

This seems relatively straightforward, all the right roles are present and the two events are tied together in a reasonable way. Recall, however, that in conjunctive representations like this all of the individual conjuncts must be taken to be true. In this case, this results in a statement that there actually was an eating of British food by Mary. Specifically, by breaking this formula apart into separate formulas by conjunction elimination, the following formula can be produced:

$$\begin{aligned}&\exists v\; ISA(v,Eating)\\ &\quad \wedge Eater(v,Mary) \wedge Eaten(v,BritishFood)\end{aligned}$$

This is clearly more than we want to say. The fact that the speaker believes this proposition does not make it true; it is only true in the world represented by the speaker's beliefs. What is needed is a representation that has a structure similar to this, but where the $Eating$ event is given a special status.

Note that reverting to the simpler predicate representations we used earlier in this chapter does not help. A common mistake using such representations would be to represent this sentence with the following kind of formula:

$$Believing(Speaker,Eating(Mary,BritishFood))$$

The problem with this representation is that it is not even valid FOPC. The second argument to the *Believing* predicate should be a FOPC term, not a formula.

This syntactic error reflects a deeper semantic problem. Predicates in FOPC hold between the objects in the domain being modeled, not between the relations that hold among the objects in the domain. Therefore, FOPC lacks a meaningful way to assert relations about full propositions, which is unfortunately exactly what words like *believe*, *want*, *imagine* and *know* want to do.

The standard method for handling this situation is to augment FOPC with *operators* that allow us to make statements about full logical formulas. Let's consider how this approach might work in the case of example (17.76). We can introduce an operator called *Believes* that takes two FOPC formulas as its arguments: a formula designating a believer, and a formula designating the believed proposition. Applying this operator would result in the following meaning representation:

$$\begin{aligned} & Believes(Speaker, \exists v ISA(v, Eating) \\ & \quad \wedge Eater(v, Mary) \wedge Eaten(v, BritishFood) \end{aligned}$$

Under this approach, the contribution of the word *believes* to this meaning representation is not a FOPC proposition at all, but rather an operator that is applied to the believed proposition. Therefore, as we discuss in Ch. 18, these world creating verbs play quite a different role in the semantic analysis than more ordinary verbs like *eat*.

As one might expect, keeping track of who believes what about whom at any given point in time gets rather complex. As we will see in Ch. 21, this is an important task in interactive systems that must track users' beliefs as they change during the course of a dialogue.

MODAL OPERATORS

MODAL LOGIC

Operators like *Believes* that apply to logical formulas are known as **modal operators**. Correspondingly, a logic augmented with such operators is known as a **modal logic**. Modal logics have found many uses in the representation of commonsense knowledge in addition to the modeling of belief, among the more prominent are representations of time and hypothetical worlds.

Not surprisingly, modal operators and modal logics raise a host of complex theoretical and practical problems that we cannot even begin to do justice to here. Among the more important issues are the following:

- How inference works in the presence of specific modal operators.
- The kinds of logical formula that particular operators can be applied to.
- How modal operators interact with quantifiers and logical connectives.
- The influence of these operators on the equality of terms across formulas.

The last issue in this list has consequences for modeling agent's knowledge and beliefs in dialogue systems and deserves some elaboration here. In standard FOPC systems, logical terms that are known to be equal to one another can be freely substituted without having any effect on the truth of sentences they occur in. Consider the following examples:

(17.77) Snow has delayed Flight 1045.

(17.78) John's sister's flight serves dinner.

Assuming that these two flights are the same, substituting *Flight 1045* for *John's sister's flight* has no effect on the truth of either sentence.

Now consider, the following variation on the first example:

(17.79) John knows that snow has delayed Flight 1045.

(17.80) John knows that his sister's flight serves dinner.

Here the substitution does not work. John may well know that Flight 1045 has been delayed without knowing that his sister's flight is delayed, simply because he may not know the number of his sister's flight. In other words, even if we assume that these sentences are true, and that John's sister is on Flight 1045, we can not say anything about the truth of the following sentence:

(17.81) John knows that snow has delayed his sister's flight.

Settings like this where a modal operator like *Know* is involved are called **referentially opaque**. In referentially opaque settings, substitution of equal terms may or may not succeed. Ordinary settings where such substitutions always work are said to be **referentially transparent**.

REFERENTIALLY OPAQUE

REFERENTIALLY TRANSPARENT

17.5.6 Pitfalls

As noted in Section 17.4, there are a number of common mistakes in representing the meaning of natural language utterances, that arise from confusing, or equating, elements from real languages with elements in FOPC. Consider the following example, which on the surface looks like a candidate for a standard implication rule:

(17.82) If you're interested in baseball, the Rockies are playing tonight.

A straightforward translation of this sentence into FOPC might look something like this:

$$\begin{aligned} &HaveInterestIn(Hearer, Baseball) \\ &\qquad \Rightarrow Playing(Rockies, Tonight) \end{aligned}$$

This representation is flawed for a large number of reasons. The most obvious ones arise from the semantics of FOPC implications. In the event that the hearer is not interested in baseball, this formula becomes meaningless. Specifically, we can not draw any conclusion about the consequent clause when the antecedent is false. But of course this is a ridiculous conclusion, we know that the Rockies game will go forward regardless of whether or not the hearer happens to like baseball. Exercise 17.10 asks you to come up with a more reasonable FOPC translation of this example.

Now consider the following example:

(17.83) One more beer and I'll fall off this stool.

Again, a simple-minded translation of this sentence might consist of a conjunction of two clauses: one representing a drinking event and one representing a falling event. In this case, the surface use of the word *and* obscures the fact that this sentence instead has an implication underlying it. The lesson of both of these examples is that English words like *and*, *or* and *if* are only tenuously related to the elements of FOPC with the same names.

Along the same lines, it is important to remember the complete lack of significance of the names we make use of in representing FOPC formulas. Consider the following constant:

$$InexpensiveVegetarianIndianFoodOnTuesdays$$

Despite its impressive morphology, this term, by itself, has no more meaning than a constant like $X99$ would have. See McDermott (1976) for a discourse on the inherent dangers of such naming schemes.

As we noted at the beginning of this chapter,

Basics and current applications/versions OWL as a kind of description logic as applied to the Semantic Web. See the Brachman thing.

basically we're going to arrange the concepts in a domain into a hierarchy. Then we're going to relate the elements in the hierarchy via type slot-filler relations (value/resritictions). Then there's inheritance.

unary and binary predicates only? I.e. what you get in a network.

17.6 Related Representational Approaches

Over the years, a fair number of representational schemes have been invented to capture the meaning of linguistic utterances for use in natural language processing systems. Other than First Order Logic, the most widely used schemes have been **semantic networks** and **frames**, which are also known as **slot-filler** representations. The KL-ONE (Brachman and Schmolze, 1985), and KRL (Bobrow and Winograd, 1977) systems were influential efforts to represent knowledge for use in natural language processing systems.

SEMANTIC NETWORKS

FRAMES

In semantic networks, objects are represented as nodes in a graph, with relations between objects being represented by named links. In frame-based systems, objects are represented as feature-structures similar to those discussed in Ch. 16, which can, of course, also be naturally represented as graphs. In this approach features are called slots and the values, or fillers, of these slots can either be atomic values or other embedded frames. The following diagram illustrates how example (17.76) might be captured in a frame-based approach.

I believe Mary ate British food.

$$\begin{bmatrix} \textsc{Believing} & \\ \textsc{Believer} & \textsc{Speaker} \\ \textsc{Believed} & \begin{bmatrix} \textsc{Eating} & \\ \textsc{eater} & \textsc{Mary} \\ \textsc{eaten} & \textsc{BritishFood} \end{bmatrix} \end{bmatrix}$$

It is now widely accepted that meanings represented in these approaches can in principle be translated into equivalent statements in FOL with relative ease. The difficulty is that in many of these approaches the semantics of a statement is defined procedurally. That is, the meaning arises from whatever the system that interprets it does with it.

17.6.1 Description Logics

Description Logics can be viewed as an effort to better understand and specify the semantics of these earlier **structured network representations**, and to provide a conceptual framework that is especially well-suited to certain kinds of domain modeling. Formally, the term Description Logics refers to a family of logical approaches that correspond to varying subsets of FOL. The various restrictions placed on the expressiveness of Description Logics serve to guarantee the tractability of various critical kinds of inference. Our focus here, however, will be on the modeling aspects of DLs rather than computational complexity issues.

TERMINOLOGY
TBOX
ABOX
ONTOLOGY

When using Description Logics to model an application domain, the emphasis is on the representation of knowledge about categories, individuals that belong to those categories, and the relationships that can hold among these individuals. The set of categories, or concepts, that make up the particular application domain is called its **Terminology**. The portion of a knowledge-base that contains the terminology is traditionally called the **TBox**; this is in contrast to the **ABox** that contains facts about individuals. The terminology is typically arranged into a hierarchical organization called an **Ontology** that captures the subset/superset relations among the categories.

To illustrate this approach, let's return to our earlier culinary domain, which included notions like restaurants, cuisines, and patrons, among others. We represented concepts like these in FOL by using unary predicates such as $Restaurant(x)$; the DL equivalent simply omits the variable, so the category corresponding to the notion of a restaurant is simply written as Restaurant.[2] To capture the notion that a particular domain element, such as *Frasca*, is a restaurant we simply assert Restaurant(Frasca) in much the same way we would in FOL. The semantics of these categories is specified in precisely the same way that was introduced earlier in Sec. 17.3: a category like Restaurant simply denotes the set of domain elements that are restaurants.

Having specified the categories of interest in a state of affairs, the next step is to arrange these categories into a hierarchical structure. There are two ways to capture the hierarchical relationships present in a terminology: we can directly assert relations between categories that are related hierarchically, or we can provide complete definitions for our concepts and then rely on these definitions to infer hierarchical relationships. The choice between these methods hinges on the use to which the resulting categories will be put and the feasibility of formulating precise definitions for many naturally occurring categories. We'll discuss the first option here, and the return to the notion of definitions later in this section.

SUBSUMPTION

To directly specify a hierarchical structure, we can assert **subsumption** relations between the appropriate concepts in a terminology. The subsumption relation is conventionally written as $C \sqsubseteq D$, and is read as C is subsumed by D; that is, all members of the category C are also members of the category D. Not surprisingly, the formal semantics of this relation is provided by a simple set relation; any domain element that is in the set denoted by C is also in the set denoted by D.

Continuing with our restaurant theme, adding the following statements to the TBox asserts that all restaurants are commercial establishments, and moreover that there are various sub-types of restaurants.

$$\begin{aligned}
\mathsf{Restaurant} &\sqsubseteq \mathsf{CommercialEstablishment} \\
\mathsf{ItalianRestaurant} &\sqsubseteq \mathsf{Restaurant} \\
\mathsf{ChineseRestaurant} &\sqsubseteq \mathsf{Restaurant} \\
\mathsf{MexicanRestaurant} &\sqsubseteq \mathsf{Restaurant}
\end{aligned}$$

Ontologies such as this are conventionally illustrated using diagrams such as the one shown in Fig. 17.6 where subsumption relations are denoted by links between the nodes representing the categories.

Note however that it was precisely the vague nature of network diagrams like this that motivated the development of Description Logics. For example, from this diagram we can't tell whether or not the given set of categories is exhaustive or disjoint. That is, we can't tell if these are all the kinds of restaurants that we'll be dealing with in our domain, or whether there might be others. We also can't

[2] DL statements are conventionally typeset with a sans serif font. We'll follow that convention here, reverting back to our standard mathematical notation when giving FOL equivalents of DL statements.

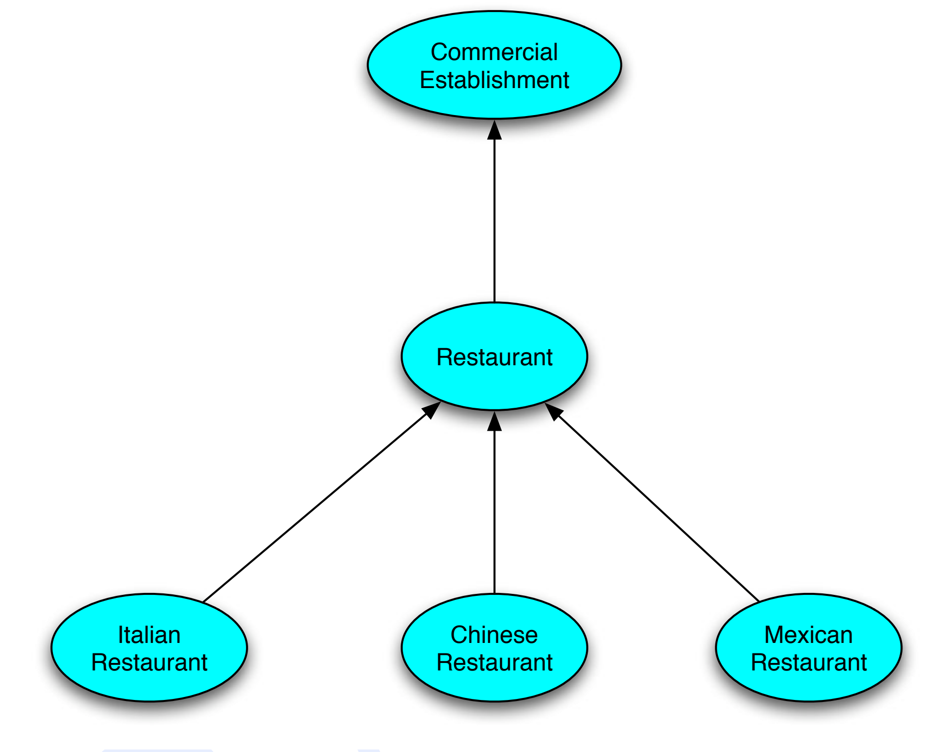


Figure 17.6 A graphical network representation of a set of subsumption relations in the restaurant domain.

tell if an individual restaurant must fall into only *one* of these categories, or if it is possible, for example, for a restaurant to be *both* Italian and Chinese. The DL statements given above are more transparent in their meaning; they simply assert a set of subsumption relations between categories and make no claims about coverage or mutual exclusion.

If an application requires coverage and disjointness information then it needs to be made explicitly. The simplest ways to capture this kind of information is through the use of negation and disjunction operators. For example, the following assertion would tell us that Chinese restaurants can't also be Italian restaurants.

$$\mathsf{ChineseRestaurant} \sqsubseteq \textbf{not}\ \mathsf{ItalianRestaurant}$$

Specifying that a set of sub-concepts covers a category can be achieved with with disjunction, as in the following:

$$\mathsf{Restaurant} \sqsubseteq (\textbf{or}\ \mathsf{ItalianRestaurant}\ \mathsf{ChineseRestaurant}\ \mathsf{MexicanRestaurant})$$

Of course, having a hierarchy such as the one given in Fig. 17.6 tells us next to nothing about the concepts in it. We certainly don't know anything about what makes a restaurant a restaurant, much less Italian, Chinese or expensive. What is needed are additional assertions about what it means to be a member of any of these categories. In Description Logics such statements come in the form of relations between the concepts being described and other concepts in the domain. In keeping with its origins in structured network representations, relations in Description Logics are typically binary and are often referred to as roles, or role-relations.

To see how such relations work, let's consider some of the facts about restaurants discussed earlier in the chapter. We'll use the hasCuisine relation to capture information as to what kinds of food restaurants serve, and the hasPriceRange relation to capture how pricey particular restaurants tend to be. We can use these relations to say something more concrete about our various classes of restaurants. Let's start with our ItalianRestaurant concept. As a first approximation, we might say something uncontroversial like Italian restaurants serve Italian cuisine. To capture these notions, let's first add some new concepts to our terminology to represent various kinds of cuisine.

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\mathsf{MexicanCuisine} & \sqsubseteq & \mathsf{Cuisine} & \mathsf{ExpensiveRestaurant} & \sqsubseteq & \mathsf{Restaurant} \\
\mathsf{ItalianCuisine} & \sqsubseteq & \mathsf{Cuisine} & \mathsf{ModerateRestaurant} & \sqsubseteq & \mathsf{Restaurant} \\
\mathsf{ChineseCuisine} & \sqsubseteq & \mathsf{Cuisine} & \mathsf{CheapRestaurant} & \sqsubseteq & \mathsf{Restaurant} \\
\mathsf{VegetarianCuisine} & \sqsubseteq & \mathsf{Cuisine} & & &
\end{array}$$

Next let's revise our earlier version of ItalianRestaurant to capture cuisine information.

$$\mathsf{ItalianRestaurant} \ \sqsubseteq \ \mathsf{Restaurant} \sqcap \exists \mathsf{hasCuisine.ItalianCuisine}$$

The way to read this expression is that individuals in the category ItalianRestaurant are subsumed both by the category Restaurant, and by an unnamed class defined by the existential clause — the set of entities that serve Italian cuisine. An equivalent statement in FOL would be:

$$\forall x ItalianRestaurant(x) \rightarrow Restaurant(x) \wedge (\exists y Serves(x,y) \wedge ItalianCuisine(y))$$

This FOL translation should make it clear what the DL assertions given above do, and do not entail. In particular, they don't say that domain entities classified as Italian restaurants can't engage in other relations like being expensive, or even serving Chinese cuisine. And critically, they don't say much about domain entities that we know do serve Italian cuisine. In fact, inspection of the FOL translation the makes it clear that we can't *infer* that any new entities belong to this category based on their characteristics. The best we can do is infer new facts about restaurants that we're explicitly told are members of this category

Of course, inferring the category membership of individuals given certain characteristics is a common and critical reasoning task that we need to support. This brings us back to the alternative approach to creating hierarchical structures in a terminology: actually providing a **definition** of the categories we're creating in the form of necessary and sufficient conditions for category membership. In this case, we might explicitly provide a definition for $\mathsf{ItalianRestaurant}$ as being those restaurants that serve Italian cuisine, and $\mathsf{ModerateRestaurant}$ as being those whose price range is moderate.

$$\begin{aligned} \mathsf{ItalianRestaurant} &\equiv \mathsf{Restaurant} \sqcap \exists \mathsf{hasCuisine.ItalianCuisine} \\ \mathsf{ModerateRestaurant} &\equiv \mathsf{Restaurant} \sqcap \mathsf{hasPriceRange.ModeratePrices} \end{aligned}$$

While our earlier statements provided necessary conditions for membership in these categories, these statements provide both necessary and sufficient conditions.

Finally, let's now consider the superficially similar case of vegetarian restaurants. Clearly, vegetarian restaurants are those that serve vegetarian cuisine. But they don't merely serve vegetarian fare, that's all they serve. We can accommodate this kind of constraint by adding an additional restriction in the form of a universal quantifier to our earlier description of $\mathsf{VegetarianRestaurants}$, as follows:

$$\begin{aligned} \mathsf{VegetarianRestaurant} \equiv\ & \mathsf{Restaurant} \\ & \sqcap \exists \mathsf{hasCuisine.VegetarianCuisine} \\ & \sqcap \forall \mathsf{hasCuisine.VegetarianCuisine} \end{aligned}$$

Inference

Paralleling the focus of Description Logics on categories, relations and individuals, is a processing focus on a restricted subset of logical inference. Rather than employ the full range of reasoning permitted by FOL, DL reasoning systems emphasize the closely coupled problems of subsumption and instance checking.

SUBSUMPTION

INSTANCE CHECKING

Subsumption, as a form of inference, is the task of determining whether a superset/subset relationship exists between two concepts based on the facts asserted in a terminology. Correspondingly, **instance checking** asks if an individual can be a member of a particular category given the facts we know about both the individual and the terminology. The inference mechanisms underlying subsumption and instance checking go beyond simply checking for explicitly stated subsumption relations in a terminology. They must explicitly reason using the relational information asserted about the terminology to infer appropriate subsumption and membership relations.

Returning to our restaurant domain, let's add a new kind of restaurant using the following statement:

$$\mathsf{OliveGarden} \sqsubseteq \mathsf{ModerateRestaurant} \sqcap \exists \mathsf{hasCuisine}.\mathsf{ItalianCuisine}$$

Given this assertion, we might ask whether the OliveGarden chain of restaurants might be classified as an Italian restaurant, or a vegetarian restaurant. More precisely, we can pose the following questions to our reasoning system:

$$\mathsf{OliveGarden} \sqsubseteq \mathsf{ItalianRestaurant}$$
$$\mathsf{OliveGarden} \sqsubseteq \mathsf{VegetarianRestaurant}$$

The answer to the first question is positive since OliveGarden meets the criteria we specified for the category ItalianRestaurant: it's a Restaurant since we explicitly classified it as a ModerateRestaurant, which is a subtype of Restaurant, and it meets the has.Cuisine class restriction since we've asserted that directly.

The answer to the second question is negative. Recall, that our criteria for vegetarian restaurants contains two requirements: it has to serve vegetarian fare, and that's all it can serve. Our current definition for OliveGarden fails on both counts since we have not asserted any relations that state that OliveGarden serves vegetarian fare, and the relation we have asserted, hasCuisine.ItalianCuisine, contradicts the second criteria.

A related reasoning task, based on the basic subsumption inference, is to derive the **implied hierarchy** for a terminology given facts about the categories in the terminology. This task roughly corresponds to a repeated application of the subsumption operator to pairs of concepts in the terminology. Given our current collection of statements, the expanded hierarchy shown in Fig. 17.7 can be inferred. You should convince yourself that this diagram contains all and only the subsumption links that should be present given our current knowledge.

Note that whereas subsumption is all about concepts and categories, **instance checking** is the task of determining whether a particular individual can be classified as a member of a particular category. This process takes what is known about a given individual, in the form of relations and explicit categorical statements, and then compares that information against what is known about the current terminology. It then returns a list of *the most specific* categories to which the individual can belong.

As an example of a categorization problem consider an establishment that we're told is a restaurant and serves Italian cuisine.

$$\mathsf{Restaurant}(\mathsf{Gondolier})$$
$$\mathsf{hasCuisine}(\mathsf{Gondolier}, \mathsf{ItalianCuisine})$$

Here, we're being told that the entity denoted by the term Gondolier is a restaurant and serves Italian food. Given this new information and the contents of our current

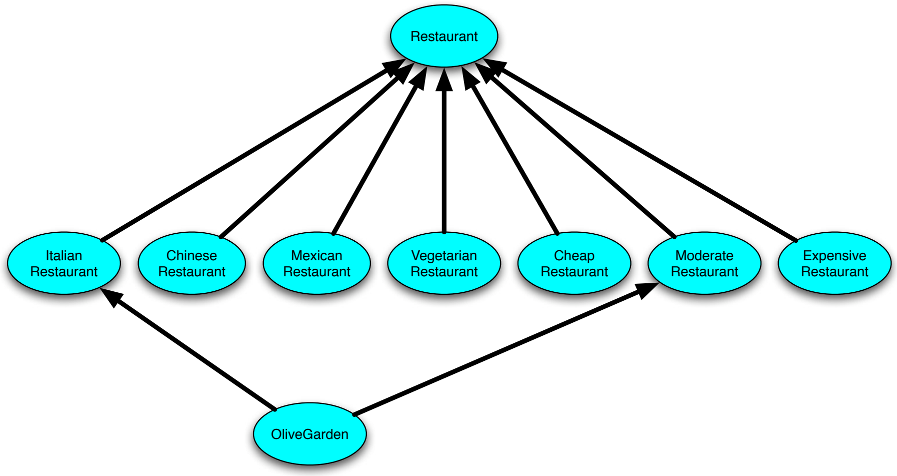


Figure 17.7 A graphical network representation of the complete set of subsumption relations in the restaurant domain given the current set of assertions in the TBox.

TBox, we might reasonably like to ask if this is an Italian restaurant, a vegetarian restaurant or if it has moderate prices.

Assuming the definitional statements given earlier, we can indeed categorize the Gondolier as an Italian restaurant. That is, the information we've been given about it meets the necessary and sufficient conditions required for membership in this category. And as with the OliveGarden category, this individual fails to match the stated criteria for the VegetarianRestaurant. Finally, the Gondolier might also turn out to be an moderately priced restaurant, but we can't tell at this point since we don't know anything about its prices. What this means is that given our current knowledge the answer to the query ModerateRestaurant(Gondolier) would be false since it lacks the required hasPriceRange relation.

The implementation of subsumption, instance checking, as well as other kinds of inferences needed for practical applications, varies depending on the expressivity of the Description Logic being used. However, for Description Logics of even modest power, the primary implementation techniques are based on satisfiability methods that in turn rely on the underlying model-based semantics introduced earlier in this chapter.

OWL and the Semantic Web

The highest-profile role for Description Logics has been as a part of the development of the Semantic Web. The Semantic Web is an effort to provide a way to formally specific the semantics of the contents of... A key component of this effort involves the creation and deployment of ontologies for various application areas of interest.

WEB ONTOLOGY LANGUAGE

The meaning representation language used to represent this knowledge is the **Web Ontology Language** (OWL). OWL embodies a Description Logic that corresponds roughly to the one we've been describing here. There are now widely available tools to facilitate the creation of and reasoning with OWL-based knowledge bases().

17.7 Alternative Approaches to Meaning

The idea that the translation of linguistic inputs into a formal representation made up of discrete symbols adequately captures the notion of meaning is, not surprisingly, subject to a considerable amount of debate. The following section give brief, wholly inadequate, overviews of some of the major concerns in these debates.

17.7.1 Meaning as Action

MEANING AS ACTION

An approach that holds considerable appeal when we consider the semantics of imperative sentences is the notion of **meaning as action**. Under this view, utterances are viewed as actions, and the meanings of these utterances reside in **procedures** that are activated in the hearer as a result of hearing the utterance. This approach was followed in the creation of the historically important SHRDLU system, and is summed up well by its creator Terry Winograd (1972).

> One of the basic viewpoints underlying the model is that all language use can be thought of as a way of activating procedures within the hearer. We can think of an utterance as a program—one that indirectly causes a set of operations to be carried out within the hearer's cognitive system.

X-SCHEMA

A more recent procedural model of semantics is the **executing schema** or **x-schema** model of Bailey et al. (1997), Narayanan (1997a, 1997b), and Chang et al. (1998). The intuition of this model is that various parts of the semantics of events, including the *aspectual* factors discussed on page 32, are based on schematized descriptions of sensory-motor processes like inception, iteration, enabling, completion, force, and effort. The model represents the aspectual semantics of events via a kind of probabilistic automaton called a **Petri net** (Murata, 1989). The nets used in the model have states like *ready*, *process*, *finish*, *suspend*, and *result*.

The meaning representation of an example like *Jack is walking to the store* activates the *process* state of the walking event. An accomplishment event like

Jack walked to the store activates the *result* state. An iterative activity like *Jack walked to the store every week* is simulated in the model by an iterative activation of the *process* and *result* nodes. This idea of using sensory-motor primitives as a foundation for semantic description is also based on the work of Regier (1996) on the role of visual primitives in a computational model of learning the semantics of spatial prepositions.

17.7.2 Embodiment as the Basis for Meaning

Coming soon...

17.8 Summary

This chapter has introduced the representational approach to meaning. The following are some of the highlights of this chapter:

- A major approach to meaning in computational linguistics involves the creation of **formal meaning representations** that capture the meaning-related content of linguistic inputs. These representations are intended to bridge the gap from language to commonsense knowledge of the world.
- The frameworks that specify the syntax and semantics of these representations are called **meaning representation languages**. A wide variety of such languages are used in natural language processing and artificial intelligence.
- Such representations need to be able to support the practical computational requirements of semantic processing. Among these are the need to **determine the truth of propositions**, to support **unambiguous representations**, to represent **variables**, to support **inference**, and to be sufficiently **expressive**.
- Human languages have a wide variety of features that are used to convey meaning. Among the most important of these is the ability to convey a **predicate-argument structure**.
- **First Order Predicate Calculus** is a well-understood computationally tractable meaning representation language that offers much of what is needed in a meaning representation language.
- Important classes of meaning including **categories**, **events**, and **time** can be captured in FOPC. Propositions corresponding to such concepts as **beliefs** and **desires** require extensions to FOPC including **modal operators**.
- **Semantic networks** and **frames** can be captured within the FOPC framework.

Bibliographical and Historical Notes

NB. Not yet updated.

The earliest computational use of declarative meaning representations in natural language processing was in the context of question-answering systems (Green et al., 1961; Raphael, 1968; Lindsey, 1963). These systems employed ad-hoc representations for the facts needed to answer questions. Questions were then translated into a form that could be matched against facts in the knowledge base. Simmons (1965) provides an overview of these early efforts.

Woods (1967) investigated the use of FOPC-like representations in question answering as a replacement for the ad-hoc representations in use at the time. Woods (1973) further developed and extended these ideas in the landmark Lunar system. Interestingly, the representations used in Lunar had both a truth-conditional and a procedural semantics. Winograd (1972) employed a similar representation based on the Micro-Planner language in his SHRDLU system.

During this same period, researchers interested in the cognitive modeling of language and memory had been working with various forms of associative network representations. Masterman (1957) was probably the first to make computational use of a semantic network-like knowledge representation, although semantic networks are generally credited to Quillian (1968). A considerable amount of work in the semantic network framework was carried out during this era (Norman and Rumelhart, 1975; Schank, 1972; Wilks, 1975b, 1975a; Kintsch, 1974). It was during this period that a number of researchers began to incorporate Fillmore's notion of case roles (Fillmore, 1968) into their representations. Simmons (1973) was the earliest adopter of case roles as part of representations for natural language processing.

Detailed analyses by Woods (1975) and Brachman (1979) aimed at figuring out what semantic networks actually mean led to the development of a number of more sophisticated network-like languages including KRL (Bobrow and Winograd, 1977) and KL-ONE (Brachman and Schmolze, 1985). As these frameworks became more sophisticated and well-defined it became clear that they were restricted variants of FOPC coupled with specialized inference procedures. A useful collection of papers covering much of this work can be found in Brachman and Levesque (1985). Russell and Norvig (1995) describe a modern perspective on these representational efforts.

Linguistic efforts to assign semantic structures to natural language sentences in the generative era began with the work of Katz and Fodor (1963). The limitations of their simple feature-based representations and the natural fit of logic to many of linguistic problems of the day quickly led to the adoption of a variety of predicate-argument structures as preferred semantic representations (Lakoff, 1972; McCawley, 1968). The subsequent introduction by Montague (1973) of truth-conditional model-theoretic framework into linguistic theory led to a much tighter integration between theories of formal syntax and a wide range of formal semantic frameworks. Good introductions to Montague semantics and its role in linguistic theory can be found in Dowty et al. (1981), Partee (1976).

The representation of events as reified objects is due to Davidson (1967). The approach presented here, which explicitly reifies event participants, is due to Parsons (1990). The use of modal operators and in the representation of knowledge and belief is due to Hintikka (1969). Moore (1977) was the first to make computational use of this approach. Fauconnier (1985) deals with a wide range of issues relating to beliefs and belief spaces from a cognitive science perspective. Most current computational approaches to temporal reasoning are based on Allen's notion of temporal intervals (Allen, 1984). ter Meulen (1995) provides a modern treatment of tense and aspect. Davis (1990) describes the use of FOPC to represent knowledge across a wide range of common sense domains including quantities, space, time, and beliefs.

A recent comprehensive treatment of logic and language can be found in van Benthem and ter Meulen (1997). The classic semantics text is Lyons (1977). McCawley (1993) is an indispensable textbook covering a wide range of topics concerning logic and language. Chierchia and McConnell-Ginet (1991) also provides broad coverage of semantic issues from a linguistic perspective. Heim and Kratzer (1998) is a more recent text written from the perspective of current generative theory.

Exercises

17.1 Choose a recipe from your favorite cookbook and try to make explicit all the common-sense knowledge that would be needed to follow it.

17.2 Proponents of information retrieval occasionally claim that natural language texts in their raw form are a perfectly suitable source of knowledge for question answering. Sketch an argument against this claim.

17.3 Peruse your daily newspaper for three examples of ambiguous sentences. Describe the various sources of the ambiguities.

17.4 Consider a domain where the word *coffee* can refer to the following concepts in a knowledge-based: a caffeinated or decaffeinated beverage, ground coffee used to make either kind of beverage, and the beans themselves. Give arguments as to which of the following uses of coffee are ambiguous and which are vague.

a. I've had my coffee for today.

b. Buy some coffee on your way home.

c. Please grind some more coffee.

17.5 Encode in FOPC as much of the knowledge as you can that you came up with for Exercise 17.1

17.6 The following rule, which we gave as a translation for Example 17.24, is not a reasonable definition of what it means to be a vegetarian restaurant.

$$\forall x VegetarianRestaurant(x) \Rightarrow Serves(x, VegetarianFood)$$

Give a FOPC rule that better defines vegetarian restaurants in terms of what they serve.

17.7 Give a FOPC translations for the following sentences:

a. Vegetarians do not eat meat.

b. Not all vegetarians eat eggs.

17.8 Give a set of facts and inferences necessary to prove the following assertions:

a. McDonalds is not a vegetarian restaurant.

b. Some vegetarians can eat at McDonalds.

Don't just place these facts in your knowledge base. Show that they can be inferred from some more general facts about vegetarians and McDonalds.

17.9 Give FOPC translations for the following sentences that capture the temporal relationships between the events.

a. When Mary's flight departed, I ate lunch.

b. When Mary's flight departed, I had eaten lunch.

17.10 Give a reasonable FOPC translation of the following example.

If you're interested in baseball, the Rockies are playing tonight.

17.11 On Page 19 we gave the following FOPC translation for Example 17.21.

$$Have(Speaker, FiveDollars) \wedge \neg Have(Speaker, LotOfTime)$$

This literal representation would not be particularly useful to a restaurant-oriented question answering system. Give a deeper FOPC meaning representation for this example that is closer to what it really means.

17.12 On Page 19, we gave the following representation as a translation for the sentence *Ay Caramba is near ICSI*.

$$Near(LocationOf(AyCaramba), LocationOf(ICSI))$$

In our truth-conditional semantics, this formula is either true or false given the contents of some knowledge-base. Critique this truth-conditional approach with respect to the meaning of words like *near*.

Allen, J. (1984). Towards a general theory of action and time. *Artificial Intelligence*, *23*(2), 123–154.

Bailey, D., Feldman, J., Narayanan, S., and Lakoff, G. (1997). Modeling embodied lexical development. In *COGSCI-97*, Stanford, CA, pp. 19–24. Lawrence Erlbaum.

Bobrow, D. G. and Winograd, T. (1977). An overview of KRL, a knowledge representation language. *Cognitive Science*, *1*(1), 3–46.

Brachman, R. J. (1979). On the epistemogical status of semantic networks. In Findler, N. V. (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, pp. 3–50. Academic Press, New York.

Brachman, R. J. and Levesque, H. J. (Eds.). (1985). *Readings in Knowledge Representation*. Morgan Kaufmann, San Mateo, CA.

Brachman, R. J. and Schmolze, J. G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, *9*(2), 171–216.

Chang, N., Gildea, D., and Narayanan, S. (1998). A dynamic model of aspectual composition. In *COGSCI-98*, Madison, WI, pp. 226–231. Lawrence Erlbaum.

Chierchia, G. and McConnell-Ginet, S. (1991). *Meaning and Grammar*. MIT Press, Cambridge, MA.

Davidson, D. (1967). The logical form of action sentences. In Rescher, N. (Ed.), *The Logic of Decision and Action*. University of Pittsburgh Press.

Davis, E. (1990). *Representations of Commonsense Knowledge*. Morgan Kaufmann, San Mateo, CA.

Dowty, D. R. (1979). *Word Meaning and Montague Grammar*. D. Reidel, Dordrecht.

Dowty, D. R., Wall, R. E., and Peters, S. (1981). *Introduction to Montague Semantics*. D. Reidel, Dordrecht.

Fauconnier, G. (1985). *Mental Spaces: Aspects of Meaning Construction in Natural Language*. MIT Press, Cambridge, MA.

Fillmore, C. J. (1968). The case for case. In Bach, E. W. and Harms, R. T. (Eds.), *Universals in Linguistic Theory*, pp. 1–88. Holt, Rinehart & Winston, New York.

Green, B. F., Wolf, A. K., Chomsky, C., and Laughery, K. (1961). Baseball: An automatic question answerer. In *Proceedings of the Western Joint Computer Conference 19*, pp. 219–224. Reprinted in Grosz et al. (1986).

Heim, I. and Kratzer, A. (1998). *Semantics in a Generative Grammar*. Blackwell Publishers, Malden, MA.

Hintikka, J. (1969). Semantics for propositional attitudes. In Davis, J. W., Hockney, D. J., and Wilson, W. K. (Eds.), *Philosophical Logic*, pp. 21–45. D. Reidel, Dordrecht, Holland.

Humphries, J. J., Woodland, P. C., and Pearce, D. (1996) In *Using Accent-specific Pronunciation Modelling for Robust Speech Recognition*.

Jackendoff, R. (1983). *Semantics and Cognition*. MIT Press, Cambridge, MA.

Katz, J. J. and Fodor, J. A. (1963). The structure of a semantic theory. *Language*, *39*, 170–210.

Kintsch, W. (1974). *The Representation of Meaning in Memory*. Wiley, New York.

Lakoff, G. (1972). Linguistics and natural logic. In Davidson, D. and Harman, G. (Eds.), *Semantics for Natural Language*, pp. 545–665. D. Reidel, Dordrecht.

Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press, Chicago, IL.

Lenat, D. B. and Guha, R. V. (1991). *Building Large Knowledge-Based Systems: Representation and Inference in CYC*. Addison-Wesley, Reading, MA.

Lindsey, R. (1963). Inferential memory as the basis of machines which understand natural language. In Feigenbaum, E. and Feldman, J. (Eds.), *Computers and Thought*, pp. 217–233. McGraw Hill.

Lyons, J. (1977). *Semantics*. Cambridge University Press, New York.

Masterman, M. (1957). The thesaurus in syntax and semantics. *Mechanical Translation*, *4*(1), 1–2.

McCawley, J. D. (1968). The role of semantics in a grammar. In Bach, E. W. and Harms, R. T. (Eds.), *Universals in Linguistic Theory*, pp. 124–169. Holt, Rinehart & Winston, New York, NY.

McCawley, J. D. (1993). *Everything that Linguists have Always Wanted to Know about Logic* (2nd edition). University of Chicago Press, Chicago, IL.

McDermott, D. (1976). Artificial intelligence meets natural stupidity. *SIGART Newsletter*, *57*.

Montague, R. (1973). The proper treatment of quantification in ordinary English. In Thomason, R. (Ed.), *Formal Philosophy: Selected Papers of Richard Montague*, pp. 247–270. Yale University Press, New Haven, CT.

Moore, R. (1977). Reasoning about knowledge and action. In *IJCAI-77*, pp. 223–227.

Murata, T. (1989). Petri nets: Properties, analysis, and applications. *Proceedings of the IEEE*, *77*(4), 541–576.

Narayanan, S. (1997a). *Knowledge-based Action Representations for Metaphor and Aspect (KARMA)*. Ph.D. thesis, University of California, Berkeley.

Narayanan, S. (1997b). Talking the talk *is* like walking the walk: A computational model of verbal aspect. In *COGSCI-97*, Stanford, CA, pp. 548–553.

Norman, D. A. and Rumelhart, D. E. (1975). *Explorations in Cognition*. Freeman, San Francisco, CA.

Parsons, T. (1990). *Events in the Semantics of English*. MIT Press, Cambridge, MA.

Partee, B. H. (Ed.). (1976). *Montague Grammar*. Academic Press, New York.

Quillian, M. R. (1968). Semantic memory. In Minsky, M. (Ed.), *Semantic Information Processing*, pp. 227–270. MIT Press, Cambridge, MA.

Raphael, B. (1968). SIR: A computer program for semantic information retrieval. In Minsky, M. (Ed.), *Semantic Information Processing*, pp. 33–145. MIT Press.

Regier, T. (1996). *The Human Semantic Potential*. MIT Press, Cambridge, MA.

Reichenbach, H. (1947). *Elements of Symbolic Logic*. Macmillan, New York.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ.

Schank, R. C. (1972). Conceptual dependency: A theory of natural language processing. *Cognitive Psychology*, *3*, 552–631.

Simmons, R. F. (1965). Answering English questions by computer: A survey. *Communications of the ACM*, *8*(1), 53–70.

Simmons, R. F. (1973). Semantic networks: Their computation and use for understanding English sentences. In Schank, R. C. and Colby, K. M. (Eds.), *Computer Models of Thought and Language*, pp. 61–113. W.H. Freeman and Co., San Francisco.

ter Meulen, A. (1995). *Representing Time in Natural Language*. MIT Press, Cambridge, MA.

van Benthem, J. and ter Meulen, A. (Eds.). (1997). *Handbook of Logic and Language*. MIT Press, Cambridge, MA.

Vendler, Z. (1967). *Linguistics in Philosophy*. Cornell University Press, Ithaca, NY.

Wilks, Y. (1975a). Preference semantics. In Keenan, E. L. (Ed.), *The Formal Semantics of Natural Language*, pp. 329–350. Cambridge Univ. Press, Cambridge.

Wilks, Y. (1975b). A preferential, pattern-seeking, semantics for natural language inference. *Artificial Intelligence*, *6*(1), 53–74.

Winograd, T. (1972). *Understanding Natural Language*. Academic Press, New York.

Woods, W. A. (1967). *Semantics for a Question-Answering System*. Ph.D. thesis, Harvard University.

Woods, W. A. (1973). Progress in natural language understanding. In *Proceedings of AFIPS National Conference*, pp. 441–450.

Woods, W. A. (1975). What's in a link: Foundations for semantic networks. In Bobrow, D. G. and Collins, A. M. (Eds.), *Representation and Understanding: Studies in Cognitive Science*, pp. 35–82. Academic Press, New York.

Zwicky, A. and Sadock, J. (1975). Ambiguity tests and how to fail them. In Kimball, J. (Ed.), *Syntax and Semantics 4*, pp. 1–36. Academic Press, New York.

18 COMPUTATIONAL SEMANTICS

> *"Then you should say what you mean," the March Hare went on.*
> *"I do," Alice hastily replied; "at least–at least I mean what I say–that's the same thing, you know."*
> *"Not the same thing a bit!" said the Hatter. "You might just as well say that 'I see what I eat' is the same thing as 'I eat what I see'!"*
>
> Lewis Carroll, *Alice in Wonderland*

SEMANTIC ANALYSIS

This chapter presents a principled computational approach to the problem of **semantic analysis**, the process whereby meaning representations of the kind discussed in the last chapter are composed and associated with linguistic expressions. The automated creation of accurate and expressive meaning representations necessarily involves a wide range of knowledge-sources and inference techniques. Among the sources of knowledge that are typically involved are the meanings of words, the conventional meanings associated with grammatical constructions, knowledge about the structure of the discourse, common-sense knowledge about the topic at hand and knowledge about the state of affairs in which the discourse is occurring.

SYNTAX-DRIVEN SEMANTIC ANALYSIS

The focus of this chapter is a kind of **syntax-driven semantic analysis** that is fairly modest in its scope. In this approach, meaning representations are assigned to sentences based solely on knowledge gleaned from the lexicon and the grammar. When we refer to an expression's meaning, or meaning representation, we have in mind a representation that is both context independent and free of inference. Representations of this type correspond to the traditional notion of literal meaning discussed in the last chapter.

There are two motivations for proceeding along these lines: there are application domains, including question answering, where such primitive representations are sufficient to produce useful results, and these impoverished representations can serve as useful inputs to subsequent processes that can produce richer, more com-

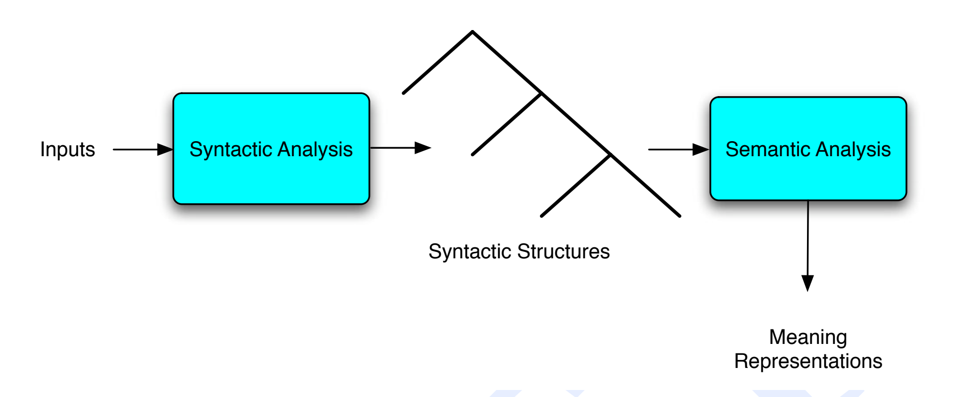


Figure 18.1 A simple pipeline approach to semantic analysis.

plete, meaning representations. Chs. 21 and 24 will discuss how these meaning representations can be used in processing extended discourses and dialogs.

18.1 Syntax-Driven Semantic Analysis

PRINCIPLE OF COMPOSITIONALITY

The approach detailed in this section is based on the **principle of compositionality**. The key idea behind this approach is that the meaning of a sentence can be constructed from the meanings of its parts. When interpreted superficially this principle is somewhat less than useful. We know that sentences are composed of words, and that words are the primary carriers of meaning in language. It would seem then that all this principle tells us is that we should compose the meaning representation for sentences from the meanings of the words that make them up.

Fortunately, the Mad Hatter has provided us with a hint as to how to make this principle useful. The meaning of a sentence is not based solely on the words that make it up, but also on the ordering and grouping of words, and on the relations among the words in the sentence. Of course, this is simply another way of saying that the meaning of a sentence is partially based on its syntactic structure. Therefore, in syntax-driven semantic analysis, the composition of meaning representations is guided by the syntactic *components* and *relations* provided by the kind of grammars discussed in Ch. 12.

SEMANTIC ANALYZER

Let's begin by assuming that the syntactic analysis of an input sentence serves as the input to a semantic analyzer. Figure 18.1 illustrates an obvious pipeline-oriented approach that follows directly from this assumption. An input is first passed through a parser to derive its syntactic analysis. This analysis is then passed as input to a **semantic analyzer** to produce a meaning representation. Note that although this diagram shows a parse tree as input, other syntactic representations such as flat chunks, feature structures, or dependency structures can also be used. For the remainder of this chapter we'll assume tree-like inputs.

Before moving on, we should touch on the role of ambiguity in this story. As we've seen, ambiguous representations can arise from numerous sources including competing syntactic analyses, ambiguous lexical items, competing anaphoric references and as we'll see later in this chapter ambiguous quantifier scopes. In the syntax-driven approach presented here, we assume that syntactic, lexical and anaphoric ambiguities are not a problem. That is, we'll assume that some larger system is capable of iterating through the possible ambiguous interpretations and passing them individually to the kind of semantic analyzer described here.

Let's consider how such an analysis might proceed with the following example:

(18.1) Franco likes Frasca.

Fig. 18.1 shows a simplified parse tree (lacking any feature attachments), along with a plausible meaning representation for this example. As suggested by the dashed arrows, a semantic analyzer given this tree as input might fruitfully proceed by first retrieving a skeletal meaning representation from the subtree corresponding to the verb *likes*. The analyzer would then retrieve or compose meaning representations corresponding to the two noun phrases in the sentence. Then using the representation acquired from the verb as a kind of template, the noun phrase meaning representations would be used to bind the appropriate variables in the verb representation, thus producing the meaning representation for the sentence as a whole.

Unfortunately, there are a number of serious difficulties with this simplified story. As described, the function used to interpret the tree in Fig. 18.1 must know, among other things, that it is the verb that carries the template upon which the final representation is based, where its corresponding arguments are and which argument fills which role in the verb's meaning representation. In other words, it requires a good deal of specific knowledge about *this particular example and its parse tree* to create the required meaning representation. Given that there are an infinite number of such trees for any reasonable grammar, any approach based on one semantic function for every possible tree is in serious trouble.

Fortunately, we have faced this problem before. Languages are not defined by enumerating the strings or trees that are permitted, but rather by specifying finite devices that are capable of generating the desired set of outputs. It would seem, therefore, that the right place for semantic knowledge in a syntax-directed approach is with the finite set of devices that are used to generate trees in the first place: the grammar rules and the lexical entries. This is known as the **rule-to-rule**

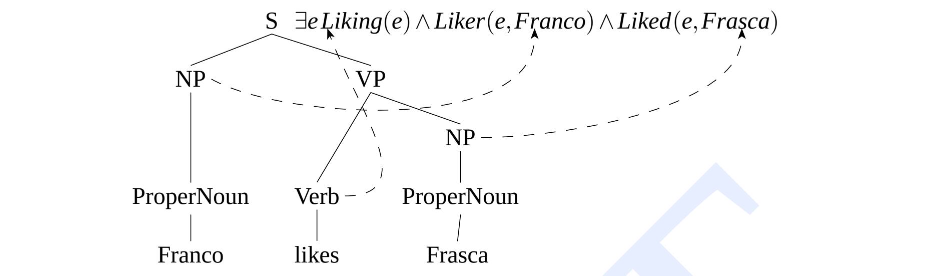


Figure 18.2 Parse tree for the sentence *Franco likes Frasca.*

RULE-TO-RULE HYPOTHESIS

hypothesis (Bach, 1976).

Designing an analyzer based on this approach brings us back to the notion of parts and what it means for them to have meanings. The following section is an attempt to answer the following two questions:

- What does it mean for a syntactic constituent to have a meaning?
- What do these meanings have to be like so that they can be composed into larger meanings?

18.2 Semantic Augmentations to Context-Free Grammar Rules

SEMANTIC ATTACHMENTS

In keeping with the approach used in Ch. 16, we will begin by augmenting our context-free grammar rules with **semantic attachments**. These attachments are instructions that specify how to compute the meaning representation of a construction from the meanings of its constituent parts. Abstractly, our augmented rules have the following structure:

$$A \rightarrow \alpha_1 \ldots \alpha_n \qquad \{f(\alpha_j.sem, \ldots, \alpha_k.sem)\}$$

The semantic attachment to the basic context-free rule is shown in the $\{\ldots\}$ to the right of the rule's syntactic constituents. This notation states that the meaning representation assigned to the construction A, which we will denote as $A.sem$, can be computed by running the function f on some subset of the semantic attachments of A's constituents.

There are myriad ways to instantiate this style of rule-to-rule approach. Our semantic attachments could, for example, take the form of arbitrary programming language fragments. A meaning representation for a given derivation could then be constructed by passing the appropriate fragments to an interpreter in a bottom-up fashion and then storing the resulting representations as the value for the associated non-terminals.[1] Such an approach would allow us to create any meaning representation we might like. Unfortunately, the unrestricted power of this approach would also allow us to create representations that have no correspondence at all with the kind of formal logical expressions described in the last chapter. Moreover, this approach would provide us with very little guidance as to how to go about designing the semantic attachments to our grammar rules.

For these reasons, more principled approaches are typically used to instantiate the rule-to-rule approach. We'll introduce two such constrained approaches in this chapter. The first makes direct use of FOL and the λ-calculus notation introduced in Ch. 17. This approach essentially uses a logical notation to guide the creation of logical forms in a principled fashion. The second approach, described later in Sec. 18.4 is based on the feature-structure and unification formalisms introduced in Ch. 16.

To get started, let's take a look at a very basic example along with a simplified target semantic representation.

(18.2) Maharani closed.

$Closed(Maharani)$

Let's work our way bottom-up through the rules involved in this example's derivation. Starting with the proper noun, the simplest possible approach is to assign a unique FOL constant to it, as in the following.

$$ProperNoun \rightarrow Maharani \qquad \{Maharani\}$$

The non-branching *NP* rule that dominates this one doesn't add anything semantically, so we'll just copy the semantics of the *ProperNoun* up unchanged to the NP.

$$NP \rightarrow ProperNoun \qquad \{ProperNoun.sem\}$$

Moving on to the *VP*, the semantic attachment for the verb needs to provide the name of the predicate, specify its arity and provide the means to incorporate an argument once it's discovered. We'll make use of a λ-expression to accomplish these tasks.

$$VP \rightarrow Verb \qquad \{Verb.sem\}$$

$$Verb \rightarrow closed \qquad \{\lambda x.Closed(x)\}$$

This attachment stipulates that the verb *closed* has a unary predicate *Closed* as its representation. The λ-notation gives us the means to leave unspecified, as the x variable, the entity that is closing. As with our earlier *NP* rule, the intransitive *VP* rule that dominates the verb simply copies upward the semantics of the verb below it.

[1] Those familiar with the UNIX compiler tools YACC and Bison will recognize this approach.

Proceeding upward, it remains for the semantic attachment for the *S* rule to bring things together by inserting the semantic representation of the subject *NP* as the first argument to the predicate.

$$S \rightarrow NP\ VP \qquad \{VP.sem(NP.sem)\}$$

Since the value of *VP.sem* is a λ-expression and the value of *NP.sem* is a simply a FOL constant, we can create our desired final meaning representation by using λ-reduction to apply the *VP.sem* to the *NP.sem*.

$$\lambda x.Closed(x)(Maharani) \Longrightarrow Closed(Maharani)$$

This example illustrates a general pattern which will repeat itself throughout this chapter. The semantic attachments to our grammar rules will consist primarily of λ-reductions, where one element of an attachment serves as a functor and the rest serve as arguments to it. As we'll see, the real work resides in the lexicon where the bulk of the meaning representations are introduced.

Although this example illustrates the basic approach, the full story is a bit more complex. Let's begin by replacing our earlier target representation with one that is more in keeping with the event-oriented representations introduced in the last chapter, and by considering an example with a more complex noun phrase as its subject.

(18.3) Every restaurant closed.

The target representation for this example should be the following.

$$\forall x Restaurant(x) \Rightarrow (\exists e Closing(e) \wedge ClosedThing(e,x)$$

Clearly, the semantic contribution of the subject noun phrase in this example is much more extensive than in our previous one. In our earlier example, the FOL constant representing the subject was simply plugged into the correct place in *Closed* predicate via a single λ-reduction. Here the final result involves a complex intertwining of the content provided by the *NP* and the content provided by the *VP*. We'll have to do some work if we want rely on λ-reduction to produce what we want here.

RESTRICTION

The first step is to determine exactly what we'd like the meaning representation of *Every restaurant* to be. Let's start by assuming that *Every* invokes the $\forall$ quantifier and that *restaurant* specifies the category of concept that we're quantifying over, which we'll call the **restriction** of the noun phrase. Putting these together we might expect the meaning representation to be something like $\forall x Restaurant(x)$. Although this is a valid FOL formula its not a terribly useful one, since it says that everything is a restaurant. What's missing from it is the notion that noun phrases like *every restaurant* are normally embedded in expressions that stipulate something about the universally quantified variable. That is, we're probably trying to *say something* about all restaurants. This notion is traditionally referred to as the *NP*'s **nuclear scope**. In this case, the nuclear scope of this noun phrase is *closed*.

NUCLEAR SCOPE

We can capture these notions in our target representation by adding a dummy predicate, Q, representing the scope and attaching that predicate to the restriction predicate with an $\Rightarrow$ logical connective, leaving us with the following expression:

$$\forall x Restaurant(x) \Rightarrow Q(x)$$

Ultimately, what we need to do to make this expression meaningful is to replace Q with the logical expression corresponding to the nuclear scope. Fortunately, the λ-calculus can come to our rescue again. All we need to do is to permit λ-variables to range over FOL predicates as well as terms. The following expression captures exactly what we need.

$$\lambda Q.\forall x Restaurant(x) \Rightarrow Q(x)$$

The following series of grammar rules with their semantic attachments serve to produce this desired meaning representation for this kind of *NP*.

$$NP \rightarrow Det\ Nominal \qquad \{Det.Sem(Nominal.Sem)\}$$

$$Det \rightarrow every \qquad \{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$$

$$Nominal \rightarrow Noun \qquad \{Noun.sem\}$$

$$Noun \rightarrow restaurant \qquad \{\lambda x Restaurant(x)\}$$

The critical step in this sequence involves the λ-reduction in the *NP* rule. This rule applies the λ-expression attached to the *Det* to the semantic attachment of the *Nominal*, which is itself a λ-expression. The following are the intermediate steps in this process.

$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda x.Restaurant(x))$$

$$\lambda Q.\forall x \lambda x.Restaurant(x)(x) \Rightarrow Q(x)$$

$$\lambda Q.\forall x\ Restaurant(x) \Rightarrow Q(x)$$

The first expression is the expansion of the $Det.Sem(Nominal.Sem)$ semantic attachment to the *NP* rule. The second formula is the result of this λ-reduction. Note that this second formula has a λ-application embedded in it. Reducing this expression in place gives us the final form.

Having revised our semantic attachment for the subject noun phrase portion of our example, let's move to the *S* and *VP* and *Verb* rules to see how they need to change to accommodate these revisions. Let's start with the *S* rule and work our way down. Since the meaning of the subject *NP* is now a λ-expression, it makes sense to consider it as a functor to be called with the meaning of the *VP* as its argument. The following attachment accomplishes this.

$$S \rightarrow NP\ VP \qquad \{NP.sem(VP.sem)\}$$

Note that we've flipped the role of functor and argument from our original proposal for this *S* rule.

The last attachment to revisit is the one for the verb *close*. We need to update it to provide a proper event-oriented representation and to make sure that it is interfaces well with the new *S* and *NP* rules. The following attachment accomplishes both goals.

$$Verb \rightarrow close \qquad \{\lambda x.\exists e Closed(e) \wedge Closed(e,x)\}$$

This attachment is passed unchanged to the *VP* constituent via the intransitive *VP* rule. It is then combined with the meaning representation of *Every restaurant* as dictated by the semantic attachment for the *S* given earlier. The following expressions illustrate the intermediate steps in this process.

$$\lambda Q.\forall x Restaurant(x) \Rightarrow Q(x)(\lambda y.\exists e Closed(e) \wedge Closed(e,y))$$

$$\forall x Restaurant(x) \Rightarrow \lambda y.\exists e Closed(e) \wedge Closed(e,y)(x)$$

$$\forall x Restaurant(x) \Rightarrow \exists e Closed(e) \wedge Closed(e,x)$$

These steps achieve our goal of getting the *VP*'s meaning representation spliced in as the nuclear scope in the *NP*'s representation.

As is always the case with any kind of grammar engineering effort we now need to make sure that our earlier simpler examples still work. One area that we need to revisit is our representation of proper nouns. Let's consider them in the context of our earlier example.

(18.4) Maharani closed.

The *S* rule now expects the subject *NP*'s semantic attachment to be a functor applied to the semantics of the *VP*, therefore our earlier representation of proper nouns as FOL constants won't do. Fortunately, we can once again exploit the flexibility of the λ-calculus to accomplish what we need with the following expression.

$$\lambda x.x(Maharani)$$

This trick turns a simple FOL constant into a *lambda*-expression, which when reduced serves to inject the constant into a larger expression. You should work through our original example with all of the new semantic rules to make sure that you can come up with the following intended representation:

$$\exists e Closing(e) \wedge closed(Maharani)$$

As one final exercise, let's see how this approach extends to an expression involving a transitive verb phrase, as in the following.

(18.5) Matthew opened a restaurant.

If we've done things correctly we ought to be able to specify the semantic attachments for transitive verb phrases, for the verb *open* and for the determiner *a*, while leaving the rest of our rules alone.

Let's start by modeling the semantics for the determiner *a* on our earlier attachment for *every*.

$$Det \rightarrow a \qquad \{\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)\}$$

This rule differs from the attachment for *every* in two ways. First we're using the existential quantifier $\exists$ to capture the semantics of *a*. And second we've replaced the $\Rightarrow$ operator with a logical $\wedge$. The overall framework remains the same with the λ-variables P and Q standing in for the restriction and nuclear scopes to be filled in later. With this addition our existing *NP* rule will create the appropriate representation for *a restaurant*:

$$\lambda Q \exists x Restaurant(x) \wedge Q(x)$$

Next let's move on to the *Verb* and *VP* rules. There are two arguments that need to be incorporated into the underlying meaning representation. One argument is available at the level of the transitive *VP* rule, and the second at the *S* rule. Let's assume the following form for the *VP* semantic attachment.

$$VP \rightarrow Verb\ NP \qquad \{Verb.Sem(NP.Sem)\}$$

This attachment assumes that the verb's semantic attachment will be applied as a functor to the semantics of its noun phrase argument. And let's assume for now that the representations we developed earlier for quantified noun phrases and proper nouns will remain unchanged. With these assumptions in mind, the following attachment for the verb *opened* will do what we want.

$$Verb \rightarrow opened\{\lambda w.\lambda z.w(\lambda x \exists e Opening(e) \wedge Opener(e,z) \wedge Opened(e,x))\}$$

With this attachment in place, the transitive *VP* rule will incorporate the variable standing for *a restaurant* as the second argument to *opened*, incorporate the entire expression representing the *opening* event as the nuclear scope of *a restaurant* and finally produce a λ-expression suitable for use with our *S* rule. As with the previous example you should walk through this example step by step to make sure that you arrive at our intended meaning representation.

$$\exists x Restaurant(x) \wedge \exists e Opening(e) \wedge Opener(e,Matthew) \wedge Opened(e,x)$$

The list of semantic attachments which we've developed for this small grammar fragment is shown in Fig. 18.2. Sec. 18.5 expands the coverage of this fragment to some of the more important constructions in English.

In walking through these examples, we have introduced three techniques that instantiate the rule-to-rule approach to semantic analysis introduced at the beginning of this section:

Grammar Rule	Semantic Attachment
$S \rightarrow NP\ VP$	$\{NP.sem(VP.sem)\}$
$NP \rightarrow Det\ Nominal$	$\{Det.sem(Nominal.sem)\}$
$NP \rightarrow ProperNoun$	$\{ProperNoun.sem\}$
$Nominal \rightarrow Noun$	$\{Noun.sem\}$
$VP \rightarrow Verb$	$\{Verb.sem\}$
$VP \rightarrow Verb\ NP$	$\{Verb.sem(NP.sem)\}$
$Det \rightarrow every$	$\{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$
$Det \rightarrow a$	$\{\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)\}$
$Noun \rightarrow restaurant$	$\{\lambda r.Restaurant(r)\}$
$ProperNoun \rightarrow Matthew$	$\{\lambda m.m(Matthew)\}$
$ProperNoun \rightarrow Franco$	$\{\lambda f.f(Franco)\}$
$ProperNoun \rightarrow Franco$	$\{\lambda f.f(Frasca)\}$
$Verb \rightarrow closed$	$\{\lambda x.\exists e Closing(e) \wedge Closed(e,x)\}$
$Verb \rightarrow opened$	$\{\lambda w.\lambda z.w(\lambda x.\exists e Opening(e) \wedge Opener(e,z)$ $\wedge Opened(e,x))$

Figure 18.3 Semantic attachments for a fragment of our English grammar and lexicon.

1. Associating complex, function-like, λ-expressions with lexical items
2. Copying of semantic values from children to parents in non-branching rules
3. Function-like application of the semantics of one of the children of a rule to the semantics of the other children of the rule via λ-reduction.

These techniques serve to illustrate a general division of labor that guides the design of semantic attachments in this compositional framework. In general, it is the lexical rules that introduce quantifiers, predicates and terms into our meaning representations. The semantic attachments for grammar rules put these elements together in the right ways, but do not in general introduce new elements into the representations being created.

18.3 Quantifier Scope Ambiguity and Underspecification

The grammar fragment developed in the last section appears to be sufficient to handle examples like the following that contain two or more quantified noun phrases.

(18.6) Every restaurant has a menu.

Systematically applying the rules given in Fig. 18.2 to this example produces the following perfectly reasonable meaning representation.

$$\begin{aligned}&\forall x\, Restaurant(x) \Rightarrow\\ &\qquad \exists y\, Menu(y) \wedge \exists e Having(e) \wedge Haver(e,x) \wedge Had(e,y)\end{aligned}$$

This formula more or less corresponds to the common sense notion that all restaurants have menus.

Unfortunately, this isn't the only possible interpretation for this example. The following is also possible.

$$\begin{aligned}&\exists y\, Menu(y) \wedge \forall x\, Restaurant(x) \Rightarrow\\ &\qquad \exists e\, Having(e) \wedge Haver(e,x) \wedge Had(e,y)\end{aligned}$$

This formula asserts that there is one menu out there in the world and all restaurants share it. Now from a common sense point of view this seems pretty unlikely, but remember that our semantic analyzer only has access to the semantic attachments in the grammar and the lexicon in producing meaning representations. Of course, world knowledge and contextual information can be used to select between these two readings, but only if we are able to produce both.

This example illustrates that expressions containing quantified terms can give rise to ambiguous representations even in the absence of syntactic, lexical or anaphoric ambiguities. This is generally known as the problem of **quantifier scoping**. The difference between the two interpretations given above arises from which of the two quantified variables has the outer scope.

QUANTIFIER SCOPING

The approach outlined in the last section can not handle this phenomena. To fix this we'll need the following capabilities.

- The ability to efficiently create *underspecified* representations that embody all possible readings without explicitly enumerating them
- A means to generate, or extract, all of the possible readings from this representation
- And the ability to choose among the possible readings

The following sections will outline approaches to the first two problems. The solution to the last, most important problem, requires the use of context and world knowledge and unfortunately remains a largely unsolved problem.

18.3.1 Store and Retrieve Approaches

One way to address the quantifier scope problem is to add a new notation to our existing semantic attachments to facilitate the compositional creation of the desired meaning representations. In this case, we'll introduce the notion of a **complex-term** that permits FOL expressions like $\forall x\, Restaurant(x)$ to appear in places where

COMPLEX-TERM

we would normally only allow FOL terms to appear. Formally, a complex-term will be an expression with the following three-part structure:

$$\langle Quantifier\ variable\ formula \rangle$$

Applying this notation to our current example, we would arrive at the following representation:

$$\begin{aligned}&\exists e\ Having(e)\\ &\quad \wedge Haver(e, \langle \forall x\ Restaurant(x) \rangle)\\ &\quad \wedge Had(e, \langle \exists y\ Menu(y) \rangle)\end{aligned}$$

The intent of the this approach is to capture the basic predicate argument structure of an expression, while remaining agnostic about where the various quantifiers will end up in the final representation.

As was the case with λ-expressions, this notational device is only useful if we can provide an algorithm to convert it back into an ordinary FOL expression. This can be accomplished by rewriting any predicate containing a complex-term according to the following schema:

$$\begin{aligned}&P(\langle Quantifier\ variable\ formula \rangle)\\ &\Longrightarrow\\ &Quantifier\ variable\ formula\ Connective\ P(variable)\end{aligned}$$

In other words, the complex-term:

1. is extracted from the predicate in which it appears,
2. is replaced by the specified variable,
3. and has its variable, quantifier, and formula prepended to the new expression through the use of an appropriate connective.

The connective that is used to attach the extracted formula to the front of the new expression depends on the type of the quantifier being used: $\wedge$ is used with $\exists$, and $\Rightarrow$ is used with $\forall$.

How does this scheme help with our ambiguity problem? Note that our new representation contains two complex terms. The order in which we process them determines which of the two readings we end up with. Let's consider the case where we proceed left-to-right through the expression transforming the complex terms as we find them. In this case, we encounter *Every restaurant* first; transforming it yields the following expression.

$$\forall x Restaurant(x) \Rightarrow \exists e\ Having(e) \wedge Haver(e,x) \wedge Had(e, \langle \exists y Menu(y) \rangle)$$

Proceeding onward we next encounter *a menu*. Transforming this complex term yields the following final form which corresponds to the non-intuitive reading that we couldn't get with our earlier method.

$$\exists y Menu(y) \wedge \forall x Restaurant(x) \Rightarrow \exists e\ Having(e) \wedge Haver(e,x) \wedge Had(e,y)$$

To get the more common-sense reading that we had earlier all we have to is pull out the complex-terms in the other order; first *a menu* and then *every restaurant*.

This approach to quantifier scope provides solutions to the two of the desiderata given earlier: complex terms provide a compact underspecified representation of all the possible quantifier-based ambiguous readings, and the method for transforming them provides a deterministic method for eliminating complex terms and thus retrieving valid FOL formulas. And by altering the ordering by which complex terms are eliminated we can recover all the possible readings. Of course, sentences with N quantifiers will have $O(N!)$ different quantifier-based readings.

In practice, most systems employ an ad hoc set of heuristic preference rules that can be used to generate preferred forms in order of their overall likelihood. In cases where no preference rules apply, a left-to-right quantifier ordering that mirrors the surface order of the quantifiers is used. Domain specific knowledge can then be used to either accept a quantified formula, or reject it and request another formula. Alshawi (1992) presents a comprehensive approach to generating plausible quantifier scopings.

18.3.2 Constraint-Based Approaches

NEXT DRAFT HOLE SEMANTICS

18.4 Unification-Based Approaches to Semantic Analysis

As mentioned in Sec. 18.2, feature structures and the unification operator provide an effective way to implement syntax-driven semantic analysis. Recall that in Ch. 16 we paired complex feature structures with individual context-free grammar rules to encode syntactic constraints such as number agreement and subcategorization; constraints that were awkward or in some cases impossible to convey directly using context-free grammars. For example, the following rule was used to capture agreement constraints on English noun phrases.

$$\begin{aligned} &NP \rightarrow Det\ Nominal \\ &\quad \langle Det\ \text{AGREEMENT}\rangle = \langle Nominal\ \text{AGREEMENT}\rangle \\ &\quad \langle NP\ \text{AGREEMENT}\rangle = \langle Nominal\ \text{AGREEMENT}\rangle \end{aligned}$$

Rules such as this one serve two functions at the same time: they insure that the grammar rejects expressions that violate this constraint, and more importantly for our current topic, they create complex structures that can be associated with parts of grammatical derivations. The following structure, for example, results from the application of the above rule to a singular noun phrase.

$$\begin{bmatrix}\text{AGREEMENT} & \begin{bmatrix}\text{NUMBER} & \text{SG}\end{bmatrix}\end{bmatrix}$$

We'll use this latter capability to compose meaning representations and associate them with constituents in parse.

In this unification-based approach, our FOL representations and λ-based semantic attachments are replaced by complex feature structures and unification equations. To see how this works, let's walk through a series of examples similar to those discussed earlier in Sec. 18.2. Let's start with a simple intransitive sentence with a proper noun as it's subject.

(18.7) Rhumba closed

Using an event-oriented approach, the meaning representation for this sentence should be something like the following.

$$\exists e\, Closing(e) \wedge Closed(e, Rhumba)$$

Our first task will be to show that we can encode representations like this within the feature structure framework. The most straightforward way to approach this task is to simply follow the BNF-style definition of FOL statements given in Ch. 17. The relevant elements of this definition stipulate that FOL formulas come in three varieties: atomic formulas consisting of predicates with the appropriate number of term arguments, formulas conjoined with other formulas via the $\wedge$, $\vee$ and $\Rightarrow$ operators, and finally quantified formulas which consist of a quantifier, variables and a formula. Using this definition as a guide, we can capture this FOL expression with the following feature structure.

$$\begin{bmatrix}
\text{QUANT} & \exists \\
\text{VAR} & \boxed{1} \\
\text{FORMULA} & \begin{bmatrix}
\text{OP} & \text{AND} \\
\text{FORMULA1} & \begin{bmatrix}\text{PRED} & \text{CLOSING} \\ \text{ARG0} & \boxed{1}\end{bmatrix} \\
\text{FORMULA2} & \begin{bmatrix}\text{PRED} & \text{CLOSED} \\ \text{ARG0} & \boxed{1} \\ \text{ARG1} & \text{RHUMBA}\end{bmatrix}
\end{bmatrix}
\end{bmatrix}$$

Fig. 18.4 shows this expression using the DAG-style notation introduced in Ch. 16. This figure reveals the way that variables are handled. Instead of introducing explicit FOL variables, we'll use the path-based feature-sharing capability of feature structures to accomplish the same goal. In this example, the event variable *e* is captured by the three paths leading to the same shared node.

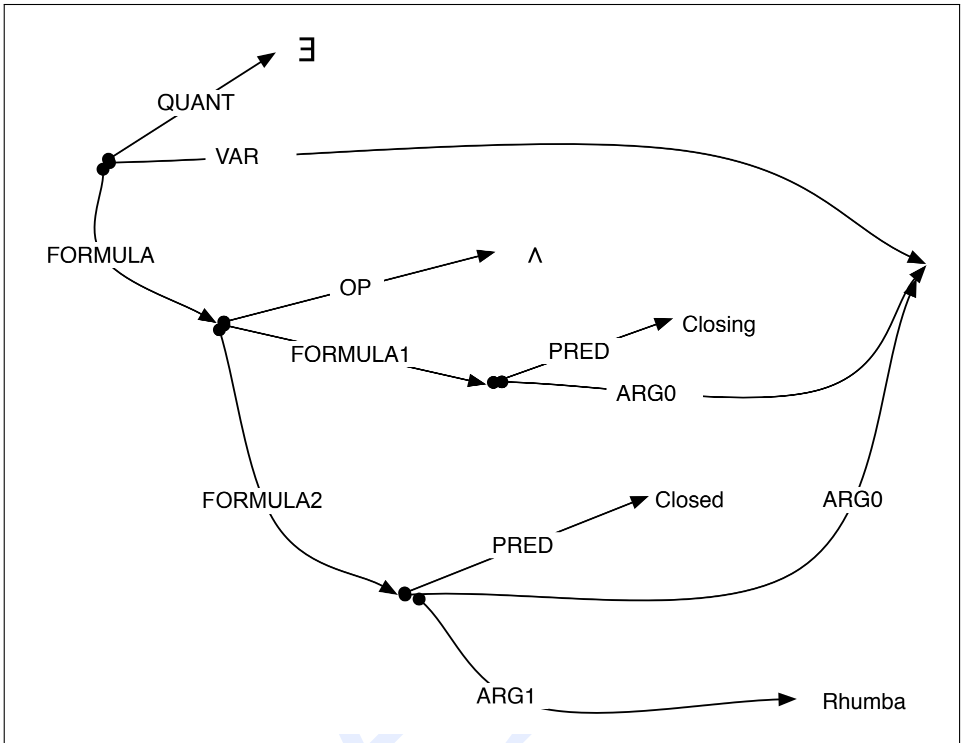


Figure 18.4 A directed graph notation for semantic feature structures.

Our next step is to associate unification equations with the grammar rules involved in this example's derivation. Let's start at the top with the *S* rule.

$$
\begin{aligned}
&S \rightarrow NP\ VP \\
&\quad \langle S\ \textsc{sem}\rangle = \langle NP\ \textsc{sem}\rangle \\
&\quad \langle VP\ \textsc{arg}0\rangle = \langle NP\ \textsc{indexvar}\rangle \\
&\quad \langle NP\ \textsc{scope}\rangle = \langle VP\ \textsc{sem}\rangle
\end{aligned}
$$

The first line simply equates the meaning representation of the *NP* (encoded under the SEM feature) with our top-level *S*. The purpose of the second equation is to assign the subject *NP* to the appropriate role inside the *VP*'s meaning representation. More concretely, it fills the appropriate role in the *VP*'s semantic representation by unifying the ARG0 feature with a path that leads to a representation of the semantics of the *NP*. Finally, it unifies the SCOPE feature in the *NP*'s meaning representation with a pointer to the VP's meaning representation. As we'll see, this is a somewhat convoluted way to bring the representation of an event up to where it belongs in the representation. The motivation for this apparatus should become clear in the ensuing discussion where we consider quantified noun phrases.

Carrying on, let's consider the attachments for the *NP* and *ProperNoun* parts of this derivation.

$\mathit{NP} \rightarrow \mathit{ProperNoun}$

$\langle \mathit{NP}\ \textsc{sem}\rangle = \langle \mathit{ProperNoun}\ \textsc{sem}\rangle$

$\langle \mathit{NP}\ \textsc{scope}\rangle = \langle \text{ProperNoun}\ \textsc{scope}\rangle$

$\langle \mathit{NP}\ \textsc{indexvar}\rangle = \langle \mathit{ProperNoun}\ \textsc{indexvar}\rangle$

$\mathit{ProperNoun} \rightarrow \mathit{Rhumba}$

$\langle \mathit{ProperNoun}\ \textsc{sem pred}\rangle = \textsc{rhumba}$

$\langle \mathit{ProperNoun}\ \textsc{indexvar}\rangle = \langle \mathit{ProperNoun}\ \textsc{sem pred}\rangle$

As we saw earlier, there isn't much to the semantics of proper nouns in this approach. Here we're just introducing a constant and providing an index variable to point at that constant.

Next, let's move on to the semantic attachments for the *VP* and *Verb* rules.

$\mathit{VP} \rightarrow \mathit{Verb}$

$\langle \mathit{VP}\ \textsc{sem}\rangle = \langle\ \text{Verb}\ \textsc{sem}\rangle$

$\langle \mathit{VP}\ \textsc{arg}0\rangle = \langle\ \text{Verb}\ \textsc{arg}0\rangle$

$\mathit{Verb} \rightarrow \mathit{closed}$

$\langle \mathit{Verb}\ \textsc{sem quant}\rangle = \exists$

$\langle \mathit{Verb}\ \textsc{sem formula op}\rangle = \wedge$

$\langle \mathit{Verb}\ \textsc{sem formula formula}1\ \textsc{pred}\rangle = \textsc{closing}$

$\langle \mathit{Verb}\ \textsc{sem formula formula}1\ \textsc{arg}0\rangle = \langle \mathit{Verb}\ \textsc{sem var}\rangle$

$\langle \mathit{Verb}\ \textsc{sem formula formula}2\ \textsc{pred}\rangle = \textsc{closed}$

$\langle \mathit{Verb}\ \textsc{sem formula formula}2\ \textsc{arg}0\rangle = \langle \mathit{Verb}\ \textsc{sem var}\rangle$

$\langle \mathit{Verb}\ \textsc{sem formula formula}2\ \textsc{arg}1\rangle = \langle \mathit{Verb}\ \textsc{arg}0\rangle$

The attachments for the *VP* rule parallel our earlier treatment of non-branching grammatical rules. These unification equations are simply making the appropriate semantic fragments of the *Verb* available at the *VP* level. In contrast, the unification equations for the *Verb* introduce the bulk of the event representation that is at the core of this example. Specifically, it introduces the quantifier, event variable and predications that make up the body of the final expression. What would be an event variable in FOL is captured by the equations unifying the *Verb* SEM VAR path with the appropriate arguments to the predicates in the body of the formula. Finally, it exposes the single missing argument (the entity being closed) through the ⟨ *Verb* ARG0⟩ equation.

Taking a step back we can see that these equations serve the same basic functions as the λ-expressions in Sec. 18.2; they provide the content of the FOL formula being created, and they serve to expose and name the external arguments that will be filled in later at higher levels in the grammar.

These last few rules also display the division of labor that we've seen several times now; lexical rules introduce the bulk of the semantic content, while higher level grammatical rules assemble the pieces in the right way, rather than introducing content.

Of course, as was the case with the λ-based approach things get quite a bit more complex when we look at expressions containing quantifiers. To see this, let's work through the following example.

(18.8) Every restaurant closed

Again, the meaning representation for this expression should be the following

$$\forall x Restaurant(x) \Rightarrow (\exists e Closing(e) \wedge Closed(e,x))$$

which is captured by the following feature structure.

$$\left[\begin{array}{ll}
\text{QUANT} & \forall \\
\text{VAR} & \boxed{1} \\
\text{FORMULA} & \left[\begin{array}{ll}
\text{OP} & \Rightarrow \\
\text{FORMULA1} & \left[\begin{array}{ll}\text{PRED} & \text{RESTAURANT}\\ \text{ARG0} & \boxed{1}\end{array}\right] \\
\text{FORMULA2} & \left[\begin{array}{ll}
\text{QUANT} & \text{EXISTS}\\
\text{VAR} & \boxed{2}\\
\text{FORMULA} & \left[\begin{array}{ll}
\text{OP} & \wedge\\
\text{FORMULA1} & \left[\begin{array}{ll}\text{PRED} & \text{CLOSING}\\ \text{ARG0} & \boxed{2}\end{array}\right]\\
\text{FORMULA2} & \left[\begin{array}{ll}\text{PRED} & \text{CLOSED}\\ \text{ARG0} & \boxed{2}\\ \text{ARG1} & \boxed{1}\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]$$

As we saw earlier with the λ-based approach, the outer structure for expressions like this comes largely from the subject noun phrase. Recall that schematically this semantic structure has the form $\forall x P(x) \Rightarrow Q(x)$ where the P expression is traditionally referred to as the *restrictor* and is provided by the head noun and Q is referred to as the *nuclear scope* and comes from the verb phrase.

This structure gives rise to two distinct tasks for our semantic attachments: the semantics of the *VP* semantics must be unified with the nuclear scope of the subject noun phrase, and the variable representing that noun phrase must be assigned to the ARG1 role of the CLOSED predicate in the event structure. The following rules involved in the derivation of *Every restaurant* address these two tasks

NP → *Det Nominal*
⟨ *NP* SEM⟩ = ⟨*Det* SEM ⟩
⟨ *NP* SEM VAR ⟩ = ⟨ *NP* INDEXVAR ⟩
⟨ *NP* SEM FORMULA FORMULA1 ⟩ = ⟨ *Nominal* SEM ⟩
⟨ *NP* SEM FORMULA FORMULA2 ⟩ = ⟨ *NP* SCOPE ⟩

Nominal → *Noun*
⟨ *Nominal* SEM ⟩ = ⟨ *Noun* SEM ⟩
⟨ *Nominal* INDEXVAR ⟩ = ⟨ *Noun* INDEXVAR ⟩

Noun → *restaurant*
⟨ *Noun* SEM PRED ⟩ = ⟨ RESTAURANT ⟩
⟨ *Noun* INDEXVAR ⟩ = ⟨ *Noun* SEM PRED ⟩

Det → *every*
⟨ *Det* SEM QUANT ⟩ = ∀
⟨ *Det* SEM FORMULA OP ⟩ = ⇒

As one final exercise, let's walk through an example with a transitive verb phrase.

(18.9) Franco opened a restaurant

This example has the following meaning representation.

$$\exists x\, Resaurant(x) \wedge \exists e\, Opening(e) \wedge Opener(e, Franco) \wedge Opened(e, x)$$

$$\begin{bmatrix}
\text{QUANT} & \text{EXISTS} \\
\text{VAR} & \boxed{1} \\
\text{FORMULA} & \begin{bmatrix}
\text{OP} & \wedge \\
\text{FORMULA1} & \begin{bmatrix}\text{PRED} & \text{RESTAURANT} \\ \text{ARG1} & \boxed{1}\end{bmatrix} \\
\text{FORMULA2} & \begin{bmatrix}
\text{QUANT} & \exists \\
\text{VAR} & \boxed{2} \\
\text{FORMULA} & \begin{bmatrix}
\text{OP} & \wedge \\
\text{FORMULA1} & \begin{bmatrix}\text{PRED} & \text{OPENING} \\ \text{ARG0} & \boxed{2}\end{bmatrix} \\
\text{FORMULA2} & \begin{bmatrix}\text{PRED} & \text{OPENER} \\ \text{ARG0} & \boxed{2} \\ \text{ARG1} & \text{FRANCO}\end{bmatrix} \\
\text{FORMULA3} & \begin{bmatrix}\text{PRED} & \text{OPENED} \\ \text{ARG0} & \boxed{2} \\ \text{ARG1} & \boxed{1}\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}$$

The only really new element that we need to address in this example is the following transitive *VP* rule.

$$\begin{aligned}
&VP \;\rightarrow\; Verb\ NP \\
&\quad \langle VP\ \text{SEM} \rangle = \langle Verb\ \text{SEM} \rangle \\
&\quad \langle NP\ \text{SCOPE} \rangle = \langle VP\ \text{SEM} \rangle \\
&\quad \langle Verb\ \text{ARG1} \rangle = \langle NP\ \text{INDEXVAR} \rangle
\end{aligned}$$

This rule has the two primary tasks that parallel those in our *S* rule: it has to fill the nuclear scope of the object *NP* with the semantics of the *VP*, and it has to insert the variable representing the object into to the right role in the *VP*'s meaning representation.

One obvious problem with the approach we just described is that it fails to generate all the possible ambiguous representations arising from quantifier scope ambiguities. Fortunately, the approaches to underspecification described earlier in Sec. 18.3 can be adapted to the unification-based approach.

18.5 SEMANTIC ATTACHMENTS FOR A FRAGMENT OF ENGLISH

This section describes a set of semantic attachments for a small fragment of English, the bulk of which are based on those used in the Core Language Engine (Alshawi, 1992). As in the rest of this chapter, to keep the presentation simple, we omit the feature structures associated with these rules when they are not needed. Remember that these features are needed to ensure that the correct rules are applied in the correct situations. Most importantly for this discussion, they are needed to ensure that the correct verb entries are being employed based on their subcategorization feature structures.

18.5.1 Sentences

To this point, we've only dealt with simple declarative sentences. This section expands our coverage to include the other sentence types first introduced in Ch. 12: imperatives, yes-no-questions, and wh-questions. Let's start by considering the following examples:

(18.10) Flight 487 serves lunch.

(18.11) Serve lunch.

(18.12) Does Flight 207 serve lunch?

(18.13) Which flights serve lunch?

The meaning representations of these examples all contain propositions concerning the serving of lunch on flights. However, they differ with respect to the role that these propositions are intended to serve in the settings in which they are uttered. More specifically, the first example is intended to convey factual information to a listener, the second is a request for an action, and the last two are requests for information. To capture these differences, we will introduce a set of operators that can be applied to FOL sentences in the same way that belief operators were used in Ch. 17. Specifically, the operators *DCL*, *IMP*, *YNQ*, and *WHQ* will be applied to the FOL representations of declaratives, imperatives, yes-no-questions, and wh-questions, respectively.

Producing meaning representations that make appropriate use of these operators requires the right set of semantic attachments for each of the possible sentence types. For declarative sentences, we can simply alter the basic sentence rule we have been using as follows:

$$S \rightarrow NP\ VP \qquad \{DCL(NP.sem(VP.sem))\}$$

The normal interpretation for a representation headed by the *DCL* operator would be as a factual statement to be added to the current knowledge-base.

Imperative sentences begin with a verb phrase and lack an overt subject. Because of the missing subject, the meaning representation for the main verb phrase will consist of a λ-expression with an unbound λ-variable representing this missing subject. To deal with this, we can simply *supply* a subject to the λ-expression by applying a final λ-reduction to a dummy constant. The *IMP* operator can then be applied to this representation as in the following semantic attachment:

$$S \rightarrow VP \qquad \{IMP(VP.sem(DummyYou))\}$$

Applying this rule to example (18.11), results in the following representation:

$$IMP(\exists e Serving(e) \wedge Server(e, DummyYou) \wedge Served(e, Lunch)$$

As will be discussed in Ch. 23, imperatives can be viewed as a kind of **speech act**.

As discussed in Ch. 12, **yes-no-questions** consist of a sentence-initial auxiliary verb, followed by a subject noun phrase and then a verb phrase. The following semantic attachment simply ignores the auxiliary, and with the exception of the *YNQ* operator, constructs the same representation that would be created for the corresponding declarative sentence:

$$S \rightarrow Aux\ NP\ VP \qquad \{YNQ(VP.sem(NP.sem))\}$$

The use of this rule with for example (18.12) produces the following representation:

$$YNQ(\exists e Serving(e) \wedge Server(e, Flt207) \wedge Served(e, Lunch))$$

Yes-no-questions should be thought as asking whether the propositional part of its meaning is true or false given the knowledge currently contained in the knowledge-base. Adopting the kind of semantics described in Ch. 17, yes-no-questions can be answered by determining if the proposition is in the knowledge-base, or can be inferred from it.

Unlike yes-no-questions, **wh-subject-questions** ask for specific information about the subject of the sentence rather than the sentence as a whole. The following attachment produces a representation that consists of the operator *WHQ*, the variable corresponding to the subject of the sentence, and the body of the proposition:

$$S \rightarrow WhWord\ NP\ VP \qquad \{WHQ(NP.sem.var, VP.sem(NP.sem))\}$$

The following representation is the result of applying this rule to example (18.13):

$$\begin{aligned}WHQ(x, \exists e, x\ Isa(e, Serving) \wedge Server(e, x)\\ \wedge Served(e, Lunch) \wedge Isa(x, Flight))\end{aligned}$$

Such questions can be answered by returning a set of assignments for the subject variable that make the resulting proposition true with respect to the current knowledge-base.

Finally, consider the following **wh-non-subject-question**:

(18.14) How can I go from Minneapolis to Long Beach?

In examples like this, the question is not about the subject of the sentence but rather some other argument, or some aspect of the proposition as a whole. In this case, the representation needs to provide an indication as to what the question is about. The following attachment provides this information by providing the semantics of the auxiliary as an argument to the *WHQ* operator:

$$S \rightarrow \textit{WhWord Aux NP VP} \quad \{WHQ(\ WhWord.sem\ VP.sem(NP.sem))\}$$

The following representation would result from an application of this rule to example (18.14):

$$\begin{aligned} &WHQ(How, \exists e\ Isa(e, Going) \wedge Goer(e, User) \\ &\qquad \wedge Origin(e, Minn) \wedge Destination(e, LongBeach)) \end{aligned}$$

As we'll see in Ch. 23, correctly answering this kind of question involves a fair amount of domain specific reasoning. For example, the correct way to answer example (18.14) is to search for flights with the specified departure and arrival cities. Note, however, that there is no mention of flights or flying in the actual question. The question-answerer, therefore, has to apply knowledge specific to this domain to the effect that questions about going places are really questions about flights to those places.

Finally, we should make it clear that this particular attachment is only useful for rather simple wh-questions without missing arguments or embedded clauses. As discussed in Ch. 16, the presence of long-distance dependencies in these questions requires additional mechanisms to determine exactly what is being asked about. Woods (1977) and Alshawi (1992) provide extensive discussions of general mechanisms for handling wh-non-subject questions.

18.5.2 Noun Phrases

As we have already seen, the meaning representations for noun phrases can be either normal FOL terms or complex-terms. The following sections detail the semantic attachments needed to produce meaning representations for some of the most frequent kinds of English noun phrases. Unfortunately, as we will see, the syntax of English noun phrases provides surprisingly little insight into their meaning. It is often the case that the best we can do is provide a rather vague intermediate level of meaning representation that can serve as input to further interpretation processes.

Compound Nominals

Compound nominals, also known as noun-noun sequences, consist of simple sequences of nouns, as in the following examples:

(18.15) Flight schedule

(18.16) Summer flight schedule

As noted in Ch. 12, the syntactic structure of this construction can be captured by the regular expression *Noun*∗, or by the following context-free grammar rules:

$$\begin{aligned} \textit{Nominal} &\rightarrow \textit{Noun} \\ \textit{Nominal} &\rightarrow \textit{Nominal Noun} \end{aligned}$$

In these constructions, the final noun in the sequence is the head of the phrase and denotes an object that is semantically related in some unspecified way to the other nouns that precede it in the sequence. In general, an extremely wide range of common-sense relations can be denoted by this construction. Discerning the exact nature of these relationships is well beyond the scope of the kind of superficial semantic analysis presented in this chapter. The attachment in the following rule builds up a vague representation that simply notes the existence of a semantic relation between the head noun and the modifying nouns, by incrementally noting such a relation between the head noun and each noun to its left:

Nominal → *Noun Nominal*
$\{\lambda x\ \textit{Nominal.sem(x)} \wedge \textit{NN(Noun.sem, x)}\}$

The relation *NN* is used to specify that a relation holds between the modifying elements of a compound nominal and the head *Noun*. In the examples given above, this leads to the following meaning representations:

$$\lambda x Isa(x, Schedule) \wedge NN(x, Flight)$$

$$\lambda x Isa(x, Schedule) \wedge NN(x, Flight) \wedge NN(x, Summer)$$

Note that this representation correctly instantiates a term representing a *Schedule*, while avoiding the creation of terms representing either a $Flight$ or *Summer*.

Genitive Noun Phrases

Recall from Ch. 12 that genitive noun phrases make use of complex determiners that consist of noun phrases with possessive markers, as in *Atlanta's airport* and *Maharani's menu*. It is quite tempting to represent the relation between these words as an abstract kind of possession. A little introspection, however, reveals that the relation between a city and its airport has little in common with a restaurant and its menu. Therefore, as with compound nominals, it's best to simply state an abstract semantic relation between the various constituents.

NP → *ComplexDet Nominal*
$\{< \exists x Nominal.sem(x) \wedge GN(x, ComplexDet.sem) >\}$

ComplexDet → *NP 's* $\{NP.sem\}$

Applying these rules to *Atlanta's airport* results in the following complex-term:

$$< \exists x Isa(x, Airport) \wedge GN(x, Atlanta) >$$

Subsequent semantic interpretation would have to determine that the relation denoted by the relation *GN* is actually a location.

Adjective Phrases

English adjectives can be split into two major categories: pre-nominal and predicative. These categories are exemplified by the following BERP examples:

(18.17) I don't mind a cheap restaurant.

(18.18) This restaurant is cheap.

For the pre-nominal case, an obvious *and often incorrect* proposal for the semantic attachment is illustrated in the following rules:

$$\begin{array}{l} Nominal \rightarrow Adj\ Nominal \\ \quad \{\lambda x\ Nominal.sem(x) \wedge Isa(x, Adj.sem)\} \end{array}$$

$$Adj \rightarrow cheap \qquad \{Cheap\}$$

This solution modifies the semantics of the nominal by applying the predicate provided by the adjective to the variable representing the nominal. For our cheap restaurant example, this yields the following not unreasonable representation:

$$\lambda x\ Isa(x, Restaurant) \wedge Isa(x, Cheap)$$

INTERSECTIVE SEMANTICS

This is an example of what is known as **intersective semantics** since the meaning of the phrase can be thought of as the intersection of the category stipulated by the nominal and the category stipulated by the adjective. In this case, this amounts to the intersection of the category of cheap things with the category of restaurants.

Unfortunately, this solution often does the wrong thing. For example, consider the following meaning representations for the phrases *small elephant*, *former friend*, and *fake gun*:

$$\lambda x\ Isa(x, Elephant) \wedge Isa(x, Small)$$

$$\lambda x\ Isa(x, Friend) \wedge Isa(x, Former)$$

$$\lambda x\ Isa(x, Gun) \wedge Isa(x, Fake)$$

Each of these representations is peculiar in some way. The first one states that this particular elephant is a member of the general category of small things, which is probably not true. The second example is strange in two ways: it asserts that the person in question is a friend, which is false, and it makes use of a fairly unreasonable category of *former things*. Similarly, the third example asserts that the object in question is a gun despite the fact that *fake* means it is not one.

As with compound nominals, there is no clever solution to these problems within the bounds of our current compositional framework. Therefore, the best approach is to simply note the status of a specific kind of modification relation and assume that some further procedure with access to additional relevant knowledge can replace this vague relation with an appropriate representation (Alshawi, 1992).

$$\begin{array}{l} Nominal \rightarrow Adj\ Nominal \\ \quad \{\lambda x\ Nominal.sem(x) \wedge AM(x, Adj.sem)\} \end{array}$$

Applying this rule to *a cheap restaurant* results in the following formula:

$$\exists x\ Isa(x, Restaurant) \wedge AM(x, Cheap)$$

Note that even this watered-down proposal produces representations that are logically incorrect for the *fake* and *former* examples. In both cases, it asserts that the objects in question are in fact members of their stated categories. In general, the solution to this problem has to be based on the specific semantics of the adjectives and nouns in question. For example, the semantics of *former* has to involve some form of temporal reasoning, while *fake* requires the ability to reason about the nature of concepts and categories.

18.5.3 Verb Phrases

The general schema for computing the semantics of verb phrases relies on the notion of function application. In most cases, the λ-expression attached to the verb is simply applied to the semantic attachments of the verb's arguments. There are, however, a number of situations that force us to depart somewhat from this general pattern.

Infinitive Verb Phrases

A fair number of English verbs take some form of verb phrase as one of their arguments. This complicates the normal verb phrase semantic schema since these argument verb phrases interact with the other arguments of the head verb in ways that are not completely obvious.

Consider the following example:

(18.19) I told Harry to go to Maharani.

The meaning representation for this example should be something like the follow-

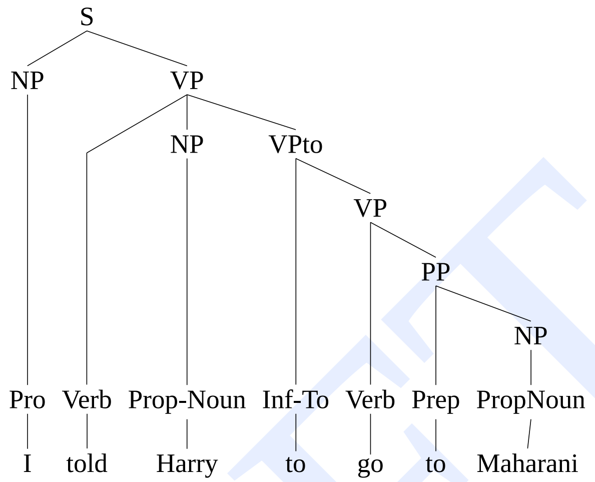


Figure 18.5 Parse tree for *I told Harry to go to Maharani.*

ing:

$$\begin{aligned}&\exists e, f, x\; Isa(e, Telling) \wedge Isa(f, Going)\\ &\quad \wedge Teller(e, Speaker) \wedge Tellee(e, Harry) \wedge ToldThing(e, f)\\ &\quad \wedge Goer(f, Harry) \wedge Destination(f, x)\end{aligned}$$

There are two interesting things to note about this meaning representation: the first is that it consists of two events, and the second is that one of the participants, *Harry*, plays a role in both of the two events. The difficulty in creating this complex representation falls to the verb phrase dominating the verb *tell* which will need something like the following as its semantic attachment:

$$\begin{aligned}&\lambda x, y\; \lambda z\; \exists e\; Isa(e, Telling)\\ &\quad \wedge Teller(e, z) \wedge Tellee(e, x) \wedge ToldThing(e, y)\end{aligned}$$

Semantically, we can interpret this subcategorization frame for *Tell* as providing three semantic roles: a person doing the telling, a recipient of the telling, and the proposition being conveyed.

The difficult part of this example involves getting the meaning representation for the main verb phrase correct. As shown in Figure 18.5, *Harry* plays the role of both the *Tellee* of the *Telling* event and the *Goer* of the *Going* event. However, *Harry* is not available when the *Going* event is created within the infinitive verb phrase.

Although there are several possible solutions to this problem, it is usually best to stick with a uniform approach to these problems. Therefore, we will start by simply applying the semantics of the verb to the semantics of the other arguments of the verb as follows:

$$VP \rightarrow Verb\ NP\ VPto \qquad \{Verb.sem(NP.sem,\ VPto.sem)\}$$

Since the *to* in the infinitive verb phrase construction does not contribute to its meaning, we simply copy the meaning of the child verb phrase up to the infinitive verb phrase. Recall, that we are relying on the unseen feature structures to ensure that only the correct verb phrases can be used with this construction.

$$VPto \rightarrow to\ VP \qquad \{VP.sem\}$$

In this solution, the verb's semantic attachment has two tasks: incorporating the *NP.sem*, the *Goer*, into the *VPto.sem*, and incorporating the *Going* event as the $ToldThing$ of the $Telling$. The following attachment performs both tasks:

$$\begin{array}{l} Verb \rightarrow tell \\ \quad \{\lambda x, y \\ \qquad \lambda z \\ \qquad\quad \exists e, y.variable\ Isa(e, Telling) \\ \qquad\qquad\quad \wedge Teller(e,z) \wedge Tellee(e,x) \\ \qquad\qquad\quad \wedge ToldThing(e, y.variable) \wedge y(x) \end{array}$$

In this approach, the λ-variable x plays the role of the $Tellee$ of the telling and the argument to the semantics of the infinitive, which is now contained as a λ-expression in the variable y. The expression $y(x)$ represents a λ-reduction that inserts $Harry$ into the *Going* event as the *Goer*. The notation *y.variable*, is analogous to the notation used for complex-term variables, and gives us access to the event variable representing the *Going* event within the infinitive's meaning representation.

Note that this approach plays fast and loose with the definition of λ-reduction, in that it allows λ-expressions to be passed as arguments to other λ-expressions, when technically only FOPC terms can serve that role. This technique is a convenience similar to the use of complex-terms in that it allows us to temporarily treat complex expressions as terms during the creation of meaning representations.

18.5.4 Prepositional Phrases

At a fairly abstract level, prepositional phrases serve two distinct functions: they assert binary relations between their heads and the constituents to which they are attached, and they signal arguments to constituents that have an argument structure. These two functions argue for two distinct types of prepositional phrases that differ based on their semantic attachments. We will consider three places in the grammar where prepositional phrases serve these roles: modifiers of noun phrases, modifiers of verb phrases, and arguments to verb phrases.

Nominal Modifier Prepositional Phrases

Modifier prepositional phrases denote a binary relation between the concept being modified, which is external to the prepositional phrase, and the head of the prepositional phrase. Consider the following example and its associated meaning representation:

(18.20) A restaurant on Broadway.

$$\exists x\, Isa(x, Restaurant) \wedge On(x, Pearl)$$

The relevant grammar rules that govern this example are the following:

$$NP \rightarrow Det\ Nominal$$
$$Nominal \rightarrow Nominal\ PP$$
$$PP \rightarrow P\ NP$$

Proceeding in a bottom-up fashion, the semantic attachment for this kind of relational preposition should provide a two-place predicate with its arguments distributed over two λ-expressions, as in the following:

$$P \rightarrow on \qquad \{\lambda y \lambda x\, On(x,y)\}$$

With this kind of arrangement, the first argument to the predicate is provided by the head of prepositional phrase and the second is provided by the constituent that the prepositional phrase is ultimately attached to. The following semantic attachment provides the first part:

$$PP \rightarrow P\ NP \qquad \{P.sem(NP.sem)\}$$

This λ-application results in a new λ-expression where the remaining argument is the inner λ-variable.

This remaining argument can be incorporated using the following nominal construction:

$$Nominal \rightarrow Nominal\ PP \qquad \{\lambda z Nominal.sem(z) \wedge PP.sem(z)\}$$

Verb Phrase Modifier Prepositional Phrases

The general approach to modifying verb phrases is similar to that of modifying nominals. The differences lie in the details of the modification in the verb phrase rule; the attachments for the preposition and prepositional phrase rules are unchanged. Let's consider the phrase *ate dinner in a hurry* which is governed by the following verb phrase rule:

$$VP \rightarrow VP\ PP$$

The meaning representation of the verb phrase constituent in this construction, *ate dinner*, is a λ-expression where the λ-variable represents the as yet unseen subject.

$$\lambda x \exists e\, Isa(e, Eating) \wedge Eater(e,x) \wedge Eaten(e, Dinner)$$

The representation of the prepositional phrase is also a λ-expression where the λ-variable is the second argument in the *PP* semantics.

$$\lambda x\, In(x, < \exists h\, Hurry(h) >)$$

The correct representation for the modified verb phrase should contain the conjunction of these two representations with the *Eating* event variable filling the first argument slot of the *In* expression. In addition, this modified representation must remain a λ-expression with the unbound *Eater* variable as the new λ-variable. The following attachment expression fulfills all of these requirements:

$$VP \rightarrow VP\ PP \qquad \{\lambda y VP.sem(y) \wedge PP.sem(VP.sem.variable)\}$$

There are two aspects of this attachment that require some elaboration. The first involves the application of the constituent verb phrases' λ-expression to the variable y. Binding the lower λ-expression's variable to a new variable allows us to *lift* the lower variable to the level of the newly created λ-expression. The result of this technique is a new λ-expression with a variable that, in effect, plays the same role as the original variable in the lower expression. In this case, this allows a λ-expression to be modified during the analysis process before the argument to the expression is actually available.

The second notable aspect of this attachment involves the $VP.sem.variable$ notation. This notation is used to access the event-variable representing the underlying meaning of the verb phrase, in this case, e. This is analogous to the notation used to provide access to the various parts of complex-terms introduced earlier.

Applying this attachment to the current example yields the following representation, which is suitable for combination with a subsequent subject noun phrase:

$$\begin{aligned}&\lambda y \exists e\, Isa(e, Eating) \wedge Eater(e,y) \wedge Eaten(e, Dinner)\\ &\qquad\quad \wedge In(e, < \exists h Hurry(h) >)\end{aligned}$$

Verb Argument Prepositional Phrases

The prepositional phrases in this category serve to signal the role an argument plays in some larger event structure. As such, the preposition itself does not actually modify the meaning of the noun phrase. Consider the following example of role signaling prepositional phrases:

(18.21) I need to go from Boston to Dallas.

In examples like this, the arguments of *go* are expressed as prepositional phrases. However, the meaning representations of these phrases should consist solely of the unaltered representation of their head nouns. To handle this, argument prepositional phrases are treated in the same way that non-branching grammatical rules are; the semantic attachment of the noun phrase is copied unchanged to the semantics of the larger phrase.

$$PP \rightarrow P\ NP \qquad \{NP.sem\}$$

The verb phrase can then assign this meaning representation to the appropriate event role. A more complete account of how these argument bearing prepositional phrases map to underlying event roles will be presented in Ch. 19.

18.6 Integrating Semantic Analysis into the Earley Parser

In Section 18.1, we suggested a simple pipeline architecture for a semantic analyzer where the results of a complete syntactic parse are passed to a semantic analyzer. The motivation for this notion stems from the fact that the compositional approach requires the syntactic parse before it can proceed. It is, however, also possible to perform semantic analysis in parallel with syntactic processing. This is possible because in our compositional framework, the meaning representation for a constituent can be created as soon as all of its constituent parts are present. This section describes just such an approach to integrating semantic analysis into the Earley parser from Ch. 13.

The integration of semantic analysis into an Earley parser is straightforward and follows precisely the same lines as the integration of unification into the algorithm given in Ch. 16. Three modifications are required to the original algorithm:

1. The rules of the grammar are given a new field to contain their semantic attachments.
2. The states in the chart are given a new field to hold the meaning representation of the constituent.
3. The ENQUEUE function is altered so that when a complete state is entered into the chart its semantics are computed and stored in the state's semantic field.

Figure 18.6 shows ENQUEUE modified to create meaning representations. When ENQUEUE is passed a complete state that can successfully unify its unification constraints it calls APPLY-SEMANTICS to compute and store the meaning representation for this state. Note the importance of performing feature-structure unification prior to semantic analysis. This ensures that semantic analysis will be

```
procedure Enqueue(state, chart-entry)
 if Incomplete?(state) then
    if state is not already in chart-entry then
       Push(state, chart-entry)
 else if Unify-State(state) succeeds then
    if Apply-Semantics(state) succeeds then
      if state is not already in chart-entry then
       Push(state, chart-entry)

procedure Apply-Semantics(state)
 meaning-rep ← Apply(state.semantic-attachment, state)
 if meaning-rep does not equal failure then
   state.meaning-rep ← meaning-rep
```

Figure 18.6 The Enqueue function modified to handle semantics. If the state is complete and unification succeeds then Enqueue calls Apply-Semantics to compute and store the meaning representation of completed states.

performed only on valid trees and that features needed for semantic analysis will be present.

The primary advantage of this integrated approach over the pipeline approach lies in the fact that Apply-Semantics can fail in a manner similar to the way that unification can fail. If a semantic ill-formedness is found in the meaning representation being created, the corresponding state can be blocked from entering the chart. In this way, semantic considerations can be brought to bear during syntactic processing. Ch. 19 describes in some detail the various ways that this notion of ill-formedness can be realized.

Unfortunately, this also illustrates one of the primary disadvantages of integrating semantics directly into the parser—considerable effort may be spent on the semantic analysis of *orphan* constituents that do not in the end contribute to a successful parse. The question of whether the gains made by bringing semantics to bear early in the process outweigh the costs involved in performing extraneous semantic processing can only be answered on a case-by-case basis.

18.7 Idioms and Compositionality

> Ce corps qui s'appelait et qui s'appelle encore le saint empire romain n'était en aucune manière ni saint, ni romain, ni empire.
>
> This body, which called itself and still calls itself the Holy Roman Empire, was neither Holy, nor Roman, nor an Empire.
>
> Voltaire[2], 1756

As innocuous as it seems, the principle of compositionality runs into trouble fairly quickly when real language is examined. There are many cases where the meaning of a constituent is not based on the meaning of its parts, at least not in the straightforward compositional sense. Consider the following WSJ examples:

(18.22) Coupons are just the tip of the iceberg.

(18.23) The SEC's allegations are only the tip of the iceberg.

(18.24) Coronary bypass surgery, hip replacement and intensive-care units are but the tip of the iceberg.

The phrase *the tip of the iceberg* in each of these examples clearly doesn't have much to do with tips or icebergs. Instead, it roughly means something like *the beginning*. The most straightforward way to handle idiomatic constructions like these is to introduce new grammar rules specifically designed to handle them. These idiomatic rules mix lexical items with grammatical constituents, and introduce semantic content that is not derived from any of its parts. Consider the following rule as an example of this approach:

$$\begin{aligned} NP &\rightarrow the\ tip\ of\ the\ iceberg \\ &\quad \{Beginning\} \end{aligned}$$

The lower case items on the right-hand side of this rule are intended to represent precisely words in the input. Although, the constant *Beginning* should not be taken too seriously as a meaning representation for this idiom, it does illustrate the idea that the meaning of this idiom is not based on the meaning of any of its parts. Note that an Earley-style analyzer with this rule will now produce two parses when this phrase is encountered: one representing the idiom and one representing the compositional meaning.

As with the rest of the grammar, it may take a few tries to get these rules right. Consider the following *iceberg* examples from the WSJ corpus:

(18.25) And that's but the tip of Mrs. Ford's iceberg.

[2] *Essai sur les moeurs et les esprit des nations.* Translation by Y. Sills, as quoted in Sills and Merton (1991).

(18.26) These comments describe only the tip of a 1,000-page iceberg.

(18.27) The 10 employees represent the merest tip of the iceberg.

The rule given above is clearly not general enough to handle these cases. These examples indicate that there is a vestigial syntactic structure to this idiom that permits some variation in the determiners used, and also permits some adjectival modification of both the *iceberg* and the *tip*. A more promising rule would be something like the following:

$$\begin{aligned} &NP \rightarrow TipNP\ of\ IcebergNP \\ &\quad \{Beginning\} \end{aligned}$$

Here the categories *TipNP* and *IcebergNP* can be given an internal nominal-like structure that permits some adjectival modification and some variation in the determiners, while still restricting the heads of these noun phrases to the lexical items *tip* and *iceberg*. Note that this syntactic solution ignores the thorny issue that the modifiers *mere* and *1000-page* seem to indicate that both the *tip* and *iceberg* may in fact play some compositional role in the meaning of the idiom. We will return to this topic in Ch. 19, when we take up the issue of metaphor.

To summarize, handling idioms requires at least the following changes to the general compositional framework:

- Allow the mixing of lexical items with traditional grammatical constituents.
- Allow the creation of additional idiom-specific constituents needed to handle the correct range of productivity of the idiom.
- Permit semantic attachments that introduce logical terms and predicates that are not related to any of the constituents of the rule.

This discussion is obviously only the tip of an enormous iceberg. Idioms are far more frequent and far more productive than is generally recognized and pose serious difficulties for many applications, including, as we will see in Ch. 24, machine translation.

18.8 Summary

This chapter explores the notion of syntax-driven semantic analysis. Among the highlights of this chapter are the following topics:

- **Semantic analysis** is the process whereby meaning representations are created and assigned to linguistic inputs.
- **Semantic analyzers** that make use of static knowledge from the lexicon and grammar can create context-independent literal, or conventional, meanings.

- The **Principle of Compositionality** states that the meaning of a sentence can be composed from the meanings of its parts.
- In **Syntax-driven semantic analysis**, the parts are the syntactic constituents of an input.
- Compositional creation of FOL formulas is possible with a few notational extensions including **λ-expressions** and **complex-terms**.
- Compositional creation of FOL formulas is also possible using the mechanisms provided by feature structures and unification.
- **Natural language quantifiers** introduce a kind of ambiguity that is difficult to handle compositionally. Complex-terms can be used to compactly encode this ambiguity.
- **Idiomatic language** defies the principle of compositionality but can easily be handled by adapting the techniques used to design grammar rules and their semantic attachments.

Bibliographical and Historical Notes

As noted earlier, the principle of compositionality is traditionally attributed to Frege; Janssen (1997) discusses this attribution. Using the categorial grammar framework described in Ch. 14, Montague (1973) demonstrated that a compositional approach could be systematically applied to an interesting fragment of natural language. The rule-to-rule hypothesis was first articulated by Bach (1976). On the computational side of things, Woods's LUNAR system (Woods, 1977) was based on a pipelined syntax-first compositional analysis. Schubert and Pelletier (1982) developed an incremental rule-to-rule system based on Gazdar's GPSG approach (Gazdar, 1981, 1982; Gazdar et al., 1985). Main and Benson (1983) extended Montague's approach to the domain of question-answering.

In one of the all-too-frequent cases of parallel development, researchers in programming languages developed essentially identical compositional techniques to aid in the design of compilers. Specifically, Knuth (1968) introduced the notion of attribute grammars that associate semantic structures with syntactic structures in a one-to-one correspondence. As a consequence, the style of semantic attachments used in this chapter will be familiar to users of the YACC-style (Johnson and Lesk, 1978) compiler tools.

Semantic Grammars are due to Burton (Brown and Burton, 1975). Similar notions developed around the same time included Pragmatic Grammars (Woods, 1977) and Performance Grammars (Robinson, 1975). All centered around the notion of reshaping syntactic grammars to serve the needs of semantic processing. It is safe to say that most modern systems developed for use in limited domains make use of some form of semantic grammar.

Most of the techniques used in the fragment of English presented in Section 18.5 are adapted from SRI's Core Language Engine (Alshawi, 1992). Additional bits and pieces were adapted from Woods (1977), Schubert and Pelletier (1982), and Gazdar et al. (1985). Of necessity, a large number of important topics were not covered in this chapter. See Alshawi (1992) for the standard gap-threading approach to semantic interpretation in the presence of long-distance dependencies. ter Meulen (1995) presents an modern treatment of tense, aspect, and the representation of temporal information. Extensive coverage of approaches to quantifier scoping can be found in Hobbs and Shieber (1987) and Alshawi (1992). van Lehn (1978) presents a set of human preferences for quantifier scoping. Over the years, a considerable amount of effort has been directed toward the interpretation of compound nominals. Linguistic research on this topic can be found in Lees (1970), Downing (1977), Levi (1978), and Ryder (1994), more computational approaches are described in Gershman (1977), Finin (1980), McDonald (1982), Pierre (1984), Arens et al. (1987), Wu (1992), Vanderwende (1994), and Lauer (1995).

There is a long and extensive literature on idioms. Fillmore et al. (1988) describe a general grammatical framework called Construction Grammar that places idioms at the center of its underlying theory. Makkai (1972) presents an extensive linguistic analysis of many English idioms. Hundreds of idiom dictionaries for second-language learners are also available. On the computational side, Becker (1975) was among the first to suggest the use of phrasal rules in parsers. Wilensky and Arens (1980) were among the first to successfully make use of this notion in their PHRAN system. Zernik (1987) demonstrated a system that could learn such phrasal idioms in context. A collection of papers on computational approaches to idioms appeared in (Fass et al., 1992).

Finally, we have skipped an entire branch of semantic analysis in which expectations driven from deep meaning representations drive the analysis process. Such systems avoid the direct representation and use of syntax, rarely making use of anything resembling a parse tree. Some of the earliest and most successful efforts along these lines were developed by Simmons (1973, 1978, 1983) and (Wilks, 1975a, 1975b). A series of similar approaches were developed by Roger Schank and his students (Riesbeck, 1975; Birnbaum and Selfridge, 1981; Riesbeck, 1986). In these approaches, the semantic analysis process is guided by detailed procedures associated with individual lexical items. The CIRCUS information extraction system (Lehnert et al., 1991) traces its roots to these systems.

EXERCISES

18.1 The attachment given on page 23 for handling noun phrases with complex determiners is not general enough to handle most possessive noun phrases. Specifically, it doesn't work for phrases like the following:

a. My sister's flight

b. My fiance's mother's flight

Create a new set of semantic attachments to handle cases like these.

18.2 Develop a set of grammar rules and semantic attachments to handle predicate adjectives such as the one following:

a. Flight 308 from New York is expensive.

b. Murphy's restaurant is cheap.

18.3 None of the attachments given in this chapter provide temporal information. Augment a small number of the most basic rules to add temporal information along the lines sketched in Ch. 17. Use your rules to create meaning representations for the following examples:

a. Flight 299 departed at 9 o'clock.

b. Flight 208 will arrive at 3 o'clock.

c. Flight 1405 will arrive late.

18.4 As noted in Ch. 17, the present tense in English can be used to refer to either the present or the future. However, it can also be used to express habitual behavior, as in the following:

Flight 208 leaves at 3 o'clock.

This could be a simple statement about today's Flight 208, or alternatively it might state that this flight leaves at 3 o'clock every day. Create a FOPC meaning representation along with appropriate semantic attachments for this habitual sense.

18.5 Implement an Earley-style semantic analyzer based on the discussion on page 30.

18.6 It has been claimed that it is not necessary to explicitly list the semantic attachment for most grammar rules. Instead, the semantic attachment for a rule should be inferable from the semantic types of the rule's constituents. For example, if a rule has two constituents, where one is a single argument λ-expression and the other is a constant, then the semantic attachment should obviously apply the λ-expression to the constant. Given the attachments presented in this chapter, does this *type-driven semantics* seem like a reasonable idea?

18.7 Add a simple type-driven semantics mechanism to the Earley analyzer you implemented for Exercise 18.5.

18.8 Using a phrasal search on your favorite Web search engine, collect a small corpus of *the tip of the iceberg* examples. Be certain that you search for an appropriate range of examples (i.e., don't just search for "the tip of the iceberg".) Analyze these examples and come up with a set of grammar rules that correctly accounts for them.

18.9 Collect a similar corpus of examples for the idiom *miss the boat*. Analyze these examples and come up with a set of grammar rules that correctly accounts for them.

18.10 There are now a fair number of Web-based natural language question answering services that purport to provide answers to questions on a wide range of topics (see the book's Web page for pointers to current services). Develop a corpus of questions for some general domain of interest and use it to evaluate one or more of these services. Report your results. What difficulties did you encounter in applying the standard evaluation techniques to this task?

18.11 Collect a small corpus of weather reports from your local newspaper or the Web. Based on an analysis of this corpus, create a set of frames sufficient to capture the semantic content of these reports.

18.12 Implement and evaluate a small information extraction system for the weather report corpus you collected for the last exercise.

Alshawi, H. (Ed.). (1992). *The Core Language Engine*. MIT Press.

Arens, Y., Granacki, J., and Parker, A. (1987). Phrasal analysis of long noun sequences. In *ACL-87*, Stanford, CA, pp. 59–64. ACL.

Bach, E. (1976). An extension of classical transformational grammar. In *Problems of Linguistic Metatheory (Proceedings of the 1976 Conference)*. Michigan State University.

Becker (1975). The phrasal lexicon. In Schank, R. and Nash-Webber, B. L. (Eds.), *Theoretical Issues in Natural Language Processing*. Cambridge, MA.

Birnbaum, L. and Selfridge, M. (1981). Conceptual analysis of natural language. In Schank, R. C. and Riesbeck, C. K. (Eds.), *Inside Computer Understanding: Five Programs plus Miniatures*, pp. 318–353. Lawrence Erlbaum.

Brown, J. S. and Burton, R. R. (1975). Multiple representations of knowledge for tutorial reasoning. In Bobrow, D. G. and Collins, A. (Eds.), *Representation and Understanding*, pp. 311–350. Academic Press.

Downing, P. (1977). On the creation and use of English compound nouns. *Language*, *53*(4), 810–842.

Fass, D., Martin, J. H., and Hinkelman, E. A. (Eds.). (1992). *Computational Intelligence: Special Issue on Non-Literal Language*, Vol. 8. Blackwell, Cambridge, MA.

Fillmore, C. J., Kay, P., and O'Connor, M. C. (1988). Regularity and idiomaticity in grammatical constructions: The case of Let Alone. *Language*, *64*(3), 510–538.

Finin, T. (1980). The semantic interpretation of nominal compounds. In *AAAI-80*, Stanford, CA, pp. 310–312.

Gazdar, G. (1981). Unbounded dependencies and coordinate structure. *Linguistic Inquiry*, *12*(2), 155–184.

Gazdar, G. (1982). Phrase structure grammar. In Jacobson, P. and Pullum, G. K. (Eds.), *The Nature of Syntactic Representation*, pp. 131–186. Reidel, Dordrecht.

Gazdar, G., Klein, E., Pullum, G. K., and Sag, I. A. (1985). *Generalized Phrase Structure Grammar*. Basil Blackwell, Oxford.

Gershman, A. V. (1977). Conceptual analysis of noun groups in English. In *IJCAI-77*, Cambridge, MA, pp. 132–138.

Hobbs, J. R. and Shieber, S. M. (1987). An algorithm for generating quantifier scopings. *Computational Linguistics*, *13*(1), 47–55.

Janssen, T. M. V. (1997). Compositionality. In van Benthem, J. and ter Meulen, A. (Eds.), *Handbook of Logic and Language*, chap. 7, pp. 417–473. North-Holland, Amsterdam.

Johnson, S. C. and Lesk, M. E. (1978). Language development tools. *Bell System Technical Journal*, *57*(6), 2155–2175.

Knuth, D. E. (1968). Semantics of context-free languages. *Mathematical Systems Theory*, *2*(2), 127–145.

Lauer, M. (1995). Corpus statistics meet the noun compound. In *ACL-95*, Cambridge, MA, pp. 47–54.

Lees, R. (1970). Problems in the grammatical analysis of English nominal compounds. In Bierwitsch, M. and Heidolph, K. E. (Eds.), *Progress in Linguistics*, pp. 174–187. Mouton, The Hague.

Lehnert, W. G., Cardie, C., Fisher, D., Riloff, E., and Williams, R. (1991). Description of the CIRCUS system as used for MUC-3. In Sundheim, B. (Ed.), *Proceedings of the Third Message Understanding Conference*, pp. 223–233. Morgan Kaufmann.

Levi, J. (1978). *The Syntax and Semantics of Complex Nominals*. Academic Press.

Main, M. G. and Benson, D. B. (1983). Denotational semantics for natural language question-answering programs. *American Journal of Computational Linguistics*, *9*(1), 11–21.

Makkai, A. (1972). *Idiom Structure in English*. Mouton, The Hague.

McDonald, D. B. (1982). *Understanding Noun Compounds*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA. CMU Technical Report CS-82-102.

Montague, R. (1973). The proper treatment of quantification in ordinary English. In Thomason, R. (Ed.), *Formal Philosophy: Selected Papers of Richard Montague*, pp. 247–270. Yale University Press, New Haven, CT.

Pierre, I. (1984). Another look at nominal compounds. In *COLING-84*, Stanford, CA, pp. 509–516.

Riesbeck, C. K. (1975). Conceptual analysis. In Schank, R. C. (Ed.), *Conceptual Information Processing*, pp. 83–156. American Elsevier, New York.

Riesbeck, C. K. (1986). From conceptual analyzer to direct memory access parsing: An overview. In *Advances in Cognitive Science 1*, pp. 236–258. Ellis Horwood, Chichester.

Robinson, J. J. (1975). Performance grammars. In Reddy, D. R. (Ed.), *Speech Recognition: Invited Paper Presented at the 1974 IEEE Symposium*, pp. 401–427. Academic Press.

Ryder, M. E. (1994). *Ordered Chaos: The Interpretation of English Noun-Noun Compounds*. University of California Press, Berkeley.

Schubert, L. K. and Pelletier, F. J. (1982). From English to logic: Context-free computation of ‘conventional’ logical translation. *American Journal of Computational Linguistics*, *8*(1), 27–44.

Sills, D. L. and Merton, R. K. (Eds.). (1991). *Social Science Quotations*. MacMillan, New York.

Simmons, R. F. (1973). Semantic networks: Their computation and use for understanding English sentences. In Schank, R. C. and Colby, K. M. (Eds.), *Computer Models of Thought and Language*, pp. 61–113. W.H. Freeman and Co., San Francisco.

Simmons, R. F. (1978). Rule-based computations on English. In Waterman, D. A. and Hayes-Roth, F. (Eds.), *Pattern-Directed Inference Systems*. Academic Press.

Simmons, R. F. (1983). *Computations from the English*. Prentice Hall.

ter Meulen, A. (1995). *Representing Time in Natural Language*. MIT Press.

van Lehn, K. (1978). Determining the scope of English quantifiers. Master’s thesis, MIT, Cambridge, MA. MIT Technical Report AI-TR-483.

Vanderwende, L. (1994). Algorithm for the automatic interpretation of noun sequences. In *COLING-94*, Kyoto, pp. 782–788.

Wilensky, R. and Arens, Y. (1980). PHRAN: A knowledge-based natural language understander. In *ACL-80*, Philadelphia, PA, pp. 117–121. ACL.

Wilks, Y. (1975a). An intelligent analyzer and understander of English. *Communications of the ACM*, *18*(5), 264–274.

Wilks, Y. (1975b). A preferential, pattern-seeking, semantics for natural language inference. *Artificial Intelligence*, *6*(1), 53–74.

Woods, W. A. (1977). Lunar rocks in natural English: Explorations in natural language question answering. In Zampolli, A. (Ed.), *Linguistic Structures Processing*, pp. 521–569. North Holland, Amsterdam.

Wu, D. (1992). *Automatic Inference: A Probabilistic Basis for Natural Language Interpretation*. Ph.D. thesis, University of California, Berkeley, Berkeley, CA. UCB/CSD 92-692.

Zernik, U. (1987). *Strategies in Language Acquisition: Learning Phrases from Examples in Context*. Ph.D. thesis, University of California, Los Angeles, Computer Science Department, Los Angeles, CA.

19 LEXICAL SEMANTICS

"When I use a word", Humpty Dumpty said in rather a scornful tone, "it means just what I choose it to mean – neither more nor less."

Lewis Carroll, *Alice in Wonderland*

How many legs does a dog have if you call its tail a leg?
Four.
Calling a tail a leg doesn't make it one.

Attributed to Abraham Lincoln

The previous two chapters focused on the representation of meaning representations for entire sentences. In those discussions, we made a simplifying assumption by representing *word meanings* as unanalyzed symbols like EAT or JOHN or RED. But representing the meaning of a word by capitalizing it is a pretty unsatisfactory model. In this chapter we introduce a richer model of the semantics of words, drawing on the linguistic study of word meaning, a field called **lexical semantics**. LEXICAL SEMANTICS

Before we try to define *word meaning* in the next section, we first need to be clear on what we mean by *word*, since we have used the word *word* in many different ways in this book.

LEXEME LEXICON We can use the word **lexeme** to mean a pairing of a particular form (orthographic or phonological) with its meaning, and a **lexicon** is a finite list of lexemes. For the purposes of lexical semantics, particularly for dictionaries and thesauruses, we represent a lexeme by a **lemma**. A **lemma** or **citation form** is the grammatical form that is used to represent a lexeme. This is often the base form; thus *carpet* is the lemma for *carpets*. The lemma or citation form for *sing*, *sang*, *sung* is *sing*. In many languages the infinitive form is used as the lemma for the verb; thus in Spanish *dormir* 'to sleep' is the lemma for verb forms like *duermes* 'you sleep'. The specific forms *sung* or *carpets* or *sing* or *duermes* are called **wordforms**. LEMMA CITATION FORM WORDFORMS

LEMMATIZATION

The process of mapping from a wordform to a lemma is called **lemmatization**. Lemmatization is not always deterministic, since it may depend on the context. For example, the wordform *found* can map to the lemma *find* (meaning ‘to locate’) or the lemma *found* (‘to create an institution’), as illustrated in the following WSJ examples:

(19.1) He has looked at 14 baseball and football stadiums and **found** that only one – private Dodger Stadium – brought more money into a city than it took out.

(19.2) Culturally speaking, this city has increasingly displayed its determination to **found** the sort of institutions that attract the esteem of Eastern urbanites.

In addition, lemmas are part-of-speech specific; thus the wordform *tables* has two possible lemmas, the noun *table* and the verb *table*.

One way to do lemmatization is via the morphological parsing algorithms of Ch. 3. Recall that morphological parsing takes a surface form like *cats* and produces *cat +PL*. But a lemma is not necessarily the same as the stem from the morphological parse. For example, the morphological parse of the word *celebrations* might produce the stem **celebrate** with the affixes *-ion* and *-s*, while the lemma for *celebrations* is the longer form **celebration**. In general lemmas may be larger than morphological stems (e.g., *New York* or *throw up*). The intuition is that we want to have a different lemma whenever we need to have a completely different dictionary entry with its own meaning representation; we expect to have *celebrations* and *celebration* share an entry, since the difference in their meanings is mainly just grammatical, but not necessarily to share one with *celebrate*.

In the remainder of this chapter, when we refer to the meaning (or meanings) of a ‘word’, we will generally be referring to a lemma rather than a wordform.

Now that we have defined the locus of word meaning, we will proceed to different ways to represent this meaning. In the next section we introduce the idea of **word sense** as the part of a lexeme that represents word meaning. In following sections we then describe ways of defining and representing these senses, as well as introducing the lexical semantic aspects of the events defined in Ch. 17.

19.1 Word Senses

The meaning of a lemma can vary enormously given the context. Consider these two uses of the lemma *bank*, meaning something like ‘financial institution’ and ‘sloping mound’, respectively:

(19.3) Instead, a *bank* can hold the investments in a custodial account in the client’s name.

(19.4) But as agriculture burgeons on the east *bank*, the river will shrink even more.

SENSE
WORD SENSE

We represent some of this contextual variation by saying that the lemma *bank* has two **senses**. A **sense** (or **word sense**) is a discrete representation of one aspect of the meaning of a word. Loosely following lexicographic tradition, we will represent each sense by placing a superscript on the orthographic form of the lemma as in **bank**$^{\mathbf{1}}$ and **bank**$^{\mathbf{2}}$. [1]

[1] Confusingly, the word “lemma” is itself very ambiguous; it is also sometimes used to mean these separate senses, rather than the citation form of the word. You should be prepared to see both uses in the literature.

The senses of a word might not have any particular relation between them; it may be almost coincidental that they share an orthographic form. For example, the *financial institution* and *sloping mound* senses of bank seem relatively unrelated. In such cases we say that the two senses are **homonyms**, and the relation between the senses is one of **homonymy**. Thus **bank**^{1} ('financial institution') and **bank**^{2} ('sloping mound') are homonyms.

HOMONYMS

HOMONYMY

Sometimes, however, there is some semantic connection between the senses of a word. Consider the following WSJ 'bank' example:

(19.5) While some *banks* furnish sperm only to married women, others are much less restrictive.

Although this is clearly not a use of the 'sloping mound' meaning of *bank*, it just as clearly is not a reference to a promotional giveaway at a financial institution. Rather, *bank* has a whole range of uses related to repositories for various biological entities, as in *blood bank*, *egg bank*, and *sperm bank*. So we could call this 'biological repository' sense **bank**^{3}. Now this new sense **bank**^{3} has some sort of relation to **bank**^{1}; both **bank**^{1} and **bank**^{3} are repositories for entities that can be deposited and taken out; in **bank**^{1} the entity is money, where in **bank**^{3} the entity is biological.

When two senses are related semantically, we call the relationship between them **polysemy** rather than homonymy. In many cases of polysemy the semantic relation between the senses is systematic and structured. For example consider yet another sense of *bank*, exemplified in the following sentence:

POLYSEMY

(19.6) The bank is on the corner of Nassau and Witherspoon.

This sense, which we can call **bank**^{4}, means something like 'the building belonging to a financial institution'. It turns out that these two kinds of senses (an organization, and the building associated with an organization) occur together for many other words as well (*school*, *university*, *hospital*, etc). Thus there is a systematic relationship between senses that we might represent as

BUILDING ↔ ORGANIZATION

METONYMY

This particular subtype of polysemy relation is often called **metonymy**. Metonymy is the use of one aspect of a concept or entity to refer to other aspects of the entity, or to the entity itself. Thus we are performing metonymy when we use the phrase *the White House* to refer to the administration whose office is in the White House.

Other common examples of metonymy include the relation between the following pairings of senses:

- Author (*Jane Austen wrote Emma*) ↔ Works of Author (*I really love Jane Austen*)
- Animal (*The chicken was domesticated in Asia*) ↔ Meat (*The chicken was overcooked*)
- Tree (*Plums have beautiful blossoms*) ↔ Fruit (*I ate a preserved plum yesterday*)

While it can be useful to distinguish polysemy from homonymy, there is no hard threshold for 'how related' two senses have to be to be considered polysemous. Thus the difference is really one of degree. This fact can make it very difficult to decide how many senses a word has, i.e., whether to make separate senses for closely related usages. There are various criteria for deciding that the differing uses of a word should be represented as distinct discrete senses. We might consider two senses discrete if they have independent truth conditions, different syntactic behavior, independent sense relations, or exhibit antagonistic meanings.

Consider the following uses of the verb *serve* from the WSJ corpus:

(19.7) They rarely *serve* red meat, preferring to prepare seafood, poultry or game birds.

(19.8) He *served* as U.S. ambassador to Norway in 1976 and 1977.

(19.9) He might have *served* his time, come out and led an upstanding life.

ZEUGMA

The *serve* of *serving red meat* and that of *serving time* clearly have different truth conditions and presuppositions; the *serve* of *serve as ambassador* has the distinct subcategorization structure *serve as NP*. These heuristic suggests that these are probably three distinct senses of *serve*. One practical technique for determining if two senses are distinct is to conjoin two uses of a word in a single sentence; this kind of conjunction of antagonistic readings is called **zeugma**. Consider the following ATIS examples:

(19.10) Which of those flights serve breakfast?

(19.11) Does Midwest Express serve Philadelphia?

(19.12) ?Does Midwest Express serve breakfast and Philadelphia?

We use (?) to mark example those that are semantically ill-formed. The oddness of the invented third example (a case of zeugma) indicates there is no sensible way to make a single sense of *serve* work for both breakfast and Philadelphia. We can use this as evidence that *serve* has two different senses in this case.

Dictionaries tend to use many fine-grained senses so as to capture subtle meaning differences, a reasonable approach given that traditional role of dictionaries in aiding word learners. For computational purposes, we often don't need these fine distinctions and so we may want to group or cluster the senses; we have already done this for some of the examples in this chapter.

HOMOPHONES

HOMOGRAPHS

We generally reserve the word **homonym** for two senses which share both a pronunciation and an orthography. A special case of multiple senses that causes problems especially for speech recognition and spelling correction is **homophones**. **Homophones** are senses that are linked to lemmas with the same pronunciation but different spellings, such as *wood*/*would* or *to*/*two*/*too*. A related problem for speech synthesis are **homographs** (Ch. 8). **Homographs** are distinct senses linked to lemmas with the same orthographic form but different pronunciations, such as these homographs of *bass*:

(19.13) The expert angler from Dora, Mo., was fly-casting for **bass** rather than the traditional trout.

(19.14) The curtain rises to the sound of angry dogs baying and ominous **bass** chords sounding.

How can we define the meaning of a word sense? Can we just look in a dictionary? Consider the following fragments from the definitions of *right*, *left*, *red*, and *blood* from the *American Heritage Dictionary* (Morris, 1985).

right	*adj.* located nearer the right hand esp. being on the right when facing the same direction as the observer.
left	*adj.* located nearer to this side of the body than the right.
red	*n.* the color of blood or a ruby.
blood	*n.* the red liquid that circulates in the heart, arteries and veins of animals.

Note the amount of circularity in these definitions. The definition of *right* makes two direct references to itself, while the entry for *left* contains an implicit self-reference in the phrase *this side of the body*, which presumably means the *left* side. The entries for *red* and *blood* avoid this kind of direct self-reference by instead referencing each other in their definitions. Such circularity is, of course, inherent in all dictionary definitions; these examples are just extreme cases. For humans, such entries are still useful since the user of the dictionary has sufficient grasp of these other terms to make the entry in question sensible.

For computational purposes, one approach to defining a sense is to make use of a similar approach to these dictionary definitions; defining a sense via its relationship with other senses. For example, the above definitions make it clear that *right* and *left* are similar kinds of lemmas that stand in some kind of alternation, or opposition, to one another. Similarly, we can glean that *red* is a color, it can be applied to both *blood* and *rubies*, and that *blood* is a *liquid*. **Sense relations** of this sort are embodied in on-line databases like **WordNet**. Given a sufficiently large database of such relations, many applications are quite capable of performing sophisticated semantic tasks (even if they do not *really* know their right from their left).

A second computational approach to meaning representation is to create a small finite set of semantic primitives, atomic units of meaning, and then create each sense definition out of these primitives. This approach is especially common when defining aspects of the meaning of *events* such as *semantic roles*.

We will explore both of these approaches to meaning in this chapter. In the next section we introduce various relations between senses, followed by a discussion of WordNet, a sense relation resource. We then introduce a number of meaning representation approaches based on semantic primitives such as semantic roles.

19.2 Relations between Senses

This section explores some of the relations that hold among word senses, focusing on a few that have received significant computational investigation: **synonymy**, **antonymy**, and **hypernymy**, as well as a brief mention of other relations like **meronymy**.

19.2.1 Synonymy and Antonymy

SYNONYM

When the meaning of two senses of two different words (lemmas) are identical or nearly identical we say the two senses are **synonyms**. Synonyms include such pairs as:

couch/sofa vomit/throw up filbert/hazelnut car/automobile

A more formal definition of synonymy (between words rather than senses) is that two words are synonymous if they are substitutable one for the other in any sentence without changing the truth conditions of the sentence. We often say in this case that the two words have the same **propositional meaning**.

PROPOSITIONAL MEANING

While substitutions between some pairs of words like *car*/*automobile* or *water*/H_2O are truth-preserving, the words are still not identical in meaning. Indeed, probably no two words are absolutely identical in meaning, and if we define synonymy as identical meanings and connotations in all contexts, there are probably no absolute synonyms. Many other facets of meaning that distinguish these words are important besides propositional meaning. For example H_2O is used in scientific contexts, and would be inappropriate in a hiking guide; this difference in genre is part of the meaning of the word. In practice the word *synonym* is therefore commonly used to describe a relationship of approximate or rough synonymy.

Instead of talking about two *words* being synonyms, in this chapter we will define synonymy (and other relations like hyponymy and meronymy) as a relation between senses rather than between words. We can see the usefulness of this by considering the words *big* and *large*. These may seem to be synonyms in the following ATIS sentences, in the sense that we could swap *big* and *large* in either sentence and retain the same meaning:

(19.15) How big is that plane?

(19.16) Would I be flying on a large or small plane?

But note the following WSJ sentence where we cannot substitute *large* for *big*:

(19.17) Miss Nelson, for instance, became a kind of big sister to Mrs. Van Tassel's son, Benjamin.

(19.18) ?Miss Nelson, for instance, became a kind of large sister to Mrs. Van Tassel's son, Benjamin.

That is because the word *big* has a sense that means being older, or grown up, while *large* lacks this sense. Thus it will be convenient to say that some senses of *big* and *large* are (nearly) synonymous while other ones are not.

ANTONYM

Synonyms are words with identical or similar meanings. **Antonyms**, by contrast, are words with opposite meaning such as the following:

long/short *big/little* *fast/slow* *cold/hot* *dark/light*
rise/fall *up/down* *in/out*

It is difficult to give a formal definition of antonymy. Two senses can be antonyms if they define a binary opposition, or are at opposite ends of some scale. This is the case for *long*/*short*, *fast/slow*, or *big*/*little*, which are at opposite ends of the *length* or *size* scale. Another groups of antonyms is **reversives**, which describe some sort of change or movement in opposite directions, such as *rise/fall* or *up/down*.

From one perspective, antonyms have very different meanings, since they are opposite. From another perspective, they have very similar meanings, since they share almost all aspects of their meaning except their position on a scale, or their direction. Thus automatically distinguishing synonyms from antonyms can be difficult.

19.2.2 Hyponymy

HYPONYM
HYPERNYM
SUPERORDINATE

One sense is a **hyponym** of another sense if the first sense is more specific, denoting a subclass of the other. For example, *car* is a hyponym of *vehicle*; *dog* is a hyponym of *animal*, and *mango* is a hyponym of *fruit*. Conversely, we say that *vehicle* is a **hypernym** of *car*, and *animal* is a hypernym of *dog*. It is unfortunate that the two words (hypernym and hyponym) are very similar and hence easily confused; for this reason the word **superordinate** is often used instead of **hypernym**.

superordinate	vehicle	fruit	furniture	mammal
hyponym	car	mango	chair	dog

We can define hypernymy more formally by saying that the class denoted by the superordinate extensionally includes the class denoted by the hyponym. Thus the class of animals includes as members all dogs, and the class of moving actions includes all walking actions. Hypernymy can also be defined in terms of entailment. Under this definition, a sense A is a hyponym of a sense B if everything that is A is also B and hence being an A entails being a B, or $\forall x \;\; A(x) \Rightarrow B(x)$. Hyponymy is usually a transitive relation; if A is a hyponym of B and B is a hyponym of C, then A is a hyponym of C.

ONTOLOGY
TAXONOMY

The concept of hyponymy is closely related to a number of other notions that play central roles in computer science, biology, and anthropology and computer science. The term **ontology** usually refers to a set of distinct objects resulting from an analysis of a domain, or **microworld**. A **taxonomy** is a particular arrangement of the elements of an ontology into a tree-like class inclusion structure. Normally, there are a set of well-formedness constraints on taxonomies that go beyond their component class inclusion relations. For example, the lexemes *hound*, *mutt*, and *puppy* are all hyponyms of *dog*, as are *golden retriever* and *poodle*, but it would be odd to construct a taxonomy from all those pairs since the concepts motivating the relations is different in each case. Instead, we normally use the word **taxonomy** to talk about the hypernymy relation between *poodle* and *dog*; by this definition **taxonomy** is a subtype of hypernymy.

19.2.3 Semantic Fields

MERONYMY
PART-WHOLE
MERONYM
HOLOYNM

So far we've seen the relations of synonymy, antonymy, hypernomy, and hyponymy. Another very common relation is **meronymy**, the **part-whole** relation. A *leg* is part of a *chair*; a *wheel* is part of a *car*. We say that *wheel* is a **meronym** of *car*, and *car* is a **holoynm** of *wheel*.

SEMANTIC FIELD

But there is a more general way to think about sense relations and word meaning. Where the relations we've defined so far have been binary relations between two senses, a **semantic field** is an attempt capture a more integrated, or holistic, relationship among entire sets of words from a single domain. Consider the following set of words extracted from the ATIS corpus:

reservation, flight, travel, buy, price, cost, fare, rates, meal, plane

We could assert individual lexical relations of hyponymy, synonymy, and so on between many of the words in this list. The resulting set of relations does not, however, add up to a complete account of how these words are related. They are clearly all defined with respect to a coherent chunk of common sense background information concerning air travel. Background knowledge of this kind has been studied under a variety of frameworks and is known variously as a **frame** (Fillmore, 1985), **model** (Johnson-Laird, 1983), or **script** (Schank and Albelson, 1977), and plays a central role in a number of computational frameworks.

We will discuss in Sec. 19.4.5 the **FrameNet** project (Baker et al., 1998), which is an attempt to provide a robust computational resource for this kind of frame knowledge. In the FrameNet representation, each of the words in the frame is defined with respect to the frame, and shares aspects of meaning with other frame words.

19.3 WordNet: A Database of Lexical Relations

WORDNET

The most commonly used resource for English sense relations is the **WordNet** lexical database (Fellbaum, 1998). WordNet consists of three separate databases, one each for nouns and verbs, and a third for adjectives and adverbs; closed class words are not included in WordNet. Each database consists of a set of lemmas, each one annotated with a set of senses. The WordNet 3.0 release has 117,097 nouns, 11,488 verbs, 22,141 adjectives, and 4,601 adverbs. The average noun has 1.23 senses, and the average verb has 2.16 senses. WordNet can be accessed via the web or downloaded and accessed locally.

GLOSS

Parts of a typical lemma entry for the noun and adjective *bass* are shown in Fig. 19.1. Note that there are 8 senses for the noun and 1 for the adjective, each of which has a **gloss** (a dictionary-style definition), a list of synonyms for the sense (called a **synset**), and sometimes also usage examples (as shown for the adjective sense). Unlike dictionaries, WordNet doesn't represent pronunciation, so doesn't distinguish the pronunciation [b ae s] in **bass**$^{\mathbf{4}}$, **bass**$^{\mathbf{5}}$, and **bass**$^{\mathbf{8}}$ from the other senses which have the pronunciation [b ey s].

SYNSET

The set of near-synonyms for a WordNet sense is called a **synset** (for **synonym set**); synsets are an important primitive in WordNet. The entry for *bass* includes synsets like *bass*^{1}, *deep*^{6}, or *bass*^{6}, *bass voice*^{1}, *basso*^{2}. We can think of a synset as representing a **concept** of the type we discussed in Ch. 17. Thus instead of representing concepts using logical terms, WordNet represents them as a lists of the word-senses that can be used to express the concept. Here's another synset example:

```
{chump, fish, fool, gull, mark, patsy, fall guy,
sucker, schlemiel, shlemiel, soft touch, mug}
```

The gloss of this synset describes it as *a person who is gullible and easy to take advantage of*. Each of the lexical entries included in the synset can, therefore, be used to express this concept. Synsets like this one actually constitute the senses associated with WordNet entries, and hence it is synsets, not wordforms, lemmas or individual senses, that participate in most of the lexical sense relations in WordNet.

Let's turn now to these these lexical sense relations, some of which are illustrated in Figures 19.2 and 19.3. For example the hyponymy relations in WordNet correspond directly to the notion of immediate hyponymy discussed on page 7. Each synset is related to its immediately more general and more specific synsets via direct hypernym

The noun “bass” has 8 senses in WordNet.
1. bass^1 - (the lowest part of the musical range)
2. bass^2, bass part^1 - (the lowest part in polyphonic music)
3. bass^3, basso^1 - (an adult male singer with the lowest voice)
4. sea bass^1, bass^4 - (the lean flesh of a saltwater fish of the family Serranidae)
5. freshwater bass^1, bass^5 - (any of various North American freshwater fish with
lean flesh (especially of the genus Micropterus))
6. bass^6, bass voice^1, basso^2 - (the lowest adult male singing voice)
7. bass^7 - (the member with the lowest range of a family of musical instruments)
8. bass^8 - (nontechnical name for any of numerous edible marine and
freshwater spiny-finned fishes)

The adjective “bass” has 1 sense in WordNet.
1. bass^1, deep^6 - (having or denoting a low vocal or instrumental range)
”a deep voice”; ”a bass voice is lower than a baritone voice”;
”a bass clarinet”

Figure 19.1 A portion of the WordNet 3.0 entry for the noun *bass*.

Relation	Also called	Definition	Example
Hypernym	Superordinate	From concepts to superordinates	*breakfast*^{1} → *meal*^{1}
Hyponym	Subordinate	From concepts to subtypes	*meal*^{1} → *lunch*^{1}
Member Meronym	Has-Member	From groups to their members	*faculty*^{2} → *professor*^{1}
Has-Instance		From concepts to instances of the concept	*composer*^{1} → *Bach*^{1}
Instance		From instances to their concepts	*Austen*^{1} → *author*^{1}
Member Holonym	Member-Of	From members to their groups	*copilot*^{1} → *crew*^{1}
Part Meronym	Has-Part	From wholes to parts	*table*^{2} → *leg*^{3}
Part Holonym	Part-Of	From parts to wholes	*course*^{7} → *meal*^{1}
Antonym		Opposites	*leader*^{1} → *follower*^{1}

Figure 19.2 Noun relations in WordNet.

Relation	Definition	Example
Hypernym	From events to superordinate events	*fly*^{9} → *travel*^{5}
Troponym	From a verb (event) to a specific manner elaboration of that verb	*walk*^{1} → *stroll*^{1}
Entails	From verbs (events) to the verbs (events) they entail	*snore*^{1} → *sleep*^{1}
Antonym	Opposites	*increase*^{1} ⟺ *decrease*^{1}

Figure 19.3 Verb relations in WordNet.

and hyponym relations. These relations can be followed to produce longer chains of more general or more specific synsets. Figure 19.4 shows hypernym chains for **bass^3** and **bass^7**.

In this depiction of hyponymy, successively more general synsets are shown on successive indented lines. The first chain starts from the concept of a human bass singer. Its immediate superordinate is a synset corresponding to the generic concept of a singer. Following this chain leads eventually to concepts such as *entertainer* and

```
Sense 3
bass, basso --
(an adult male singer with the lowest voice)
=> singer, vocalist, vocalizer, vocaliser
   => musician, instrumentalist, player
      => performer, performing artist
         => entertainer
            => person, individual, someone...
               => organism, being
                  => living thing, animate thing,
                     => whole, unit
                        => object, physical object
                           => physical entity
                              => entity
               => causal agent, cause, causal agency
                  => physical entity
                     => entity


Sense 7
bass --
(the member with the lowest range of a family of
musical instruments)
=> musical instrument, instrument
   => device
      => instrumentality, instrumentation
         => artifact, artefact
            => whole, unit
               => object, physical object
                  => physical entity
                     => entity
```

Figure 19.4 Hyponymy chains for two separate senses of the lemma *bass*. Note that the chains are completely distinct, only converging at the very abstract level *whole, unit*.

person. The second chain, which starts from musical instrument, has a completely different chain leading eventually to such concepts as musical instrument, device and physical object. Both paths do eventually join at the very abstract synset *whole, unit*, and then proceed together to *entity* which is the top (root) of the noun hierarchy (in WordNet this root is generally called the **unique beginner**).

UNIQUE BEGINNER

19.4 Event Participants: Semantic Roles and Selectional Restrictions

An important aspect of lexical meaning has to do with the semantics of events. When we discussed events in Ch. 17, we introduced the importance of predicate-argument structure for representing an event, and in particular the use of Davidsonian reification of events which let us represent each participant distinct from the event itself. We turn in this section to representing the meaning of these event *participants*. We introduce two kinds of semantic constraints on the arguments of event predicates: **semantic roles** and **selectional restrictions**, starting with a particular model of semantic roles called **thematic roles**.

19.4.1 Thematic Roles

Consider how we represented the meaning of arguments in Ch. 17 for sentences like these:

(19.19) Sasha broke the window.

(19.20) Pat opened the door.

A neo-Davidsonian event representation of these two sentences would be the following:

$$\begin{aligned}&\exists e,x,y\ Isa(e,Breaking)\wedge Breaker(e,Sasha)\\&\qquad\qquad\wedge BrokenThing(e,y)\wedge Isa(y,Window)\\&\exists e,x,y\ Isa(e,Opening)\wedge Opener(e,Pat)\\&\qquad\qquad\wedge OpenedThing(e,y)\wedge Isa(y,Door)\end{aligned}$$

DEEP ROLES

In this representation, the roles of the subjects of the verbs *break* and *open* are *Breaker* and *Opener* respectively. These **deep roles** are specific to each possible kind of event; *Breaking* events have *Breakers*, *Opening* events have *Openers*, *Eating* events have *Eaters*, and so on.

If we are going to be able to answer questions, perform inferences, or do any further kinds of natural language understanding of these events, we'll need to know a little more about the semantics of these arguments. *Breakers* and *Openers* have something in common. They are both volitional actors, often animate, and they have direct causal responsibility for their events.

THEMATIC ROLE

AGENTS

THEME

Thematic roles are one attempt to capture this semantic commonality between *Breakers* and *Eaters*. We say that the subjects of both these verbs are **agents**. Thus AGENT is the thematic role which represents an abstract idea such as volitional causation. Similarly, the direct objects of both these verbs, the *BrokenThing* and *OpenedThing*, are both prototypically inanimate objects which are affected in some way by the action. The thematic role for these participants is **theme**.

Thematic roles are one of the oldest linguistic models, proposed first by the Indian grammarian Panini sometime between the 7th and 4th centuries BCE. Their modern formulation is due to Fillmore (1968) and Gruber (1965). Although there is no universally agreed-upon set of thematic roles, Figures 19.5 and 19.6 present a list of some thematic roles which have been used in various computational papers, together with rough definitions and examples.

Thematic Role	Definition
AGENT	The volitional causer of an event
EXPERIENCER	The experiencer of an event
FORCE	The non-volitional causer of the event
THEME	The participant most directly affected by an event
RESULT	The end product of an event
CONTENT	The proposition or content of a propositional event
INSTRUMENT	An instrument used in an event
BENEFICIARY	The beneficiary of an event
SOURCE	The origin of the object of a transfer event
GOAL	The destination of an object of a transfer event

Figure 19.5 Some commonly-used thematic roles with their definitions.

Thematic Role	Example
AGENT	*The waiter* spilled the soup.
EXPERIENCER	*John* has a headache.
FORCE	*The wind* blows debris from the mall into our yards.
THEME	Only after Benjamin Franklin broke *the ice*...
RESULT	The French government has built a *regulation-size baseball diamond*...
CONTENT	Mona asked *"You met Mary Ann at a supermarket"?*
INSTRUMENT	He turned to poaching catfish, stunning them *with a shocking device*...
BENEFICIARY	Whenever Ann Callahan makes hotel reservations *for her boss*...
SOURCE	I flew in *from Boston*.
GOAL	I drove *to Portland*.

Figure 19.6 Some prototypical examples of various thematic roles.

19.4.2 Diathesis Alternations

The main reason computational systems use thematic roles, and semantic roles in general, is to act as a shallow semantic language that can let us make simple inferences that aren't possible from the pure surface string of words, or even the parse tree. For example, if a document says that *Company A acquired Company B*, we'd like to know that this answers the query *Was Company B acquired?* despite the fact that the two sentences have very different surface syntax. Similarly, this shallow semantics might act as a useful intermediate language in machine translation.

Thus thematic roles are used in helping us generalize over different surface realizations of predicate arguments. For example while the AGENT is often realized as the subject of the sentence, in other cases the THEME can be the subject. Consider these possible realizations of the thematic arguments of the verb *break*:

(19.21) *John* *broke the window.*
AGENT THEME

(19.22) *John broke the window with a rock.*
AGENT THEME INSTRUMENT

(19.23) *The rock broke the door.*
INSTRUMENT THEME

(19.24) *The window broke.*
THEME

(19.25) *The window was broken by John.*
THEME AGENT

The examples above suggest that *break* has (at least) the possible arguments AGENT, THEME, and INSTRUMENT. The set of thematic role arguments taken by a verb is often called the **thematic grid**, θ-grid, or **case frame**. We can also notice that there are (among others) the following possibilities for the realization of these arguments of *break*:

THEMATIC GRID
CASE FRAME

- AGENT:Subject, THEME:Object
- AGENT:Subject, THEME:Object , INSTRUMENT:PP_{with}
- INSTRUMENT:Subject, THEME:Object
- THEME:Subject

It turns out that many verbs allow their thematic roles to be realized in various syntactic positions. For example, verbs like *give* can realize the THEME and GOAL arguments in two different ways:

(19.26) a. *Doris gave the book to Cary.*
AGENT THEME GOAL
b. *Doris gave Cary the book.*
AGENT GOAL THEME

These multiple argument structure realizations (the fact that *break* can take AGENT, INSTRUMENT, or THEME as subject, and *give* can realize its THEME and GOAL in either order) are called **verb alternations** or **diathesis alternations**. The alternation we showed above *give*, the **dative alternation**, seems to occur with particular semantic classes of verbs, including "verbs of future having" (*advance*, *allocate*, *offer*, *owe*), "send verbs" (*forward*, *hand*, *mail*), "verbs of throwing" (*kick*, *pass*, *throw*), and so on. Levin (1993) is a reference book which lists for a large set of English verbs the semantic classes they belong to and the various alternations that they participate in. These lists of verb classes have been incorporated into the online resource VerbNet (Kipper et al., 2000).

VERB ALTERNATIONS
DIATHESIS ALTERNATIONS
DATIVE ALTERNATION

19.4.3 Problems with Thematic Roles

Representing meaning at the thematic role level seems like it should be useful in dealing with complications like diathesis alternations. But despite this potential benefit, it has proved very difficult to come up with a standard set of roles, and equally difficult to produce a formal definition of roles like AGENT, THEME, or INSTRUMENT.

For example, researchers attempting to define role sets often find they need to fragment a role like AGENT or THEME into many specific roles. Levin and Rappaport Hovav (2005) summarizes a number of such cases, such as the fact there seem to be at least two kinds of INSTRUMENTS, *intermediary* instruments that can appear as subjects and *enabling* instruments that cannot:

(19.28) The cook opened the jar with the new gadget.
(19.29) The new gadget opened the jar.
(19.30) Shelly ate the sliced banana with a fork.
(19.32) *The fork ate the sliced banana.

In addition to the fragmentation problem, there are cases where we'd like to reason about and generalize across semantic roles, but the finite discrete lists of roles don't let us do this.

Finally, it has proved very difficult to formally define the semantic roles. Consider the AGENT role; most cases of AGENTS are animate, volitional, sentient, causal, but any individual noun phrase might not exhibit all of these properties.

GENERALIZED SEMANTIC ROLES
PROTO-AGENT
PROTO-PATIENT

These problems have led most research to alternative models of semantic roles. One such model is based on defining **generalized semantic roles** that abstract over the specific thematic roles. For example PROTO-AGENT and PROTO-PATIENT are generalized roles that express roughly agent-like and roughly patient-like meanings. These roles are defined, not by necessary and sufficient conditions, but rather by a set of heuristic features that accompany more agent-like or more patient-like meanings. Thus the more an argument displays agent-like properties (intentionality, volitionality, causality, etc) the greater likelihood the argument can be labeled a PROTO-AGENT. The more patient-like properties (undergoing change of state, causally affected by another participant, stationary relative to other participants, etc), the greater likelihood the argument can be labeled a PROTO-PATIENT.

In addition to using proto-roles, many computational models avoid the problems with thematic roles by defining semantic roles that are specific to a particular verb, or specific to a particular set of verbs or nouns.

In the next two sections we will describe two commonly used lexical resources which make use of some of these alternative versions of semantic roles. **PropBank** uses both proto-roles and verb-specific semantic roles. **FrameNet** uses frame-specific semantic roles.

19.4.4 The Proposition Bank

PROPBANK

The **Proposition Bank**, generally referred to as **PropBank**, is a resource of sentences annotated with semantic roles. The English PropBank labels all the sentences in the Penn TreeBank; there is also a Chinese PropBank which labels sentences in the Penn Chinese TreeBank. Because of the difficulty of defining a universal set of thematic roles, the semantic roles in PropBank are defined with respect to an individual verb sense. Each sense of each verb thus has a specific set of roles, which are given only numbers rather than names: **Arg0**, **Arg1 Arg2**, and so on. In general, **Arg0** is used to represent the PROTO-AGENT, and **Arg1** the PROTO-PATIENT; the semantics of the other roles are specific to each verb sense. Thus the **Arg2** of one verb is likely to have nothing in common with the **Arg2** of another verb.

Here are some slightly simplified PropBank entries for one sense each of the verbs *agree* and *fall*; the definitions for each role ("Other entity agreeing", "amount fallen") are informal glosses intended to be read by humans, rather than formal definitions of the role.

(19.33) Frameset **agree.01**

Arg0:	Agreer
Arg1:	Proposition
Arg2:	Other entity agreeing
Ex1:	[$_{\text{Arg0}}$ The group] *agreed* [$_{\text{Arg1}}$ it wouldn't make an offer unless it had Georgia Gulf's consent].
Ex2:	[$_{\text{ArgM-Tmp}}$ Usually] [$_{\text{Arg0}}$ John] *agrees* [$_{\text{Arg2}}$ with Mary] [$_{\text{Arg1}}$ on everything].

(19.34) **fall.01** "move downward"

Arg1:	Logical subject, patient, thing falling
Arg2:	Extent, amount fallen
Arg3:	start point
Arg4:	end point, end state of arg1
ArgM-LOC:	medium
Ex1:	[$_{\text{Arg1}}$ Sales] *fell* [$_{\text{Arg4}}$ to $251.2 million] [$_{\text{Arg3}}$ from $278.7 million].
Ex1:	[$_{\text{Arg1}}$ The average junk bond] *fell* [$_{\text{Arg2}}$ by 4.2%] [$_{\text{ArgM-TMP}}$ in October].

Note that there is no Arg0 role for *fall*, because the normal subject of *fall* is a PROTO-PATIENT.

The PropBank semantic roles can be useful in recovering shallow semantic information about verbal arguments. Consider the verb *increase*:

(19.35) **increase.01** "go up incrementally"

Arg0:	causer of increase
Arg1:	thing increasing
Arg2:	amount increased by, EXT, or MNR
Arg3:	start point
Arg4:	end point

A PropBank semantic role labeling would allow us to infer the commonality in the event structures of the following three examples, showing that in each case *Big Fruit Co.* is the AGENT, and *the price of bananas* is the THEME, despite the differing surface forms.

(19.36) [$_{\text{Arg0}}$ Big Fruit Co.] increased [$_{\text{Arg1}}$ the price of bananas].
(19.37) [$_{\text{Arg1}}$ The price of bananas] was increased again [$_{\text{Arg0}}$ by Big Fruit Co.]
(19.38) [$_{\text{Arg1}}$ The price of bananas] increased [$_{\text{Arg2}}$ 5%].

19.4.5 FrameNet

While making inferences about the semantic commonalities across different sentences with *increase* is useful, it would be even more useful if we could make such inferences in many more situations, across different verbs, and also between verbs and nouns.

For example, we'd like to extract the similarity between these three sentences:

(19.39) [$_{\text{Arg1}}$ The price of bananas] increased [$_{\text{Arg2}}$ 5%].

(19.40) [$_{\text{Arg1}}$ The price of bananas] rose [$_{\text{Arg2}}$ 5%].

(19.41) There has been a [$_{\text{Arg2}}$ 5%] rise [$_{\text{Arg1}}$ in the price of bananas].

Note that the second example uses the different verb *rise*, and the third example uses the noun rather than the verb *rise*. We'd like a system to recognize that *the price of bananas* is what went up, and that *5%* is the amount it went up, no matter whether the *5%* appears as the object of the verb *increased* or as a nominal modifier of the noun *rise*.

FRAMENET

FRAME

FRAME ELEMENTS

The **FrameNet** project is another semantic role labeling project that attempts to address just these kinds of problems (Baker et al., 1998; Lowe et al., 1997; Ruppenhofer et al., 2006). Where roles in the PropBank project are specific to an individual verb, roles in the FrameNet project are specific to a **frame**. A **frame** is a script-like structure, which instantiates a set of frame-specific semantic roles called **frame elements**. Each word evokes a frame and profiles some aspect of the frame and its elements. For example, the **change_position_on_a_scale** frame is defined as follows:

> This frame consists of words that indicate the change of an Item's position on a scale (the Attribute) from a starting point (Initial_value) to an end point (Final_value).

Some of the semantic roles (frame elements) in the frame, separated into **core** roles and **non-core roles**, are defined as follows (definitions are taken from the FrameNet labelers guide (Ruppenhofer et al., 2006)).

	Core Roles
ATTRIBUTE	The ATTRIBUTE is a scalar property that the ITEM possesses.
DIFFERENCE	The distance by which an ITEM changes its position on the scale.
FINAL_STATE	A description that presents the ITEM's state after the change in the ATTRIBUTE's value as an independent predication.
FINAL_VALUE	The position on the scale where the Item ends up.
INITIAL_STATE	A description that presents the ITEM's state before the change in the ATTRIBUTE's value as an independent predication.
INITIAL_VALUE	The initial position on the scale from which the ITEM moves away.
ITEM	The entity that has a position on the scale.
VALUE_RANGE	A portion of the scale, typically identified by its end points, along which the values of the ATTRIBUTE fluctuate.
	Some Non-Core Roles
DURATION	The length of time over which the change takes place.
SPEED	The rate of change of the VALUE.
GROUP	The GROUP in which an ITEM changes the value of an ATTRIBUTE in a specified way.

Here are some example sentences:

(19.42) [$_{\text{ITEM}}$ Oil] *rose* [$_{\text{ATTRIBUTE}}$ in price] in price [$_{\text{DIFFERENCE}}$ by 2%].

(19.43) [$_{\text{ITEM}}$ It] has *increased* [$_{\text{FINAL\_STATE}}$ to having them 1 day a month].

(19.44) [$_{\text{ITEM}}$ Microsoft shares] *fell* [$_{\text{FINAL\_VALUE}}$ to 7 5/8].

(19.45) [$_{\text{ITEM}}$ Colon cancer incidence] *fell* [$_{\text{DIFFERENCE}}$ by 50%] [$_{\text{GROUP}}$ among men].

(19.46) a steady *increase* [$_{\text{INITIAL\_VALUE}}$ from 9.5] [$_{\text{FINAL\_VALUE}}$ to 14.3] [$_{\text{ITEM}}$ in dividends]

(19.47) a [$_{\text{DIFFERENCE}}$ 5%] [$_{\text{ITEM}}$ dividend] *increase*...

Note from these example sentences that the frame includes target words like *rise*, *fall*, and *increase*. In fact, the complete frame consists of the following words:

VERBS:
advance
climb
decline
decrease
diminish
dip
double
drop
dwindle
edge
explode
fall
fluctuate
gain
grow
increase
jump
move
mushroom
plummet
reach
rise
rocket
shift
skyrocket
slide
soar
swell
swing
triple
tumble

NOUNS:
decline
decrease
escalation
explosion
fall
fluctuation
gain
growth
hike
increase
rise
shift
tumble

ADVERBS:
increasingly

FrameNet also codes relationships between frames and frame elements. Frames can inherit from each other, and generalizations among frame elements in different frames can be captured by inheritance as well. Other relations between frames like causation are also represented. Thus there is a **Cause_change_of_position_on_a_scale** frame which is linked to the **Change_of_position_on_a_scale** frame by the **cause** relation, but adds an AGENT role and is used for causative examples such as the following:

(19.48) [$_{\text{AGENT}}$ They] *raised* [$_{\text{ITEM}}$ the price of their soda] [$_{\text{DIFFERENCE}}$ by 2%].

Together, these two frames would allow an understanding system to extract the common event semantics of all the verbal and nominal causative and non-causative usages.

Ch. 20 will discuss automatic methods for extracting various kinds of semantic roles; indeed one main goal of PropBank and FrameNet is to provide training data for such semantic role labeling algorithms.

19.4.6 Selectional Restrictions

Semantic roles gave us a way to express some of the semantics of an argument in its relation to the predicate. In this section we turn to another way to express semantic constraints on arguments. A **selectional restriction** is a kind of semantic type constraint that a verb imposes on the kind of concepts that are allowed to fill its argument roles. Consider the two meanings associated with the following example:

(19.49) I want to eat someplace that's close to ICSI.

There are two possible parses and semantic interpretations for this sentence. In the sensible interpretation *eat* is intransitive and the phrase *someplace that's close to* ICSI is an adjunct that gives the location of the eating event. In the nonsensical *speaker-as-Godzilla* interpretation, *eat* is transitive and the phrase *someplace that's close to* ICSI is the direct object and the THEME of the eating, like the NP *Malaysian food* in the following sentences:

(19.50) I want to eat Malaysian food.

How do we know that *someplace that's close to* ICSI isn't the direct object in this sentence? One useful cue is the semantic fact that the THEME of EATING events tends to be something that is *edible*. This restriction placed by the verb *eat* on the filler of its THEME argument, is called a **selectional restriction**. A selectional restriction is a constraint on the semantic type of some argument.

SELECTIONAL RESTRICTION

Selectional restrictions are associated with senses, not entire lexemes. We can see this in the following examples of the lexeme *serve*:

(19.51) Well, there was the time they served green-lipped mussels from New Zealand.

(19.52) Which airlines serve Denver?

Example (19.51) illustrates the cooking sense of *serve*, which ordinarily restricts its THEME to be some kind foodstuff. Example (19.52) illustrates the *provides a commercial service to* sense of *serve*, which constrains its THEME to be some type of appropriate location. We will see in Ch. 20 that the fact that selectional restrictions are associated with senses can be used as a cue to help in word sense disambiguation.

Selectional restrictions vary widely in their specificity. Note in the following examples that the verb *imagine* impose strict requirements on its AGENT role (restricting it to humans and other animate entities) but places very few semantic requirements on its THEME role. A verb like *diagonalize*, on the other hand, places a very specific constraint on the filler of its THEME role: it has to be a matrix, while the arguments of the adjectives *odorless* are restricted to concepts that could possess an odor.

(19.53) In rehearsal, I often ask the musicians to imagine a tennis game.

(19.54) I cannot even imagine what this lady does all day. Radon is a naturally occurring odorless gas that can't be detected by human senses.

(19.55) To diagonalize a matrix is to find its eigenvalues.

These examples illustrate that the set of concepts we need to represent selectional restrictions (being a matrix, being able to possess an oder, etc) is quite open-ended. This distinguishes selectional restrictions from other features for representing lexical knowledge, like parts-of-speech, which are quite limited in number.

Representing Selectional Restrictions

One way to capture the semantics of selectional restrictions is to use and extend the event representation of Ch. 17. Recall that the neo-Davidsonian representation of an event consists of a single variable that stands for the event, a predicate denoting the kind of event, and variables and relations for the event roles. Ignoring the issue of the λ-structures, and using thematic roles rather than deep event roles, the semantic contribution of a verb like *eat* might look like the following:

$$\exists e,x,y\ Eating(e) \wedge Agent(e,x) \wedge Theme(e,y)$$

With this representation, all we know about y, the filler of the THEME role, is that it is associated with an *Eating* event via the *Theme* relation. To stipulate the selectional restriction that y must be something edible, we simply add a new term to that effect:

$$\exists e,x,y\ Eating(e) \wedge Agent(e,x) \wedge Theme(e,y) \wedge Isa(y,EdibleThing)$$

```
Sense 1
hamburger, beefburger --
(a fried cake of minced beef served on a bun)
=> sandwich
   => snack food
      => dish
         => nutriment, nourishment, nutrition...
            => food, nutrient
               => substance
                  => matter
                     => physical entity
                        => entity
```

Figure 19.7 Evidence from WordNet that hamburgers are edible.

When a phrase like *ate a hamburger* is encountered, a semantic analyzer can form the following kind of representation:

$$\exists e,x,y\ Eating(e) \wedge Eater(e,x) \wedge Theme(e,y) \wedge Isa(y, EdibleThing)$$
$$\wedge Isa(y, Hamburger)$$

This representation is perfectly reasonable since the membership of *y* in the category *Hamburger* is consistent with its membership in the category *EdibleThing*, assuming a reasonable set of facts in the knowledge base. Correspondingly, the representation for a phrase such as *ate a takeoff* would be ill-formed because membership in an event-like category such as *Takeoff* would be inconsistent with membership in the category *EdibleThing*.

While this approach adequately captures the semantics of selectional restrictions, there are two practical problems with its direct use. First, using FOPC to perform the simple task of enforcing selectional restrictions is overkill. There are far simpler formalisms that can do the job with far less computational cost. The second problem is that this approach presupposes a large logical knowledge-base of facts about the concepts that make up selectional restrictions. Unfortunately, although such common sense knowledge-bases are being developed, none currently have the kind of scope necessary to the task.

A more practical approach is to state selectional restrictions in terms of WordNet synsets, rather than logical concepts. Each predicate simply specifies a WordNet synset as the selectional restriction on each of its arguments. A meaning representation is well-formed if the role filler word is a hyponym (subordinate) of this synset.

For our *ate a hamburger* example, for example, we could set the selectional restriction on the THEME role of the verb *eat* to the synset {**food, nutrient**}, glossed as *any substance that can be metabolized by an animal to give energy and build tissue*: Luckily, the chain of hypernyms for *hamburger* shown in Fig. 19.7 reveals that hamburgers are indeed food. Again, the filler of a role need not match the restriction synset exactly, it just needs to have the synset as one of its superordinates.

We can apply this approach to the THEME roles of the verbs *imagine*, *lift* and *di-*

agonalize, discussed earlier. Let us restrict *imagine*'s THEME to the synset {entity}, *lift*'s THEME to {physical entity} and *diagonalize* to {matrix}. This arrangement correctly permits *imagine a hamburger* and *lift a hamburger*, while also correctly ruling out *diagonalize a hamburger*.

Of course WordNet is unlikely to have the exactly relevant synsets to specify selectional restrictions for all possible words of English; other taxonomies may also be used. In addition, it is possible to learn selectional restrictions automatically from corpora.

We will return to selectional restrictions in Ch. 20 where we introduce the extension to **selectional preferences**, where a predicate can place probabilistic preferences rather than strict deterministic constraints on its arguments.

19.5 Primitive Decomposition

Back at the beginning of the chapter, we said that one way of defining a word is to decompose its meaning into a set of primitive semantics elements or features. We saw one aspect of this method in our discussion of finite lists of thematic roles (agent, patient, instrument, etc). We turn now to a brief discussion of how this kind of model, called **primitive decomposition**, or **componential analysis**, could be applied to the meanings of all words. Wierzbicka (1992, 1996) shows that this approach dates back at least to continental philosophers like Descartes and Leibniz.

SEMANTIC FEATURES

Consider trying to define words like *hen*, *rooster*, or *chick*. These words have something in common (they all describe chickens) and something different (their age and sex). This can be represented by using **semantic features**, symbols which represent some sort of primitive meaning:

hen	`+female, +chicken, +adult`
rooster	`-female, +chicken, +adult`
chick	`+chicken, -adult`

A number of studies of decompositional semantics, especially in the computational literature, have focused on the meaning of verbs. Consider these examples for the verb *kill*:

(19.56) Jim killed his philodendron.

(19.57) Jim did something to cause his philodendron to become not alive.

There is a truth-conditional ('propositional semantics') perspective from which these two sentences have the same meaning. Assuming this equivalence, we could represent the meaning of *kill* as:

(19.58) KILL(x,y) ⇔ CAUSE(x, BECOME(NOT(ALIVE(y))))

thus using semantic primitives like *do*, *cause*, *become not*, and *alive*.

Indeed, one such set of potential semantic primitives has been used to account for some of the verbal alternations discussed in Sec. 19.4.2 (Lakoff, 1965; Dowty, 1979). Consider the following examples.

(19.59) John opened the door. ⇒ (CAUSE(John(BECOME(OPEN(door)))))

(19.60) The door opened. ⇒ (BECOME(OPEN(door)))

(19.61) The door is open. ⇒ (OPEN(door))

The decompositional approach asserts that a single state-like predicate associated with *open* underlies all of these examples. The differences among the meanings of these examples arises from the combination of this single predicate with the primitives CAUSE and BECOME.

While this approach to primitive decomposition can explain the similarity between states and actions, or causative and non-causative predicates, it still relies on having a very large number of predicates like *open*. More radical approaches choose to break down these predicates as well. One such approach to verbal predicate decomposition is **Conceptual Dependency** (CD), a set of ten primitive predicates, shown in Fig. 19.8.

CONCEPTUAL DEPENDENCY

Primitive	Definition
ATRANS	The abstract transfer of possession or control from one entity to another.
PTRANS	The physical transfer of an object from one location to another
MTRANS	The transfer of mental concepts between entities or within an entity.
MBUILD	The creation of new information within an entity.
PROPEL	The application of physical force to move an object.
MOVE	The integral movement of a body part by an animal.
INGEST	The taking in of a substance by an animal.
EXPEL	The expulsion of something from an animal.
SPEAK	The action of producing a sound.
ATTEND	The action of focusing a sense organ.

Figure 19.8 A set of conceptual dependency primitives.

Below is an example sentence along with its CD representation. The verb *brought* is translated into the two primitives ATRANS and PTRANS to indicate the fact that the waiter both physically conveyed the check to Mary and passed control of it to her. Note that CD also associates a fixed set of thematic roles with each primitive to represent the various participants in the action.

(19.62) The waiter brought Mary the check.

$$\exists x,y\; Atrans(x) \wedge Actor(x,Waiter) \wedge Object(x,Check) \wedge To(x,Mary)$$
$$\wedge Ptrans(y) \wedge Actor(y,Waiter) \wedge Object(y,Check) \wedge To(y,Mary)$$

There are also sets of semantic primitives that cover more than just simple nouns and verbs. The following list comes from Wierzbicka (1996):

substantives:	I, YOU, SOMEONE, SOMETHING, PEOPLE
mental predicates:	THINK, KNOW, WANT, FEEL, SEE, HEAR
speech:	SAY
determiners and quantifiers:	THIS, THE SAME, OTHER, ONE, TWO, MANY (MUCH), ALL, SOME, MORE
actions and events:	DO, HAPPEN
evaluators:	GOOD, BAD
descriptors:	BIG, SMALL
time:	WHEN, BEFORE, AFTER
space:	WHERE, UNDER, ABOVE,
partonomy and taxonomy:	PART (OF), KIND (OF)
movement, existence, life:	MOVE, THERE IS, LIVE
metapredicates:	NOT, CAN, VERY
interclausal linkers:	IF, BECAUSE, LIKE
space:	FAR, NEAR, SIDE, INSIDE, HERE
time:	A LONG TIME, A SHORT TIME, NOW
imagination and possibility:	IF... WOULD, CAN, MAYBE

Because of the difficulty of coming up with a set of primitives that can represent all possible kinds of meanings, most current computational linguistic work does not use semantic primitives. Instead, most computational work tends to use the lexical relations of Sec. 19.2 to define words.

19.6 ADVANCED CONCEPTS: METAPHOR

METAPHOR

We use a **metaphor** when we refer to and reason about a concept or domain using words and phrases whose meanings come from a completely different domain. Metaphor is similar to **metonymy**, which we introduced as the use of one aspect of a concept or entity to refer to other aspects of the entity. In Sec. 19.1 we introduced metonymies like the following,

(19.63) Author (*Jane Austen wrote Emma*) ↔ Works of Author (*I really love Jane Austen*).

in which two senses of a polysemous word are systematically related. In metaphor, by contrast, there is a systematic relation between two completely different domains of meaning.

Metaphor is pervasive. Consider the following WSJ sentence:

(19.64) That doesn't scare Digital, which has grown to be the world's second-largest computer maker by poaching customers of IBM's mid-range machines.

The verb *scare* means 'to cause fear in', or 'to cause to lose courage'. For this sentence to make sense, it has to be the case that corporations can experience emotions like fear or courage as people do. Of course they don't, but we certainly speak of them and reason about them as if they do. We can therefore say that this use of *scare* is based on a metaphor that allows us to view a corporation as a person, which we will refer to the CORPORATION AS PERSON metaphor.

This metaphor is neither novel nor specific to this use of *scare*. Instead, it is a fairly conventional way to think about companies and motivates the use of *resuscitate*, *hemorrhage* and *mind* in the following WSJ examples:

(19.65) Fuqua Industries Inc. said Triton Group Ltd., a company it helped **resuscitate**, has begun acquiring Fuqua shares.

(19.66) And Ford was **hemorrhaging**; its losses would hit $1.54 billion in 1980.

(19.67) But if it changed its **mind**, however, it would do so for investment reasons, the filing said.

Each of these examples reflects an elaborated use of the basic CORPORATION AS PERSON metaphor. The first two examples extend it to use the notion of health to express a corporation's financial status, while the third example attributes a mind to a corporation to capture the notion of corporate strategy.

CONVENTIONAL METAPHORS

Metaphorical constructs such as CORPORATION AS PERSON are known as **conventional metaphors**. Lakoff and Johnson (1980) argue that many if not most of the metaphorical expressions that we encounter every day are motivated by a relatively small number of these simple conventional schemas.

19.7 Summary

This chapter has covered a wide range of issues concerning the meanings associated with lexical items. The following are among the highlights:

- **Lexical semantics** is the study of the meaning of words, and the systematic meaning-related connections between words.
- A **word sense** is the locus of word meaning; definitions and meaning relations are defined at the level of the word sense rather than wordforms as a whole.
- **Homonymy** is the relation between unrelated senses that share a form, while **polysemy** is the relation between related senses that share a form.
- **Synonymy** holds between different words with the same meaning.
- **Hyponymy** relations hold between words that are in a class-inclusion relationship.
- **Semantic fields** are used to capture semantic connections among groups of lexemes drawn from a single domain.
- **WordNet** is a large database of lexical relations for English words.
- **Semantic roles** abstract away from the specifics of deep semantic roles by generalizing over similar roles across classes of verbs.
- **Thematic roles** are a model of semantic roles based on a single finite list of roles. Other semantic role models include per-verb semantic roles lists and **proto-agent**/**proto-patient** both of which are implemented in **PropBank**, and per-frame role lists, implemented in **FrameNet**.
- Semantic **selectional restrictions** allow words (particularly predicates) to post constraints on the semantic properties of their argument words.

- **Primitive decomposition** is another way to represent the meaning of word, in terms of finite sets of sub-lexical primitives.

Bibliographical and Historical Notes

Cruse (2004) is a useful introductory linguistic text on lexical semantics. Levin and Rappaport Hovav (2005) is a research survey covering argument realization and semantic roles. Lyons (1977) is another classic reference. Collections describing computational work on lexical semantics can be found in Pustejovsky and Bergler (1992), Saint-Dizier and Viegas (1995) and Klavans (1995).

The most comprehensive collection of work concerning WordNet can be found in Fellbaum (1998). There have been many efforts to use existing dictionaries as lexical resources. One of the earliest was Amsler's (1980, 1981) use of the Merriam Webster dictionary. The machine readable version of Longman's Dictionary of Contemporary English has also been used (Boguraev and Briscoe, 1989). See Pustejovsky (1995), Pustejovsky and Boguraev (1996), Martin (1986) and Copestake and Briscoe (1995), inter alia, for computational approaches to the representation of polysemy. Pustejovsky's theory of the **Generative Lexicon**, and in particular his theory of the **qualia structure** of words, is another way of accounting for the dynamic systematic polysemy of words in context.

GENERATIVE LEXICON

QUALIA STRUCTURE

As we mentioned earlier, thematic roles are one of the oldest linguistic models, proposed first by the Indian grammarian Panini sometimes between the 7th and 4th centuries BCE. Their modern formulation is due to Fillmore (1968) and Gruber (1965). Fillmore's work had a large and immediate impact on work in natural language processing, as much early work in language understanding used some version of Fillmore's case roles (e.g., Simmons (1973, 1978, 1983)).

Work on selectional restrictions as a way of characterizing semantic well-formedness began with Katz and Fodor (1963). McCawley (1968) was the first to point out that selectional restrictions could not be restricted to a finite list of semantic features, but had to be drawn from a larger base of unrestricted world knowledge.

Lehrer (1974) is a classic text on semantic fields. More recent papers addressing this topic can be found in Lehrer and Kittay (1992). Baker et al. (1998) describe ongoing work on the FrameNet project.

The use of semantic primitives to define word meaning dates back to Leibniz; in linguistics, the focus on componential analysis in semantics was due to ? (?). See Nida (1975) for a comprehensive overview of work on componential analysis. Wierzbicka (1996) has long been a major advocate of the use of primitives in linguistic semantics; Wilks (1975) has made similar arguments for the computational use of primitives in machine translation and natural language understanding. Another prominent effort has been Jackendoff's Conceptual Semantics work (1983, 1990), which has also been applied in machine translation (Dorr, 1993, 1992).

Computational approaches to the interpretation of metaphor include convention-based and reasoning-based approaches. Convention-based approaches encode specific knowledge about a relatively small core set of conventional metaphors. These representations are then used during understanding to replace one meaning with an appropriate metaphorical one (Norvig, 1987; Martin, 1990; Hayes and Bayer, 1991; Veale and Keane, 1992; Jones and McCoy, 1992). Reasoning-based approaches eschew representing metaphoric conventions, instead modeling figurative language processing via general reasoning ability, such as analogical reasoning, rather than as a specifically language-related phenomenon. (Russell, 1976; Carbonell, 1982; Gentner, 1983; Fass, 1988, 1991, 1997).

An influential collection of papers on metaphor can be found in Ortony (1993). Lakoff and Johnson (1980) is the classic work on conceptual metaphor and metonymy. Russell (1976) presents one of the earliest computational approaches to metaphor. Additional early work can be found in DeJong and Waltz (1983), Wilks (1978) and Hobbs (1979). More recent computational efforts to analyze metaphor can be found in Fass (1988, 1991, 1997), Martin (1990), Veale and Keane (1992), Iverson and Helmreich (1992), and Chandler (1991). Martin (1996) presents a survey of computational approaches to metaphor and other types of figurative language.

STILL NEEDS SOME UPDATES.

EXERCISES

19.1 Collect three definitions of ordinary non-technical English words from a dictionary of your choice that you feel are flawed in some way. Explain the nature of the flaw and how it might be remedied.

19.2 Give a detailed account of similarities and differences among the following set of lexemes: *imitation*, *synthetic*, *artificial*, *fake*, and *simulated*.

19.3 Examine the entries for these lexemes in WordNet (or some dictionary of your choice). How well does it reflect your analysis?

19.4 The WordNet entry for the noun *bat* lists 6 distinct senses. Cluster these senses using the definitions of homonymy and polysemy given in this chapter. For any senses that are polysemous, give an argument as to how the senses are related.

19.5 Assign the various verb arguments in the following WSJ examples to their appropriate thematic roles using the set of roles shown in Figure 19.6.

a. The intense heat buckled the highway about three feet.

b. He melted her reserve with a husky-voiced paean to her eyes.

c. But Mingo, a major Union Pacific shipping center in the 1890s, has melted away to little more than the grain elevator now.

19.6 Using WordNet, describe appropriate selectional restrictions on the verbs *drink*, *kiss*, and *write*.

19.7 Collect a small corpus of examples of the verbs *drink*, *kiss*, and *write*, and analyze how well your selectional restrictions worked.

19.8 Consider the following examples from (McCawley, 1968):

My neighbor is a father of three.

?My buxom neighbor is a father of three.

What does the ill-formedness of the second example imply about how constituents satisfy, or violate, selectional restrictions?

19.9 Find some articles about business, sports, or politics from your daily newspaper. Identify as many uses of conventional metaphors as you can in these articles. How many of the words used to express these metaphors have entries in either WordNet or your favorite dictionary that directly reflect the metaphor.

19.10 Consider the following example:

> The stock exchange wouldn't talk publicly, but a spokesman said a news conference is set for today to introduce a new technology product.

Assuming that stock exchanges are not the kinds of things that can literally talk, give a sensible account for this phrase in terms of a metaphor or metonymy.

19.11 Choose an English verb that occurs in both FrameNet and PropBank. Compare and contrast the FrameNet and PropBank representations of the arguments of the verb.

Amsler, R. A. (1980). *The Structure of the Merriam-Webster Pocket Dictionary*. Ph.D. thesis, University of Texas, Austin, Texas. Report No.

Amsler, R. A. (1981). A taxonomy of English nouns and verbs. In *ACL-81*, Stanford, CA, pp. 133–138. ACL.

Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The Berkeley FrameNet project. In *COLING/ACL-98*, pp. 86–90.

Boguraev, B. and Briscoe, T. (Eds.). (1989). *Computational Lexicography for Natural Language Processing*. Longman, London.

Carbonell, J. (1982). Metaphor: An inescapable phenomenon in natural language comprehension. In Lehnert, W. G. and Ringle, M. (Eds.), *Strategies for Natural Language Processing*, pp. 415–434. Lawrence Erlbaum.

Chandler, S. (1991). Metaphor comprehension: A connectionist approach to implications for the mental lexicon. *Metaphor and Symbolic Activity*, *6*(4), 227–258.

Copestake, A. and Briscoe, T. (1995). Semi-productive polysemy and sense extension. *Journal of Semantics*, *12*(1), 15–68.

Cruse, D. A. (2004). *Meaning in Language: an Introduction to Semantics and Pragmatics*. Oxford University Press, Oxford. Second edition.

DeJong, G. F. and Waltz, D. L. (1983). Understanding novel language. *Computers and Mathematics with Applications*, *9*.

Dorr, B. (1992). The use of lexical semantics in interlingual machine translation. *Journal of Machine Translation*, *7*(3), 135–193.

Dorr, B. (1993). *Machine Translation*. MIT Press.

Dowty, D. R. (1979). *Word Meaning and Montague Grammar*. D. Reidel, Dordrecht.

Fass, D. (1988). *Collative Semantics: A Semantics for Natural Language*. Ph.D. thesis, New Mexico State University, Las Cruces, New Mexico. CRL Report No. MCCS-88-118.

Fass, D. (1991). met*: A method for discriminating metaphor and metonymy by computer. *Computational Linguistics*, *17*(1).

Fass, D. (1997). *Processing Metonymy and Metaphor*. Ablex Publishing, Greenwich, CT.

Fellbaum, C. (Ed.). (1998). *WordNet: An Electronic Lexical Database*. MIT Press.

Fillmore, C. J. (1968). The case for case. In Bach, E. W. and Harms, R. T. (Eds.), *Universals in Linguistic Theory*, pp. 1–88. Holt, Rinehart & Winston.

Fillmore, C. J. (1985). Frames and the semantics of understanding. *Quaderni di Semantica*, *VI*(2), 222–254.

Gentner, D. (1983). Structure mapping: A theoretical framework for analogy. *Cognitive Science*, *7*, 155–170.

Gruber, J. S. (1965). *Studies in Lexical Relations*. Ph.D. thesis, MIT.

Hayes, E. and Bayer, S. (1991). Metaphoric generalization through sort coercion. In *Proceedings of the 29th ACL*, Berkeley, CA, pp. 222–228. ACL.

Hobbs, J. R. (1979). Metaphor, metaphor schemata, and selective inferencing. Tech. rep. Technical Note 204, SRI.

Iverson, E. and Helmreich, S. (1992). Metallel: An integrated approach to non-literal phrase interpretation. *Computational Intelligence*, *8*(3).

Jackendoff, R. (1983). *Semantics and Cognition*. MIT Press.

Jackendoff, R. (1990). *Semantic Structures*. MIT Press.

Johnson-Laird, P. N. (1983). *Mental Models*. Harvard University Press, Cambridge, MA.

Jones, M. A. and McCoy, K. (1992). Transparently-motivated metaphor generation. In Dale, R., Hovy, E. H., Rösner, D., and Stock, O. (Eds.), *Aspects of Automated Natural Language Generation*, Lecture Notes in Artificial Intelligence 587, pp. 183–198. Springer Verlag, Berlin.

Katz, J. J. and Fodor, J. A. (1963). The structure of a semantic theory. *Language*, *39*, 170–210.

Kipper, K., Dang, H. T., and Palmer, M. (2000). Class-based construction of a verb lexicon. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX.

Klavans, J. L. (Ed.). (1995). *Representation and Acquisition of Lexical Knowledge: Polysemy, Ambiguity and Generativity*. AAAI Press, Menlo Park, CA. AAAI Technical Report SS-95-01.

Lakoff, G. (1965). *On the Nature of Syntactic Irregularity*. Ph.D. thesis, Indiana University. Published as *Irregularity in Syntax*. Holt, Rinehart, and Winston, New York, 1970.

Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press, Chicago, IL.

Lehrer, A. (1974). *Semantic Fields and Lexical Structure*. North-Holland, Amsterdam.

Lehrer, A. and Kittay, E. (Eds.). (1992). *Frames, Fields and Contrasts: New Essays in Semantic and Lexical Organization*. Lawrence Erlbaum.

Levin, B. (1993). *English Verb Classes And Alternations: A Preliminary Investigation*. University of Chicago Press, Chicago.

Levin, B. and Rappaport Hovav, M. (2005). *Argument Realization*. Cambridge University Press.

Lowe, J. B., Baker, C. F., and Fillmore, C. J. (1997). A frame-semantic approach to semantic annotation. In *Proceedings of ACL SIGLEX Workshop on Tagging Text with Lexical Semantics*, Washington, D.C., pp. 18–24. ACL.

Lyons, J. (1977). *Semantics*. Cambridge University Press.

Martin, J. H. (1986). The acquisition of polysemy. In *ICML 1986*, Irvine, CA, pp. 198–204.

Martin, J. H. (1990). *A Computational Model of Metaphor Interpretation*. Perspectives in Artificial Intelligence. Academic Press.

Martin, J. H. (1996). Computational approaches to figurative language. *Metaphor and Symbolic Activity*, *11*(1), 85–100.

McCawley, J. D. (1968). The role of semantics in a grammar. In Bach, E. W. and Harms, R. T. (Eds.), *Universals in Linguistic Theory*, pp. 124–169. Holt, Rinehart & Winston.

Morris, W. (Ed.). (1985). *American Heritage Dictionary* (2nd College Edition edition). Houghton Mifflin.

Nida, E. A. (1975). *Componential Analysis of Meaning: An Introduction to Semantic Structures*. Mouton, The Hague.

Norvig, P. (1987). *A Unified Theory of Inference for Text Understanding*. Ph.D. thesis, University of California, Berkeley, CA. Available as University of California at Berkeley Computer Science Division Tech. rep. #87/339.

Ortony, A. (Ed.). (1993). *Metaphor* (2nd edition). Cambridge University Press, Cambridge.

Pustejovsky, J. (1995). *The Generative Lexicon*. MIT Press.

Pustejovsky, J. and Bergler, S. (Eds.). (1992). *Lexical Semantics and Knowledge Representation*. Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin.

Pustejovsky, J. and Boguraev, B. (Eds.). (1996). *Lexical Semantics: The Problem of Polysemy*. Oxford University Press, Oxford.

Ruppenhofer, J., Ellsworth, M., Petruck, M. R. L., Johnson, C. R., and Scheffczyk, J. (2006). FrameNet ii: Extended theory and practice. Version 1.3, http://www.icsi.berkeley.edu/framenet/.

Russell, S. W. (1976). Computer understanding of metaphorically used verbs. *American Journal of Computational Linguistics*, *2*. Microfiche 44.

Saint-Dizier, P. and Viegas, E. (Eds.). (1995). *Computational Lexical Semantics*. Cambridge University Press.

Schank, R. C. and Albelson, R. P. (1977). *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum.

Simmons, R. F. (1973). Semantic networks: Their computation and use for understanding English sentences. In Schank, R. C. and Colby, K. M. (Eds.), *Computer Models of Thought and Language*, pp. 61–113. W.H. Freeman and Co., San Francisco.

Simmons, R. F. (1978). Rule-based computations on English. In Waterman, D. A. and Hayes-Roth, F. (Eds.), *Pattern-Directed Inference Systems*. Academic Press.

Simmons, R. F. (1983). *Computations from the English*. Prentice Hall.

Veale, T. and Keane, M. T. (1992). Conceptual scaffolding: A spatially founded meaning representation for metaphor comprehension. *Computational Intelligence*, *8*(3), 494–519.

Wierzbicka, A. (1992). *Semantics, Culture, and Cognition: University Human Concepts in Culture-Specific Configurations*. Oxford University Press.

Wierzbicka, A. (1996). *Semantics: Primes and Universals*. Oxford University Press.

Wilks, Y. (1975). An intelligent analyzer and understander of English. *Communications of the ACM*, *18*(5), 264–274.

Wilks, Y. (1978). Making preferences more active. *Artificial Intelligence*, *11*(3), 197–223.

20 COMPUTATIONAL LEXICAL SEMANTICS

To get a single right meaning is better than a ship-load of pearls,
To resolve a single doubt is like the bottom falling off the bucket.

Yuen Mei (1785) (translation by Arthur Waley)

The asphalt that Los Angeles is famous for occurs mainly on its freeways. But in the middle of the city is another patch of asphalt, the La Brea tar pits, and this asphalt preserves millions of fossil bones from the last of the Ice Ages of the Pleistocene Epoch. One of these fossils is the *Smilodon*, or sabre-toothed tiger, instantly recognizable by its long canines. Five million years ago or so, a completely different sabre-tooth tiger called *Thylacosmilus* lived in Argentina and other parts of South America. Thylacosmilus was a marsupial where Smilodon was a placental mammal, but had the same long upper canines and, like Smilodon, had a protective bone flange on the lower jaw. The similarity of these two mammals is one of many example of parallel or convergent evolution, in which particular contexts or environments lead to the evolution of very similar structures in different species (Gould, 1980).

The role of context is also important in the similarity of a less biological kind of organism: the word. Suppose we wanted to decide if two words have similar meanings. Not surprisingly, words with similar meanings often occur in similar contexts, whether in terms of corpora (having similar neighboring words or syntactic structures in sentences) or in terms of dictionaries and thesauruses (having similar definitions, or being nearby in the thesaurus hierarchy). Thus similarity of context turns out to be an important way to detect semantic similarity. Semantic similarity turns out to play an important roles in a diverse set of applications including information retrieval, question answering, summarization and generation, text classification, automatic essay grading and the detection of plagiarism.

In this chapter we introduce a series of topics related to computing with word meanings, or **computational lexical semantics**. Roughly in parallel with the sequence of topics in Ch. 19, we'll introduce computational tasks associated with word senses, relations among words, and the thematic structure of predicate-bearing words. We'll see the role of important role of context and similarity of sense in each of these.

We begin with **word sense disambiguation**, the task of examining word tokens in context and determining which sense of each word is being used. WSD is a task with a long history in computational linguistics, and as we will see, is a non-trivial undertaking given the somewhat elusive nature of many word senses. Nevertheless, there are robust algorithms that can achieve high levels of accuracy given certain reasonable assumptions. Many of these algorithms rely on contextual similarity to help choose the proper sense.

This will lead us natural to a consideration of the computation of **word similarity** and other relations between words, including the **hypernym**, **hyponym**, and **meronym** WordNet relations introduced in Ch. 19. We'll introduce methods based purely on corpus similarity, and others based on structured resources such as WordNet.

Finally, we describe algorithms for **semantic role labeling**, also known as **case role** or **thematic role assignment**. These algorithms generally use features extracted from syntactic parses to assign semantic roles such as AGENT, THEME and INSTRUMENT to the phrases in a sentence with respect to particular predicates.

20.1 WORD SENSE DISAMBIGUATION: OVERVIEW

Our discussion of compositional semantic analyzers in Ch. 18 pretty much ignored the issue of lexical ambiguity. It should be clear by now that this is an unreasonable approach. Without some means of selecting correct senses for the words in an input, the enormous amount of homonymy and polysemy in the lexicon would quickly overwhelm any approach in an avalanche of competing interpretations.

WORD SENSE DISAMBIGUATION

WSD

The task of selecting the correct sense for a word is called **word sense disambiguation**, or **WSD**. Disambiguating word senses has the potential to improve many natural language processing tasks. As we'll see in Ch. 25, **machine translation** is one area where word sense ambiguities can cause severe problems; others include **question-answering**, **information retrieval**, and **text classification**. The way that WSD is exploited in these and other applications varies widely based on the particular needs of the application. The discussion presented here ignores these application-specific differences and focuses on the implementation and evaluation of WSD systems as a stand-alone task.

In their most basic form, WSD algorithms take as input a word in context along with a fixed inventory of potential word senses, and return the correct word sense for that use. Both the nature of the input and the inventory of senses depends on the task. For machine translation from English to Spanish, the sense tag inventory for an English word might be the set of different Spanish translations. If speech synthesis is our task, the inventory might be restricted to homographs with differing pronunciations such as *bass* and *bow*. If our task is automatic indexing of medical articles, the sense tag inventory might be the set of MeSH (Medical Subject Headings) thesaurus entries. When we are evaluating WSD in isolation, we can use the set of senses from a dictionary/thesaurus resource like WordNet or LDOCE. Fig. 20.1 shows an example for the word *bass*, which can refer to a musical instrument or a kind of fish.[1]

[1] The WordNet database includes 8 senses; we have arbitrarily selected two for this example; we have also arbitrarily selected one of the many possible Spanish names for fishes which could be used to translate English *sea-bass*.

WordNet Sense	Spanish Translation	Roget Category	Target Word in Context
bass^4	lubina	FISH/INSECT	. . . fish as Pacific salmon and striped **bass** and. . .
bass^4	lubina	FISH/INSECT	. . . produce filets of smoked **bass** or sturgeon. . .
bass^7	bajo	MUSIC	. . . exciting jazz **bass** player since Ray Brown. . .
bass^7	bajo	MUSIC	. . . play **bass** because he doesn't have to solo. . .

Figure 20.1 Possible definitions for the inventory of sense tags for *bass*.

LEXICAL SAMPLE

It is useful to distinguish two variants of the generic WSD task. In the **lexical sample** task, a small pre-selected set of target words is chosen, along with an inventory of senses for each word from some lexicon. Since the set of words and the set of senses is small, **supervised machine learning** approaches are often used to handle lexical sample tasks. For each word, a number of corpus instances (context sentences) can be selected and hand-labeled with the correct sense of the target word in each. Classifier systems can then be trained using these labeled examples. Unlabeled target words in context can then be labeled using such a trained classifier. Early work in word sense disambiguation focused solely on lexical sample tasks of this sort, building word-specific algorithms for disambiguating single words like *line*, *interest*, or *plant*.

ALL-WORDS

In contrast, in the **all-words** task systems are given entire texts and a lexicon with an inventory of senses for each entry, and are required to disambiguate every content word in the text. The all-words task is very similar to part-of-speech tagging, except with a much larger set of tags, since each lemma has its own set. A consequence of this larger set of tags is a serious data sparseness problem; there is unlikely to be adequate training data for every word in the test set. Moreover, given the number of polysemous words in reasonably-sized lexicons, approaches based on training one classifier per term are unlikely to be practical.

In the following sections we explore the application of various machine learning paradigms to word sense disambiguation. We begin with supervised learning, followed by a section on how systems are standardly evaluated. We then turn to a variety of methods for dealing with the lack of sufficient day for fully-supervised training, including dictionary-based approaches and bootstrapping techniques.

Finally, after we have introduced the necessary notions of distributional word similarity in Sec. 20.7, we return in Sec. 20.10 to the problem of unsupervised approaches to sense disambiguation.

20.2 SUPERVISED WORD SENSE DISAMBIGUATION

If we have data which has been hand-labeled with correct word senses, we can use a **supervised learning** approach to the problem of sense disambiguation. extracting features from the text that are helpful in predicting particular senses, and then training a classifier to assign the correct sense given these features. The output of training is thus a classifier system capable of assigning sense labels to unlabeled words in context.

For **lexical sample** tasks, there are various labeled corpora for individual words, consisting of context sentences labeled with the correct sense for the target word. These include the *line-hard-serve* corpus containing 4,000 sense-tagged examples of *line* as a noun, *hard* as an adjective and *serve* as a verb (Leacock et al., 1993) , and the *interest* corpus with 2,369 sense-tagged examples of *interest* as a noun (Bruce and Wiebe, 1994). The SENSEVAL project has also produced a number of such sense-labeled lexical sample corpora (SENSEVAL-1 with 34 words from the HECTOR lexicon and corpus (Kilgarriff and Rosenzweig, 2000; Atkins, 1993), SENSEVAL-2 and -3 with 73 and 57 target words, respectively (Palmer et al., 2001; Kilgarriff, 2001)).

SEMANTIC CONCORDANCE

For training **all-word** disambiguation tasks we use a **semantic concordance**, a corpus in which each open-class word in each sentence is labeled with its word sense from a specific dictionary or thesaurus. One commonly used corpus is SemCor, a subset of the Brown Corpus consisting of over 234,000 words which were manually tagged with WordNet senses (Miller et al., 1993; Landes et al., 1998). In addition, sense-tagged corpora have been built for the SENSEVAL all-word tasks. The SENSEVAL-3 English all-words test data consisted of 2081 tagged content word tokens, from 5,000 total running words of English from the WSJ and Brown corpora (Palmer et al., 2001).

20.2.1 Extracting Feature Vectors for Supervised Learning

The first step in supervised training is to extract a useful set of features that are predictive of word senses. As Ide and Véronis (1998b) point out, the insight that underlies all modern algorithms for word sense disambiguation was first articulated by Weaver (1955) in the context of machine translation:

> If one examines the words in a book, one at a time as through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of the words. [. . .] But if one lengthens the slit in the opaque mask, until one can see not only the central word in question but also say N words on either side, then if N is large enough one can unambiguously decide the meaning of the central word. [. . .] The practical question is : “What minimum value of N will, at least in a tolerable fraction of cases, lead to the correct choice of meaning for the central word?”

FEATURE VECTOR

To extract useful features from such a window, a minimal amount of processing is first performed on the sentence containing the window. This processing varies from approach to approach but typically includes part-of-speech tagging, lemmatization or stemming, and in some cases syntactic parsing to reveal information such as head words and dependency relations. Context features relevant to the target word can then be extracted from this enriched input. A **feature vector** consisting of numeric or nominal values is used to encode this linguistic information as an input to most machine learning algorithms.

COLLOCATION

COLLOCATIONAL FEATURES

Two classes of features are generally extracted from these neighboring contexts: collocational features and bag-of-words features. A **collocation** is a word or phrase in a position-specific relationship to a target word (i.e., exactly one word to the right, or exactly 4 words to the left, and so on). Thus **collocational features** encode information about *specific* positions located to the left or right of the target word. Typical features extracted for these context words include the word itself, the root form of the word, and the word’s part-of-speech. Such features are effective at encoding local lexical and grammatical information that can often accurately isolate a given sense.

As an example of this type of feature-encoding, consider the situation where we need to disambiguate the word *bass* in the following WSJ sentence:

(20.1) An electric guitar and **bass** player stand off to one side, not really part of the scene, just as a sort of nod to gringo expectations perhaps.

A collocational feature-vector, extracted from a window of two words to the right and left of the target word, made up of the words themselves and their respective parts-of-speech, i.e.,

$$[w_{i-2}, \mathrm{POS}_{i-2}, w_{i-1}, \mathrm{POS}_{i-1}, w_{i+1}, \mathrm{POS}_{i+1}, w_{i+2}, \mathrm{POS}_{i+2}] \quad (20.2)$$

would yield the following vector:

```
[guitar, NN, and, CC, player, NN, stand, VB]
```

BAG-OF-WORDS

The second type of feature consists of **bag-of-words** information about neighboring words. A **bag-of-words** means an unordered set of words, ignoring their exact position. The simplest bag-of-words approach represents the context of a target word by a vector of features, each binary feature indicating whether a vocabulary word *w* does or doesn't occur in the context. This vocabulary is typically preselected as some useful subset of words in a training corpus. In most WSD applications, the context region surrounding the target word is generally a small symmetric fixed size window with the target word at the center. Bag-of-word features are effective at capturing the general topic of the discourse in which the target word has occurred. This, in turn, tends to identify senses of a word that are specific to certain domains. We generally don't use stop-words as features, and may also limit the bag-of-words to only consider a small number of frequently used content words.

For example a bag-of-words vector consisting of the 12 most frequent content words from a collection of *bass* sentences drawn from the WSJ corpus would have the following ordered word feature set:

[*fishing, big, sound, player, fly, rod, pound, double, runs, playing, guitar, band*]

Using these word features with a window size of 10, example (20.1) would be represented by the following binary vector:

```
[0,0,0,1,0,0,0,0,0,0,1,0]
```

We'll revisit the bag-of-words technique in Ch. 23 where we'll see that it forms the basis for the **vector space model** of search in modern search engines.

Most approaches to sense disambiguation use both collocational and bag-of-words features, either by joining them into one long vector, or by building a distinct classifier for each feature type, and combining them in some manner.

20.2.2 Naive Bayes and Decision List Classifiers

Given training data together with the extracted features, any supervised machine learning paradigm can be used to train a sense classifier. We will restrict our discussion here to the naive Bayes and decision list approaches, since they have been the focus of considerable work in word sense disambiguation and have not yet been introduced in previous chapters.

NAIVE BAYES CLASSIFIER

The **naive Bayes classifier** approach to WSD is based on the premise that choosing the best sense $\hat{s}$ out of the set of possible senses S for a feature vector $\vec{f}$ amounts to choosing the most probable sense given that vector. In other words:

$$\hat{s} = \operatorname*{argmax}_{s \in S} P(s|\vec{f}) \tag{20.3}$$

As is almost always the case, it would be difficult to collect reasonable statistics for this equation directly. To see this, consider that a simple binary bag of words vector defined over a vocabulary of 20 words would have 2^{20} possible feature vectors. It's unlikely that any corpus we have access to will provide coverage to adequately train this kind of feature vector. To get around this problem we first reformulate our problem in the usual Bayesian manner as follows:

$$\hat{s} = \operatorname*{argmax}_{s \in S} \frac{P(\vec{f}|s)P(s)}{P(\vec{f})} \tag{20.4}$$

Even this equation isn't helpful enough, since the data available that associates specific vectors $\vec{f}$ with each sense s is also too sparse. However, what is available in greater abundance in a tagged training set is information about individual feature-value pairs in the context of specific senses. Therefore, we can make the independence assumption that gives this method its name, and that has served us well in part-of-speech tagging, speech recognition, and probabilistic parsing — **naively** assume that the features are independent of one another. Making this assumption that the features are **conditionally independent given the word sense** yields the following approximation for $P(\vec{f}|s)$:

$$P(\vec{f}|s) \approx \prod_{j=1}^{n} P(f_j|s) \tag{20.5}$$

In other words, we can estimate the probability of an entire vector given a sense by the product of the probabilities of its individual features given that sense. Since $P(\vec{f})$ is the same for all possible senses, it does not effect the final ranking of senses, leaving us with the following formulation of a **naive Bayes classifier for WSD**:

$$\hat{s} = \operatorname*{argmax}_{s \in S} P(s) \prod_{j=1}^{n} P(f_j|s) \tag{20.6}$$

Given this equation, **training** a naive Bayes classifier consists of estimating each of these probabilities. (20.6) first requires an estimate for the prior probability of each sense $P(s)$. We get the maximum likelihood estimate of this probability from the sense-tagged training corpus by counting the number of times the sense s_i occurs and dividing by the total count of the target word w_j (i.e. the sum of the instances of each sense of the word). That is:

$$P(s_i) = \frac{\text{count}(s_i, w_j)}{\text{count}(w_j)} \tag{20.7}$$

We also need to know each of the individual feature probabilities $P(f_j|s)$. The maximum likelihood estimate for these would be:

$$P(f_j|s) = \frac{\text{count}(f_j, s)}{\text{count}(s)} \tag{20.8}$$

Thus, if a collocational feature such as $[w_{i-2} = \text{guitar}]$ occurred 3 times for sense bass^1, and sense bass^1 itself occurred 60 times in training, the MLE estimate is $P(f_j|s) = 0.05$. Binary bag-of-word features are treated in a similar manner; we simply count the number of times a given vocabulary item is present with each of the possible senses and divide by the count for each sense.

With the necessary estimates in place, we can assign senses to words in context by applying Equation (20.6). More specifically, we take the target word in context, extract the specified features, compute $P(s)\prod_{j=1}^{n} P(f_j|s)$ for each sense, and return the sense associated with the highest score. Note that in practice, the probabilities produced for even the highest scoring senses will be dangerously low due to the various multiplications involved; mapping everything to log-space and instead performing additions is the usual solution.

The use of a simple maximum likelihood estimator means that in testing, when a target word cooccurs with a word that it did not cooccur with in training, all of its senses will receive a probability of zero. Smoothing is therefore essential to the whole enterprise. Naive Bayes approaches to sense disambiguation generally use the simple Laplace (add-one or add-k) smoothing discussed in Ch. 4.

DECISION LIST CLASSIFIERS

One problem with naive Bayes and some other classifiers is that it's hard for humans to examine their workings and understand their decisions. Decision lists and decision trees are somewhat more transparent approaches that lend themselves to inspection. **Decision list classifiers** are equivalent to simple case statements in most programming languages. In a decision list classifier, a sequence of tests is applied to each target word feature vector. Each test is indicative of a particular sense. If a test succeeds, then the sense associated with that test is returned. If the test fails, then the next test in the sequence is applied. This continues until the end of the list, where a default test simply returns the majority sense.

Figure 20.2 shows a portion of a decision list for the task of discriminating the fish sense of *bass* from the music sense. The first test says that if the word *fish* occurs anywhere within the input context then **bass**^{1} is the correct answer. If it doesn't then each of the subsequent tests is consulted in turn until one returns true; as with case statements a default test that returns true is included at the end of the list.

Learning a decision list classifier consists of generating and ordering individual tests based on the characteristics of the training data. There are a wide number of methods that can be used to create such lists. In the approach used by Yarowsky (1994) for binary homonym discrimination, each individual feature-value pair constitutes a test. We can measure how much a feature indicates a particular sense by computing the log-likelihood of the sense given the feature. The ratio between the log-likelihoods of the two senses tells us how discriminative a feature is between senses:

$$\left| \text{Log}\left(\frac{P(Sense_1|f_i)}{P(Sense_2|f_i)}\right) \right| \tag{20.9}$$

The decision list is then created from these tests by simply ordering the tests in the list according to the log-likelihood ratio. Each test is checked in order and returns the appropriate sense. This training method differs quite a bit from standard decision list learning algorithms. For the details and theoretical motivation for these approaches see Rivest (1987) or Russell and Norvig (1995).

Rule		Sense
fish within window	⇒	**bass**^{1}
striped bass	⇒	**bass**^{1}
guitar within window	⇒	**bass**^{2}
bass player	⇒	**bass**^{2}
piano within window	⇒	**bass**^{2}
tenor within window	⇒	**bass**^{2}
sea bass	⇒	**bass**^{1}
play/V *bass*	⇒	**bass**^{2}
river within window	⇒	**bass**^{1}
violin within window	⇒	**bass**^{2}
salmon within window	⇒	**bass**^{1}
on bass	⇒	**bass**^{2}
bass are	⇒	**bass**^{1}

Figure 20.2 An abbreviated decision list for disambiguating the fish sense of bass from the music sense. Adapted from Yarowsky (1997).

20.3 WSD Evaluation, Baselines, and Ceilings

EXTRINSIC EVALUATION

IN VIVO

Evaluating component technologies like WSD is always a complicated affair. In the long term, we're primarily interested in the extent to which they improve performance in some end-to-end application such as information retrieval, question answering or machine translation. Evaluating component NLP tasks embedded in end-to-end applications is called **extrinsic evaluation**, **task-based** evaluation, **end-to-end** evaluation, or **in vivo** evaluation. It is only with extrinsic evaluation that we can tell if a technology such as WSD is working in the sense of actually improving performance on some real task.

Extrinsic evaluations are much more difficult and time-consuming to implement, however, since they require integration into complete working systems. Furthermore, an extrinsic evaluation may only tell us something about WSD in the context of the application, and may not generalize to other applications.

INTRINSIC

IN VITRO

SENSE ACCURACY

For these reasons, WSD systems are typically developed and evaluated intrinsically. In **intrinsic** or **in vitro** we treat a WSD component as if it were a stand-alone system operating independently of any given application. In this style of evaluation, systems are evaluated either using exact match **sense accuracy**: the percentage of words that are tagged identically with the hand-labeled sense tags in a test set; or with standard precision and recall measures if systems are permitted to pass on labeling some instances. In general, we evaluate using held out data from the same sense-tagged corpora that we used for training, such as the SemCor corpus discussed above, or the various corpora produced by the SENSEVAL effort.

Many aspects of sense evaluation have been standardized by the SENSEVAL/SEMEVAL efforts (Palmer et al., 2006; Kilgarriff and Palmer, 2000). This framework provides a shared task with training and testing materials along with sense inventories for all-words and lexical sample tasks in a variety of languages.

Whichever WSD task we are performing, we ideally need two additional measures to assess how well we're doing: a baseline measure to tell use how well we're doing as compared to relatively simple approaches, and a ceiling to tell us how close we are to optimal performance.

MOST FREQUENT SENSE

TAKE THE FIRST SENSE

The simplest baseline is to choose the **most frequent sense** for each word (Gale et al., 1992b) from the senses in a labeled corpus. For WordNet, this corresponds to the **take the first sense** heuristic, since senses in WordNet are generally ordered from most-frequent to least-frequent. WordNet sense frequencies come from the SemCor sense-tagged corpus described above.

Unfortunately, many WordNet senses do not occur in SemCor; these unseen senses are thus ordered arbitrarily after those that do. The four WordNet senses of the noun *plant*, for example, are as follows:

Freq	Synset	Gloss
338	plant^1, works, industrial plant	buildings for carrying on industrial labor
207	plant^2, flora, plant life	a living organism lacking the power of locomotion
2	plant^3	something planted secretly for discovery by another
0	plant^4	an actor situated in the audience whose acting is rehearsed but seems spontaneous to the audience

The most frequent sense baseline can be quite accurate, and is therefore often used as a default, to supply a word sense when a supervised algorithm has insufficient training data. A second commonly used baseline is the **Lesk algorithm**, discussed in the next section.

Human inter-annotator agreement is generally considered as a ceiling, or upper bound, for sense disambiguation evaluations. Human agreement is measured by comparing the annotations of two human annotators on the same data given the same tagging guidelines. The ceiling (inter-annotator agreement) for many all-words corpora using WordNet-style sense inventories seems to range from about 75% to 80% (Palmer et al., 2006). Agreement on more coarse grained, often binary, sense inventories is closer to 90% (Gale et al., 1992b).

PSEUDOWORDS

While using hand-labeled test sets is the best current method for evaluation, labeling large amounts of data is still quite expensive. For supervised approaches, we need this data anyhow for training so the effort to label large amounts of data seems justified. But for unsupervised algorithms like those we will discuss in Sec. 20.10, it would be nice to have an evaluation method that avoided hand labeling. The use of **pseudowords** is one such simplified evaluation method (Gale et al., 1992a; Schütze, 1992a). A pseudoword is an artificial word created by concatenating two randomly-chosen words together (e.g., *banana* and *door* to create *banana-door*.) Each occurrence of the two words in the test set is replaced by the new concatenation, creating a new 'word' which is now ambiguous between the senses *banana* and *door*. The 'correct sense' is defined by the original word, and so we can apply our disambiguation algorithm and compute accuracy as usual. In general, pseudowords give an overly optimistic measure of performance, since they are a bit easier to disambiguate than average ambiguous words. This is because the different senses of real words tend to be similar, while pseudowords are generally not semantically similar, acting like homonymous but not polysemous words (Gaustad, 2001). Nakov and Hearst (2003) shows that it is possible to improve the accuracy of pseudoword evaluation by more carefully choosing the pseudowords.

20.4 WSD: Dictionary and Thesaurus Methods

Supervised algorithms based on sense-labeled corpora are the best performing algorithms for sense disambiguation. However, such labeled training data is expensive and limited and supervised approaches fail on words not in the training data. Thus this section and the next describe different ways to get indirect supervision from other sources. In this section, we describe methods for using a dictionary or thesaurus as an indirect kind of supervision; the next section describes bootstrapping approaches.

20.4.1 The Lesk Algorithm

LESK ALGORITHM

SIMPLIFIED LESK

By far the most well-studied dictionary-based algorithm for sense disambiguation is the **Lesk algorithm**, really a family of algorithms that choose the sense whose dictionary gloss or definition shares the most words with the target word's neighborhood. Fig. 20.3 shows the simplest version of the algorithm, often called the **Simplified Lesk** algorithm (Kilgarriff and Rosenzweig, 2000).

```
function SIMPLIFIED LESK(word, sentence) returns best sense of word

  best-sense ← most frequent sense for word
  max-overlap ← 0
  context ← set of words in sentence
  for each sense in senses of word do
    signature ← set of words in the gloss and examples of sense
    overlap ← COMPUTEOVERLAP(signature, context)
    if overlap > max-overlap then
          max-overlap ← overlap
          best-sense ← sense
  end
  return(best-sense)
```

Figure 20.3 The Simplified Lesk Algorithm. The ComputeOverlap function returns the number of words in common between two sets, ignoring function words or other words on a stop list. The original Lesk algorithm defines the *context* in a more complex way. The *Corpus Lesk* algorithm weights each overlapping word w by its $-\log P(w)$, and includes labeled training corpus data in the *signature*.

As an example of the Lesk algorithm at work, consider disambiguating the word *bank* in the following context:

(20.10) The **bank** can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

given the following two WordNet senses:

bank^1	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	“he cashed a check at the bank”, “that bank holds the mortgage on my home”
bank^2	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	“they pulled the canoe up on the bank”, “he sat on the bank of the river and watched the currents”

Sense **bank**^{1} has two (non-stop) words overlapping with the context in (20.10): *deposits* and *mortgage*, while sense bank^2 has zero, so sense **bank**^{1} is chosen.

There are many obvious extensions to Simplified Lesk. The original Lesk algorithm (Lesk, 1986) is slightly more indirect. Instead of comparing a target word's signature with the context words, the target signature is compared with the signatures of each of the context words. For example, consider Lesk's example of selecting the appropriate sense of *cone* in the phrase *pine cone* given the following definitions for *pine* and *cone*.

pine	1	kinds of evergreen tree with needle-shaped leaves
	2	waste away through sorrow or illness
cone	1	solid body which narrows to a point
	2	something of this shape whether solid or hollow
	3	fruit of certain evergreen trees

In this example, Lesk's method would select **cone**^{3} as the correct sense since two of the words in its entry, *evergreen* and *tree*, overlap with words in the entry for *pine*, whereas neither of the other entries have any overlap with words in the definition of *pine*. In general Simplified Lesk seems to work better than original Lesk.

The primary problem with either the original or simplified approaches, however, is that the dictionary entries for the target words are short, and may not provide enough chance of overlap with the context.[2] One remedy is to expand the list of words used in the classifier to include words related to, but not contained in their individual sense definitions. But the best solution, if any sense-tagged corpus data like SemCor is available, is to add all the words in the labeled corpus sentences for a word sense into the signature for that sense. This version of the algorithm, the **Corpus Lesk** algorithm is the best-performing of all the Lesk variants (Kilgarriff and Rosenzweig, 2000; Vasilescu et al., 2004) and is used as a baseline in the SENSEVAL competitions. Instead of just counting up the overlapping words, the **Corpus Lesk** algorithm also applies a weight to each overlapping word. The weight is the **inverse document frequency** or **IDF**, a standard information-retrieval measure to be introduced in Ch. 23. IDF measures how many different ’documents’ (in this case glosses and examples) a word occurs in (Ch. 23) and is thus a way of discounting function words. Since function words like *the*, *of*, etc, occur in many documents, their IDF is very low, while the IDF of content words is high. Corpus Lesk thus uses IDF instead of a stoplist.

CORPUS LESK

INVERSE DOCUMENT FREQUENCY

IDF

Formally the IDF for a word *i* can be defined as

$$\text{idf}_i = \log\left(\frac{Ndoc}{nd_i}\right) \tag{20.11}$$

[2] Indeed, Lesk (1986) notes that the performance of his system seems to roughly correlate with the length of the dictionary entries.

where *Ndoc* is the total number of 'documents' (glosses and examples) and nd_i is the number of these documents containing word *i*.

Finally, it is possible to combine the Lesk and supervised approaches, by adding new Lesk-like bag-of-words features. For example, the glosses and example sentences for the target sense in WordNet could be used to compute the supervised bag-of-words features instead of (or in addition to) the words in the SemCor context sentence for the sense (Yuret, 2004).

20.4.2 Selectional Restrictions and Selectional Preferences

One of the earliest knowledge-sources for sense disambiguation is the notion of **selectional restrictions** defined in Ch. 19. For example the verb *eat* might have a restriction that its THEME argument be `[+FOOD]`. In early systems, selectional restrictions were used to rule out senses that violate the selectional restrictions of neighboring words (Katz and Fodor, 1963; Hirst, 1987). Consider the following pair of WSJ examples of the word *dish*:

(20.12) "In our house, everybody has a career and none of them includes washing **dishes**," he says.

(20.13) In her tiny kitchen at home, Ms. Chen works efficiently, stir-frying several simple **dishes**, including braised pig's ears and chicken livers with green peppers.

These correspond to WordNet **dish**^{1} (a piece of dishware normally used as a container for holding or serving food), with hypernyms like *artifact*, and **dish**^{2} (a particular item of prepared food) with hypernyms like *food*.

The fact that we perceive no ambiguity in these examples can be attributed to the selectional restrictions imposed by *wash* and *stir-fry* on their THEME semantic roles. The restrictions imposed by *wash* (perhaps [+WASHABLE]) conflict with **dish**^{2}. The restrictions on *stir-fry* ([+EDIBLE]) conflict with **dish**^{1}. In early systems, the predicate strictly selected the correct sense of an ambiguous argument by eliminating the sense that fails to match one of its selectional restrictions. But such hard constraints have a number of problems. The main problem is that selectional restriction violations often occur in well-formed sentences, either because they are negated as in (20.14), or because selectional restrictions are overstated as in (20.15):

(20.14) But it fell apart in 1931, perhaps because people realized you can't **eat** gold for lunch if you're hungry.

(20.15) In his two championship trials, Mr. Kulkarni **ate** glass on an empty stomach, accompanied only by water and tea.

As Hirst (1987) observes, examples like these often result in the elimination of all senses, bringing semantic analysis to a halt. Modern models thus adopt the view of selectional restrictions as preferences, rather than rigid requirements. Although there have been many instantiations of this approach over the years (e.g., Wilks, 1975c, 1975b, 1978), we'll discuss a member of the popular probabilistic or information-theoretic family of approaches: Resnik's (1997) model of **selectional association**.

SELECTIONAL PREFERENCE STRENGTH

Resnik first defines the **selectional preference strength** as the general amount of information that a predicate tells us about the semantic class of its arguments. For example, the verb *eat* tells us a lot about the semantic class of its direct object, since they tend to be edible. The verb *be*, by contrast, tells us less about its direct objects. The selectional preference strength can be defined by the difference in information between two distributions: the distribution of expected semantic classes $P(c)$ (how likely is it that a direct object will fall into class c) and the distribution of expected semantic classes for the particular verb $P(c|v)$ (how likely is it that the direct object of specific verb v will fall into semantic class c). The greater the difference between these distributions, the more information the verb is giving us about possible objects. This difference can be quantified by the **relative entropy** between these two distributions, or **Kullback-Leibler divergence** (Kullback and Leibler, 1951). The Kullback-Leibler or KL divergence $D(P||Q)$ can be used to express the difference between two probability distributions P and Q, and will be discussed further when we discuss word similarity in Equation (20.50).

RELATIVE ENTROPY

KULLBACK-LEIBLER DIVERGENCE

$$D(P||Q) = \sum_x P(x)\log\frac{P(x)}{Q(x)} \tag{20.16}$$

The selectional preference $S_R(v)$ uses the KL divergence to express how much information, in bits, the verb v expresses about the possible semantic class of its argument.

$$\begin{aligned} S_R(v) &= D(P(c|v)||P(c)) \\ &= \sum_c P(c|v)\log\frac{P(c|v)}{P(c)} \end{aligned} \tag{20.17}$$

SELECTIONAL ASSOCIATION

Resnik then defines the **selectional association** of a particular class and verb as the relative contribution of that class to the general selectional preference of the verb:

$$A_R(v,c) = \frac{1}{S_R(p)}P(c|v)\log\frac{P(c|v)}{P(c)} \tag{20.18}$$

The selectional association is thus a probabilistic measure of the strength of association between a predicate and a class dominating the argument to the predicate. Resnik estimates the probabilities for these associations by parsing a corpus, counting all the times each predicate occurs with each argument word, and assuming that each word is a partial observation of all the WordNet concepts containing the word. The following table from Resnik (1996) shows some sample high and low selectional associations for verbs and some WordNet semantic classes of their direct objects.

	Direct Object		Direct Object	
Verb	Semantic Class	Assoc	Semantic Class	Assoc
read	WRITING	6.80	ACTIVITY	-.20
write	WRITING	7.26	COMMERCE	0
see	ENTITY	5.79	METHOD	-0.01

Resnik (1998) shows that these selectional associations can be used to perform a limited form of word sense disambiguation. Roughly speaking the algorithm selects as the correct sense for an argument the one that has the highest selectional association between one of its ancestor hypernyms and the predicate.

While we have presented only the Resnik model of selectional preferences, there are other more recent models, using probabilistic methods and using other relations than just direct object; see the end of the chapter for a brief summary. In general, selectional restriction approaches perform as well as other unsupervised approaches at sense disambiguation, but not as well as Lesk or as supervised approaches.

20.5 Minimally Supervised WSD: Bootstrapping

BOOTSTRAPPING

YAROWSKY ALGORITHM

Both the supervised approach and the dictionary-based approach to WSD require large hand-built resources; supervised training sets in one case, large dictionaries in the other. We can instead use **bootstrapping** algorithms, often called **semi-supervised learning** or **minimally supervised learning**, which need only a very small hand-labeled training set. The most widely emulated bootstrapping algorithm for WSD is the **Yarowsky algorithm** (Yarowsky, 1995).

The goal of the Yarowsky algorithm is to learn a classifier for a target word (in a lexical-sample task). The algorithm is given a small seed-set Λ_0 of labeled instances of each sense, and a much larger unlabeled corpus V_0. The algorithm first trains an initial decision-list classifier on the seed-set Λ_0. It then uses this classifier to label the unlabeled corpus V_0. The algorithm then selects the examples in V_0 that it is most confident about, removes them, and adds them to the training set (call it now Λ_1). The algorithm then trains a new decision list classifier (a new set of rules) on Λ_1, and iterates by applying the classifier to the now-smaller unlabeled set V_1, extracting a new training set Λ_2 and so on. With each iteration of this process, the training corpus grows and the untagged corpus shrinks. The process is repeated until some sufficiently low error-rate on the training set is reached, or until no further examples from the untagged corpus are above threshold.

The key to any bootstrapping approach lies in its ability to create a larger training set from a small set of seeds. This requires an accurate initial set of seeds and a good confidence metric for picking good new examples to add to the training set. The confidence metric used by Yarowsky (1995) is the measure described earlier in Sec. 20.2.2, the log-likelihood ratio of the decision-list rule that classified the example.

ONE SENSE PER COLLOCATION

One way to generate the initial seeds is to hand-label a small set of examples (Hearst, 1991). Instead of hand-labeling, it is also possible to use a heuristic to automatically select accurate seeds. Yarowsky (1995) used the **One Sense per Collocation** heuristic, which relies on the intuition that certain words or phrases strongly associated with the target senses tend not to occur with the other sense. Yarowsky defines his seed set by choosing a single collocation for each sense. As an illustration of this technique, consider generating seed sentences for the fish and musical senses of *bass*. Without too much thought, we might come up with *fish* as a reasonable indicator of $\mathbf{bass}^1$, and *play* as a reasonable indicator of $\mathbf{bass}^2$. Figure 20.5 shows a partial result of such a search for the strings “fish” and “play” in a corpus of *bass* examples drawn from the WSJ.

We can also suggest collocates automatically, for example extracting words from machine readable dictionary entries, and selecting seeds using collocational statistics such as those described in Sec. 20.7 (Yarowsky, 1995).

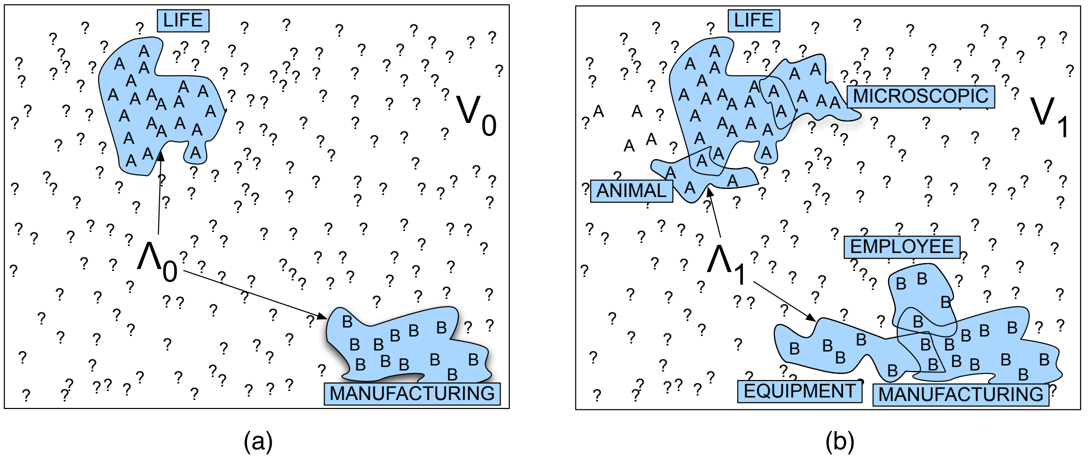


Figure 20.4 The Yarowsky algorithm disambiguating 'plant' at two stages; '?' indicates an unlabeled observation, A and B are observations labeled as SENSE-A or SENSE-B. 'LIFE' indicates observations occur with collocate "life". The initial stage (a) shows only seed sentences Λ_0 labeled by collocates ('life' and 'manufacturing'). An intermediate stage is shown in (b) where more collocates have been discovered ('equipment', 'microscopic', etc) and more instances in V_0 have been moved into Λ_1, leaving a smaller unlabeled set V_1. Figure adapted from Yarowsky (1995).

We need more good teachers – right now, there are only a half a dozen who can **play** the free **bass** with ease.

An electric guitar and **bass play**er stand off to one side, not really part of the scene, just as a sort of nod to gringo expectations perhaps.

When the New Jersey Jazz Society, in a fund-raiser for the American Jazz Hall of Fame, honors this historic night next Saturday, Harry Goodman, Mr. Goodman's brother and **bass play**er at the original concert, will be in the audience with other family members.

The researchers said the worms spend part of their life cycle in such **fish** as Pacific salmon and striped **bass** and Pacific rockfish or snapper.

And it all started when **fish**ermen decided the striped **bass** in Lake Mead were too skinny.

Though still a far cry from the lake's record 52-pound **bass** of a decade ago, "you could fillet these **fish** again, and that made people very, very happy," Mr. Paulson says.

Figure 20.5 Samples of *bass* sentences extracted from the WSJ using the simple correlates *play* and *fish*.

ONE SENSE PER DISCOURSE

The original Yarowsky algorithm also makes use of a second heuristic, called **One Sense Per Discourse**, based on the work of Gale et al. (1992c), who noticed that a particular word appearing multiple times in a text or discourse often appeared with the same sense. Yarowsky (1995), for example, showed in a corpus of 37,232 examples that every time the word *bass* occurred more than once in a discourse, that it occurred in only the *fish* or only the *music* coarse-grain sense throughout the discourse. The validity of this heuristic depends on the granularity of the sense inventory and is not valid in every discourse situation; it seems to be true mostly for coarse-grain senses, and particularly for cases of homonymy rather than polysemy (Krovetz, 1998). Nonetheless, it has still been useful in a number of unsupervised and semi-supervised sense disambiguation situations.

20.6 Word Similarity: Thesaurus Methods

We turn now to the computation of various semantic relations that hold between words. We saw in Ch. 19 that such relations include synonymy, antonymy, hyponymy, hypernymy, and meronymy. Of these, the one that has been most computationally developed and has the greatest number of applications is the idea of word **synonymy** and **similarity**.

WORD SIMILARITY
SEMANTIC DISTANCE

Synonymy is a binary relation between words; two words are either synonyms or not. For most computational purposes we use instead a looser metric of **word similarity** or **semantic distance**. Two words are more similar if they share more features of meaning, or are near-synonyms. Two words are less similar, or have greater semantic distance, if they have fewer common meaning elements. Although we have described them as relations between words, synonymy, similarity, and distance are actually relations between word *senses*. For example of the two senses of *bank*, we might say that the financial sense is similar to one of the senses of *fund* while the riparian sense is more similar to one of the senses of *slope*. In the next few sections of this chapter, we will need to compute these relations over both words and senses.

The ability to compute word similarity is a useful part of many language understanding applications. In **information retrieval** or **question answering** we might want to retrieve documents whose words have similar meanings to the query words. In **summarization**, **generation**, and **machine translation**, we need to know whether two words are similar to know if we can substitute one for the other in particular contexts. In **language modeling**, we can use semantic similarity to cluster words for class-based models. One interesting class of applications for word similarity is automatic grading of student responses. For example algorithms for **automatic essay grading** use word similarity to determine if an essay is similar in meaning to a correct answer. We can also use word-similarity as part of an algorithm to *take* an exam, such as a multiple-choice vocabulary test. Automatically taking exams is useful in test designs in order to see how easy or hard a particular multiple-choice question or exam is.

There are two classes of algorithms for measuring word similarity. This section focuses on **thesaurus-based** algorithms, in which we measure the distance between two senses in an on-line thesaurus like WordNet or MeSH. The next section focuses on **distributional** algorithms, in which we estimate word similarity by finding words that have similar distributions in a corpus.

The thesaurus-based algorithms use the structure of the thesaurus to define word similarity. In principle we could measure similarity using any information available in a thesaurus (meronymy, glosses, etc). In practice, however, thesaurus-based word similarity algorithms generally use only the hypernym/hyponym (*is-a* or subsumption) hierarchy. In WordNet, verbs and nouns are in separate hypernym hierarchies, so a thesaurus-based algorithm for WordNet can thus only compute noun-noun similarity, or verb-verb similarity; we can't compare nouns to verbs, or do anything with adjectives or other parts of speech.

WORD RELATEDNESS

Resnik (1995) and Budanitsky and Hirst (2001) draw the important distinction between **word similarity** and **word relatedness**. Two words are similar if they are near-synonyms, or roughly substitutable in context. Word relatedness characterizes a larger set of potential relationships between words; antonyms, for example, have high relatedness, but low similarity. The words *car* and *gasoline* are very related, but not similar, while *car* and *bicycle* are similar. Word similarity is thus a subcase of word relatedness. In general, the five algorithms we describe in this section do not attempt to distinguish between similarity and semantic relatedness; for convenience we will call them *similarity* measures, although some would be more appropriately described as relatedness measures; we return to this question in Sec. 20.8.

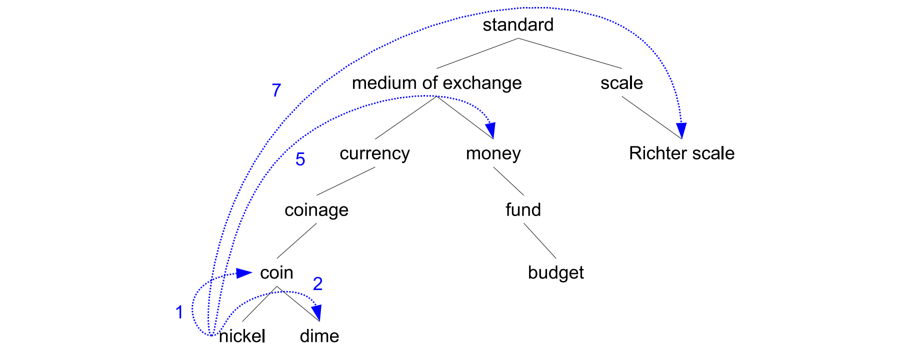


Figure 20.6 A fragment of the WordNet hypernym hierarchy, showing path lengths from *nickel* to *coin* (1), *dime* (2), *money* (5), and *Richter scale* (7).

The oldest and simplest thesaurus-based algorithms are based on the intuition that the shorter the **path** between two words or senses in the graph defined by the thesaurus hierarchy, the more similar they are. Thus a word/sense is very similar to its parents or its siblings, and less similar to words that are far away in the network. This notion can be operationalized by measuring the number of edges between the two concept nodes in the thesaurus graph. Fig. 20.6 shows an intuition; the concept *dime* is most similar to *nickel* and *coin*, less similar to *money*, and even less similar to *Richter scale*. Formally, we specify path length as follows:

> $\text{pathlen}(c_1, c_2) =$ the number of edges in the shortest path in the thesaurus graph between the sense nodes c_1 and c_2

PATH-LENGTH BASED SIMILARITY

Path-based similarity can be defined just as the path length, often with a log transform (Leacock and Chodorow, 1998), resulting in the following common definition of **path-length based similarity**:

$$\text{sim}_{\text{path}}(c_1, c_2) = -\log \text{pathlen}(c_1, c_2) \quad (20.19)$$

For most applications, we don't have sense-tagged data, and thus we need our algorithm to give us the similarity between words rather than between senses or concepts. For any of the thesaurus-based algorithms, following Resnik (1995), we can approximate the correct similarity (which would require sense disambiguation) by just using the pair of senses for the two words that results in maximum sense similarity. Thus WORD SIMILARITY based on sense similarity we can define **word similarity** as follows:

(20.20)
$$\text{wordsim}(w_1,w_2) = \max_{\substack{c_1 \in \text{senses}(w_1)\\ c_2 \in \text{senses}(w_2)}} \text{sim}(c_1,c_2)$$

The basic path-length algorithm makes the implicit assumption that each link in the network represents a uniform distance. In practice, this assumption is not appropriate. Some links (for example those that are very deep in the WordNet hierarchy) often seem to represent an intuitively narrow distance, while other links (e.g., higher up in the WordNet hierarchy) represent an intuitively wider distance. For example, in Fig. 20.6, the distance from *nickel* to *money* (5) seems intuitively much shorter than the distance from *nickel* to an abstract word *standard*; the link between *medium of exchange* and *standard* seems wider than that between, say, *coin* and *coinage*.

It is possible to refine path-based algorithms with normalizations based on depth in the hierarchy (Wu and Palmer, 1994), but in general we'd like an approach which lets us represent the distance associated with each edge independently.

INFORMATION CONTENT A second class of thesaurus-based similarity algorithms attempts to offer just such a fine-grained metric. These **information content word similarity** algorithms still rely on the structure of the thesaurus, but also add probabilistic information derived from a corpus.

Using similar notions to those we introduced earlier to define soft selectional restrictions, let's first define $P(c)$, following Resnik (1995), as the probability that a randomly selected word in a corpus is an instance of concept c (i.e., a separate random variable, ranging over words, associated with each concept). This implies that $P(root) = 1$, since any word is subsumed by the root concept. Intuitively, the lower a concept in the hierarchy, the lower its probability. We train these probabilities by counting in a corpus; each word in the corpus counts as an occurrence of each concept that contains it. For example, in Fig. 20.6 above, an occurrence of the word *dime* would count toward the frequency of *coin*, *currency*, *standard*, etc. More formally, Resnik computes $P(c)$ as follows:

(20.21)
$$P(c) = \frac{\sum_{w \in \text{words}(c)} count(w)}{N}$$

where $\text{words}(c)$ is the set of words subsumed by concept c, and N is the total number of words in the corpus that are also present in the thesaurus.

Fig. 20.7, from Lin (1998b), shows a fragment of the WordNet concept hierarchy augmented with the probabilities $P(c)$.

We now need two additional definitions. First, following basic information theory, we define the information content (IC) of a concept c as:

(20.22)
$$\text{IC}(c) = -\log P(c)$$

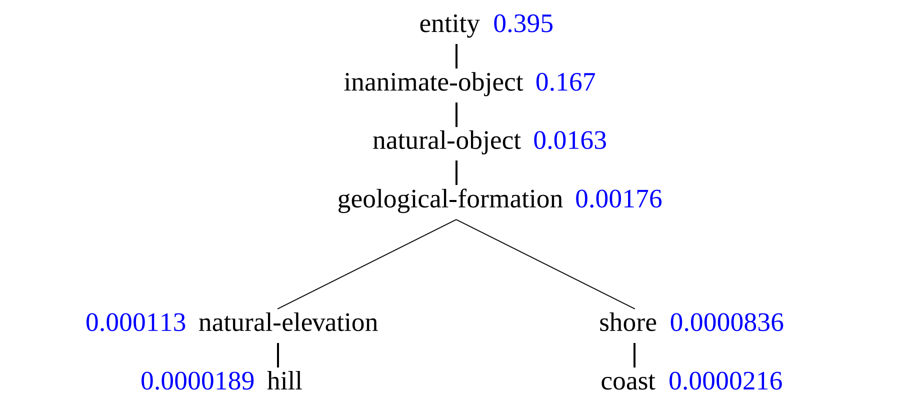


Figure 20.7 A fragment of the WordNet hierarchy, showing the probability $P(c)$ attached to each content, adapted from a figure from Lin (1998b)

LOWEST COMMON SUBSUMER
LCS

Second, we define the **lowest common subsumer** or **LCS** of two concepts:

> $\text{LCS}(c_1,c_2) =$ the lowest common subsumer, i.e., the lowest node in the hierarchy that subsumes (is a hypernym of) both c_1 and c_2

There are now a number of ways to use the information content of a node in a word similarity metric. The simplest way was first proposed by Resnik (1995). We think of the similarity between two words as related to their common information; the more two words have in common, the more similar they are. Resnik proposes to estimate the common amount of information by the **information content of the lowest common subsumer of the two nodes**. More formally, the **Resnik similarity** measure is:

RESNIK SIMILARITY

(20.23)
$$\text{sim}_{\text{resnik}}(c_1,c_2) = -\log P(\text{LCS}(c_1,c_2))$$

Lin (1998b) extended the Resnik intuition by pointing out that a similarity metric between objects A and B needs to do more than measure the amount of information in common between A and B. For example, he pointed out that in addition, the more **differences** between A and B, the less similar they are. In summary:

- **commonality**: the more information A and B have in common, the more similar they are.
- **difference**: the more differences between the information in A and B, the less similar they are

Lin measures the commonality between A and B as the information content of the proposition that states the commonality between A and B:

(20.24)
$$\text{IC(Common(A,B))}$$

He measures the difference between A and B as

(20.25)
$$\text{IC(description(A,B))} - \text{IC(common(A,B))}$$

where description(A,B) describes A and B. Given a few additional assumptions about similarity, Lin proves the following theorem:

> Similarity Theorem: The similarity between A and B is measured by the ratio between the amount of information needed to state the commonality of A and B and the information needed to fully describe what A and B are:

$$\text{sim}_{\text{Lin}}(A,B) = \frac{\log P(\text{common(A,B)})}{\log P(\text{description(A,B)})} \tag{20.26}$$

LIN SIMILARITY

Applying this idea to the thesaurus domain, Lin shows (in a slight modification of Resnik's assumption) that the information in common between two concepts is twice the information in the lowest common subsumer LCS(c_1,c_2). Adding in the above definitions of the information content of thesaurus concepts, the final **Lin similarity** function is:

$$\text{sim}_{\text{Lin}}(c_1,c_2) = \frac{2\times\log P(LCS(c_1,c_2))}{\log P(c_1)+\log P(c_2)} \tag{20.27}$$

For example, using sim_{lin}, Lin (1998b) shows that the similarity between the concepts of *hill* and *coast* from Fig. 20.7 is:

$$\text{sim}_{\text{Lin}}(\text{hill},\text{coast}) = \frac{2\times\log P(\text{geological-formation})}{\log P(\text{hill})+\log P(\text{coast}))} = 0.59 \tag{20.28}$$

JIANG-CONRATH DISTANCE

A very similar formula, **Jiang-Conrath distance** (Jiang and Conrath, 1997) (although derived in a completely different way from Lin, and expressed as a distance rather than similarity function) has been shown to work as well or better than all the other thesaurus-based methods:

$$\text{dist}_{\text{JC}}(c_1,c_2) = 2\times\log P(\text{LCS}(c_1,c_2)) - (\log P(c_1)+\log P(c_2)) \tag{20.29}$$

dist_{jc} can be transformed into a similarity by taking the reciprocal.

EXTENDED GLOSS OVERLAP

EXTENDED LESK

Finally, we describe a **dictionary-based** method, an extension of the Lesk algorithm for word-sense disambiguation described in Sec. 20.4.1. We call this a dictionary rather than a thesaurus method because it makes use of glosses, which are in general a property of dictionaries rather than thesauri (although WordNet does have glosses). Like the Lesk algorithm, the intuition of this **Extended Gloss Overlap**, or **Extended Lesk** measure (Banerjee and Pedersen, 2003) is that two concepts/senses in a thesaurus are similar if their glosses contain overlapping words. We'll begin by sketching an overlap function for two glosses. Consider these two concepts, with their glosses:

- *drawing paper:* paper that is specially prepared for use in drafting
- *decal:* the art of transferring designs from specially prepared paper to a wood or glass or metal surface.

For each *n*-word phrase that occurs in both glosses, Extended Lesk adds in a score of n^2 (the relation is non-linear because of the Zipfian relationship between lengths of phrases and their corpus frequencies; longer overlaps are rare so should be weighted more heavily). Here the overlapping phrases are *paper* and *specially prepared*, for a total similarity score of $1^2+2^2=5$.

Given such an overlap function, when comparing two concepts (synsets), Extended Lesk not only looks for overlap between their glosses, but also between the glosses of the senses which are hypernyms, hyponyms, meronyms, and other relations of the two concepts. For example if we just considered hyponyms, and defined gloss(hypo(A)) as the concatenation of all the glosses of all the hyponym senses of A, the total relatedness between two concepts A and B might be:

$$\begin{aligned}
\text{similarity(A,B)} &= \text{overlap(gloss(A), gloss(B))}\\
&\quad +\text{overlap(gloss(hypo(A)), gloss(hypo(B)))}\\
&\quad +\text{overlap(gloss(A), gloss(hypo(B)))}\\
&\quad +\text{overlap(gloss(hypo(A)),gloss(B))}
\end{aligned}$$

Let RELS be the set of possible WordNet relations whose glosses we compare; assuming a basic overlap measure as sketched above, we can then define the **Extended Lesk** overlap measure as:

$$\text{sim}_{\text{eLesk}}(c_1,c_2) = \sum_{r,q\in\text{RELS}} \text{overlap}(\text{gloss}(r(c_1)),\text{gloss}(q(c_2))) \tag{20.30}$$

$$\begin{aligned}
\text{sim}_{\text{path}}(c_1,c_2) &= -\log \text{pathlen}(c_1,c_2)\\
\text{sim}_{\text{Resnik}}(c_1,c_2) &= -\log P(\text{LCS}(c_1,c_2))\\
\text{sim}_{\text{Lin}}(c_1,c_2) &= \frac{2\times\log P(LCS(c_1,c_2))}{\log P(c_1)+\log P(c_2)}\\
\text{sim}_{\text{jc}}(c_1,c_2) &= \frac{1}{2\times\log P(\text{LCS}(c_1,c_2))-(\log P(c_1)+\log P(c_2))}\\
\text{sim}_{\text{eLesk}}(c_1,c_2) &= \sum_{r,q\in\text{RELS}} \text{overlap}(\text{gloss}(r(c_1)),\text{gloss}(q(c_2)))
\end{aligned}$$

Figure 20.8 Five thesaurus-based (and dictionary-based) similarity measures.

Fig. 20.8 summarizes the five similarity measures we have described in this section The publicly available `Wordnet::Similarity` package implementing all these and other thesaurus-based word similarity measures is described in Pedersen et al. (2004).

Evaluating Thesaurus-based Similarity Which of these similarity measures is best? Word similarity measures have been evaluated in two ways. One instrinic method is to compute the correlation coefficient between word similarity scores from an algorithm and word similarity ratings assigned by humans; such human ratings have been obtained for 65 word pairs by Rubenstein and Goodenough (1965), and 30 word pairs by

Miller and Charles (1991). Another more extrinsic evaluation method is to embed the similarity measure in some end application like detection of **malapropisms** (real-word spelling errors) (Budanitsky and Hirst, 2006; Hirst and Budanitsky, 2005), or other NLP applications like word-sense disambiguation (Patwardhan et al., 2003; McCarthy et al., 2004) and evaluate its impact on end-to-end performance. All of these evaluations suggest that all the above measures perform relatively well, and that of these, Jiang-Conrath similarity and Extended Lesk similarity are two of the best approaches, depending on the application.

20.7 WORD SIMILARITY: DISTRIBUTIONAL METHODS

The previous section showed how to compute similarity between any two senses in a thesaurus, and by extension between any two words in the thesaurus hierarchy. But of course we don't have such thesauri for every language. Even for languages where we do have such resources, thesaurus-based methods have a number of limitations. The obvious limitation is that thesauri often lack words, especially new or domain-specific words. In addition, thesaurus-based methods only work if rich hyponymy knowledge is present in the thesaurus. While we have this for nouns, hyponym information for verbs tends to be much sparser, and doesn't exist at all for adjectives and adverbs. Finally, it is more difficult with thesaurus-based methods to compare words in different hierarchies, such as nouns with verbs.

For these reasons, methods which can automatically extract synonyms and other word relations from corpora have been developed. In this section we introduce such **distributional** methods, which can be applied directly to supply a word relatedness measure for NLP tasks. Distributional methods can also be used for **automatic thesaurus generation** for automatically populating or augmenting on-line thesauruses like WordNet with new synonyms and, as we will see in Sec. 20.8, with other relations like hyponymy and meronymy.

The intuition of distributional methods is that the meaning of a word is related to the distribution of words around it; in the famous dictum of Firth (1957), "You shall know a word by the company it keeps!". Consider the following example, modified by Lin (1998a) from (?):

(20.31) A bottle of *tezgüino* is on the table.
Everybody likes *tezgüino*.
Tezgüino makes you drunk.
We make *tezgüino* out of corn.

The contexts in which *tezgüino* occurs suggest that it might be some kind of fermented alcoholic drink made from corn. The distributional method tries to capture this intuition by representing features of the context of *tezgüino* that might overlap with features of similar words like *beer*, *liquor*, *tequila*, and so on. For example such features might be occurs before *drunk* or occurs after *bottle* or is the direct object of *likes*.

FEATURE VECTOR

We can then represent a word w as a **feature vector** just as we saw with the bag-of-words features in Sec. 20.2. For example, suppose we had one binary feature f_i representing each of the N words in the lexicon v_i. The feature means w occurs in the neighborhood of word v_i, and hence takes the value 1 if w and v_i occur in some context window, and 0 otherwise. We could represent the meaning of word w as the feature vector

$$\vec{w} = (f_1, f_2, f_3, \cdots, f_N)$$

If w= *tezgüino*, v_1=*bottle*, v_2=*drunk*, and v_3=*matrix*, the co-occurrence vector for w from the corpus above would be:

$$\vec{w} = (1, 1, 0, \cdots)$$

Given two words represented by such sparse feature vectors, we can apply a vector distance measure and say that the words are similar if the two vectors are close by this measure. Fig. 20.9 shows an intuition about vector similarity for the four words *apricot*, *pineapple*, *digital*, and *information*. Based on the meanings of these four words, we would like a metric that shows *apricot* and *pineapple* to be similar, *digital* and *information*, to be similar, and the other four pairings to produce low similarity. For each word, Fig. 20.9 shows a short piece (8 dimensions) of the (binary) word co-occurrence vectors, computed from words that occur within a two-line context in the Brown corpus. The reader should convince themselves that the vectors for *apricot* and *pineapple* are indeed more similar than those of, say, *apricot* and *information*. For pedagogical purposes we've shown the context words that are particularly good at discrimination. Note that since vocabularies are quite large (10,000-100,000 words) and most words don't occur near each other in any corpus, real vectors are quite sparse.

	arts	**boil**	**data**	**function**	**large**	**sugar**	**summarized**	**water**
apricot	0	1	0	0	1	1	0	1
pineapple	0	1	0	0	1	1	0	1
digital	0	0	1	1	1	0	1	0
information	0	0	1	1	1	0	1	0

Figure 20.9 Co-occurrence vectors for four words, computed from the Brown corpus, showing only 8 of the (binary) dimensions (hand-picked for pedagogical purposes to show discrimination). Note that *large* occurs in all the contexts and *arts* occurs in none; a real vector would be extremely sparse.

Now that we have some intuitions, let's move on to examine the details of these measures. Specifying a distributional similarity measure requires that we specify three parameters: (1) how the co-occurrence terms are defined (i.e. what counts as a neighbor), (2) how these terms are weighted (binary? frequency? mutual information?) and (3) what vector distance metric we use (cosine? Euclidean distance?). Let's look at each of these requirements in the next three subsections.

20.7.1 Defining a Word's Co-occurrence Vectors

In our example feature vector, we used the feature w occurs in the neighborhood of word v_j. That is, for a vocabulary size N, each word w had N features, specifying whether vocabulary element v_j occurred in the neighborhood. Neighborhoods range from a small window of words (as few as one or two words on either side) to very large windows of ±500 words. In a minimal window, for example, we might have two features for each word v_j in the vocabulary, word v_k occurs immediately before word w and word v_k occurs immediately after word w.

To keep these contexts efficient, we often ignore very frequent words which tend not to be very discriminative, e.g., function words such as *a*, *am*, *the*, *of*, *1*, *2*, and so on. These removed words are called **stopwords** or the **stoplist**.

STOPWORDS

STOPLIST

Even with the removal of the stopwords, when used on very large corpora these co-occurrence vectors tend to be very large. Instead of using every word in the neighborhood, Hindle (1990) suggested choosing words that occur in some sort of **grammatical relation** or **dependency** to the target words. Hindle suggested that nouns which bear the same grammatical relation to the same verb might be similar. For example, the words *tea*, *water*, and *beer* are all frequent direct objects of the verb *drink*. The words *senate*, *congress*, *panel*, and *legislature* all tend to be subjects of the verbs *consider*, *vote*, and *approve*.

Hindle's intuition follows from the early work of Harris (1968), who suggested that:

> The meaning of entities, and the meaning of grammatical relations among them, is related to the restriction of combinations of these entities relative to other entities.

There have been a wide variety of realizations of Hindle's idea since then. In general, in these methods each sentence in a large corpus is parsed and a dependency parse is extracted. We saw in Ch. 12 lists of grammatical relations produced by dependency parsers, including noun-verb relations like subject, object, indirect object, and noun-noun relations like genitive, ncomp, and so on. A sentence like the following would result in the set of dependencies shown here:

(20.32) I discovered dried tangerines:

discover (subject I)
tangerine (obj-of discover)
dried (adj-mod-of tangerine)
I (subj-of discover)
tangerine (adj-mod dried)

Since each word can be in a variety of different dependency relations with other words, we'll need to augment the feature space. Each feature is now a pairing of a word and a relation, so instead of a vector of N features, we have a vector of $N \times R$ features, where R is the number of possible relations. Fig. 20.10 shows a schematic example of such a vector, taken from Lin (1998a), for the word *cell*. As the value of each attribute we have shown the frequency of the feature co-occurring with *cell*; the next section will discuss the use of what values and weights to use for each attribute.

Since full parsing is very expensive, it is common to use a chunker or shallow parser of the type defined in Sec. **??**, with the goal of extracting only a smaller set of relations like subject, direct object, and prepositional object of a particular preposition (Curran, 2003).

	subj-of, absorb	*subj-of*, adapt	*subj-of*, behave	...	*pobj-of*, inside	*pobj-of*, into	...	*nmod-of*, abnormality	*nmod-of*, anemia	*nmod-of*, architecture	...	*obj-of*, attack	*obj-of*, call	*obj-of*, come from	*obj-of*, decorate	...	*nmod*, bacteria	*nmod*, body	*nmod*, bone marrow
cell	1	1	1		16	30		3	8	1		6	11	3	2		3	2	2

Figure 20.10 Co-occurrence vector for the word *cell*, from Lin (1998a), showing grammatical function (dependency) features. Values for each attribute are frequency counts from a 64-million word corpus, parsed by an early version of MINIPAR.

20.7.2 Measures of Association with Context

ASSOCIATION

Now that we have a definition for the features or dimensions of a word's context vector, we are ready to discuss the values that should be associated with those features. These values are typically thought of as **weights** or measures of **association** between each target word w and a given feature f. In the example in Fig. 20.9, our association measure was a binary value for each feature, 1 if the relevant word had occurred in the context, 0 if not. In the example in Fig. 20.10, we used a richer association measure, the relative frequency with which the particular context feature had co-occurred with the target word.

Frequency, or probability, is certainly a better measure of association than just a binary value; features that occur often with a target word are more likely to be good indicators of the word's meaning. Let's define some terminology for implementing a probabilistic measure of association. For a target word w, each element of its co-occurrence vector is a feature f, consisting of a relation r and a related word w'; we can say $f = (r, w')$. For example, one of the features of the word *cell* in Fig. 20.10 is $f = (r, w')$ =(obj-of, *attack*). The probability of a feature f given a target word w is $P(f|w)$, for which the maximum likelihood estimate is:

$$P(f|w) = \frac{\text{count}(f,w)}{\text{count}(w)} \tag{20.33}$$

Similarly, the maximum likelihood estimate for the joint probability $P(f,w)$ is:

$$P(f,w) \;=\; \frac{\text{count}(f,w)}{\sum_{w'} \text{count}(w'))} \tag{20.34}$$

$P(w)$ and $P(f)$ are computed similarly.

Thus if we were to define simple probability as a measure of association it would look as follows:

$$\text{assoc}_{\text{prob}}(w,f) = P(f|w) \tag{20.35}$$

It turns out, however, that simple probability doesn't work as well as more sophisticated association schemes for word similarity.

Why isn't frequency or probability a good measure of association between a word and a context feature? Intuitively, if we want to know what kinds of contexts are shared by *apricot* and *pineapple* but not by *digital* and *information*, we're not going to get good discrimination from words like *the*, *it*, or *they*, which occur frequently with all sorts of words, and aren't informative about any particular word. We'd like context words which are particularly informative about the target word. We, therefore, need a weighting or measure of association which asks how much more often than chance that the feature co-occurs with the target word. As Curran (2003) points out, such a weighting is what we also want for finding good **collocations**, and so the measures of association used for weighting context words for semantic similarity are exactly the same measure used for finding a word's collocations.

COLLOCATIONS

One of the most important measures of association was first proposed by Church and Hanks (1989, 1990) and is based on the notion of **mutual information**. The **mutual information** between two random variables X and Y is

MUTUAL INFORMATION

$$I(X,Y) = \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)} \tag{20.36}$$

POINTWISE MUTUAL INFORMATION

The **pointwise mutual information** (Fano, 1961)[3] is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} \tag{20.37}$$

We can apply this intuition to co-occurrence vectors, by defining the pointwise mutual information association between a target word w and a feature f as:

$$\text{assoc}_{\text{PMI}}(w,f) = \log_2 \frac{P(w,f)}{P(w)P(f)} \tag{20.38}$$

The intuition of the PMI measure is that the numerator tells us how often we observed the two words together (assuming we compute probability using MLE as above). The denominator tells us how often we would **expect** the two words to co-occur assuming they each occurred independently, so their probabilities could just be multiplied. Thus the ratio gives us an estimate of how much more the target and feature co-occur than we expect by chance.

Since f is itself composed of two variables r and w', there is a slight variant on this model, due to Lin (1998a), that breaks down the expected value for $P(f)$ slightly differently; we'll call it the **Lin association measure** $\text{assoc}_{\text{Lin}}$, not to be confused with the WordNet measure sim_{Lin} that we discussed in the previous section:

LIN ASSOCIATION MEASURE

$$\text{assoc}_{\text{Lin}}(w,f) = \log_2 \frac{P(w,f)}{P(w)P(r|w)P(w'|w)} \tag{20.39}$$

For both $\text{assoc}_{\text{PMI}}$ and $\text{assoc}_{\text{Lin}}$, we generally only use the feature f for a word w if the assoc value is positive, since negative PMI values (implying things are co-

[3] Fano actually used the phrase *mutual information* to refer to what we now call *pointwise mutual information*, and the phrase *expectation of the mutual information* for what we now call *mutual information*; the term *mutual information* is still often used to mean *pointwise mutual information*.

Object	Count	PMI assoc	Object	Count	PMI assoc
bunch beer	2	12.34	wine	2	9.34
tea	2	11.75	water	7	7.65
Pepsi	2	11.75	anything	3	5.15
champagne	4	11.75	much	3	5.15
liquid	2	10.53	it	3	1.25
beer	5	10.20	<SOME AMOUNT>	2	1.22

Figure 20.11 Objects of the verb *drink*, sorted by PMI, from Hindle (1990).

occurring *less often* than we would expect by chance) tend to be unreliable unless the training corpora are enormous (Dagan et al., 1993; Lin, 1998a). In addition, when we are using the assoc-weighted features to compare two target words, we only use features that co-occur with both target words.

Fig. 20.11 from Hindle (1990) shows the difference between raw frequency counts and PMI-style association, for some direct objects of the verb *drink*.

T-TEST

One of the most successful association measures for word similarity attempts to capture the same intuition as mutual information, but uses the **t-test** statistic to measure how much more frequent the association is than chance. This measure was proposed for collocation-detection by Manning and Schütze (1999, Chapter 5) and then applied to word similarity by Curran and Moens (2002), Curran (2003).

The t-test statistic computes the difference between observed and expected means, normalized by the variance. The higher the value of t, the more likely we can reject the null hypothesis that the observed and expected means are the same.

$$t = \frac{\bar{x} - \mu}{\sqrt{\frac{s^2}{N}}} \tag{20.40}$$

When applied to association between words, the null hypothesis is that the two words are independent, and hence $P(f,w) = P(f)P(w)$ correctly models the relationship between the two words. We want to know how different the actual MLE probability $P(f,w)$ is from this null hypothesis value, normalized by the variance. Note the similarity to the comparison with the product model in the PMI measure above. The variance s^2 can be approximated by the expected probability $P(f)P(w)$ (see Manning and Schütze (1999)). Ignoring N (since it is constant), the resulting t-test association measure from Curran (2003) is thus:

$$\text{assoc}_{\text{t-test}}(w,f) = \frac{P(w,f) - P(w)P(f)}{\sqrt{P(f)P(w)}} \tag{20.41}$$

See the history section for a summary of various other weighting factors that have been tested on word similarity.

20.7.3 Defining similarity between two vectors

From the previous sections we can now compute a co-occurrence vector for a target word, with each co-occurrence feature weighted by an association measure, giving us a distributional definition of the meaning of a target word.

To define similarity between two target words v and w, we need a measure for taking two such vectors and giving a measure of vector similarity. Perhaps the simplest two measures of vector distance are the Manhattan and Euclidean distance. Fig. 20.12 shows a graphical intuition for Euclidean and Manhattan distance between two two-dimensional vectors $\vec{a}$ and $\vec{b}$. The **Manhattan distance**, also known as **Levenshtein distance** or **L1 norm**, is

MANHATTAN DISTANCE
LEVENSHTEIN DISTANCE
L1 NORM

$$\text{distance}_{\text{manhattan}}(\vec{x},\vec{y}) = \sum_{i=1}^{N} |x_i - y_i| \tag{20.42}$$

L2 NORM

The **Euclidean distance**, also called the **L2 norm**, was introduced in Ch. 9:

$$\text{distance}_{\text{euclidean}}(\vec{x},\vec{y}) = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2} \tag{20.43}$$

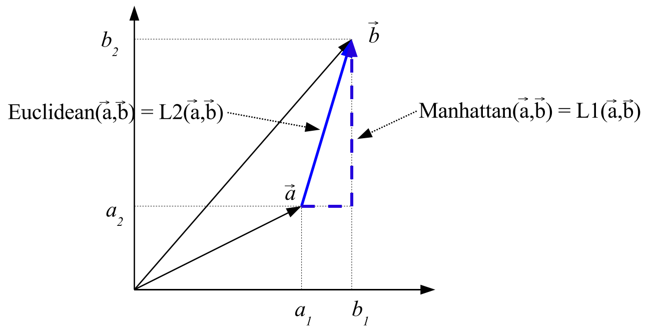

b_2
$\vec{b}$
Euclidean($\vec{a}$,$\vec{b}$) = L2($\vec{a}$,$\vec{b}$)
Manhattan($\vec{a}$,$\vec{b}$) = L1($\vec{a}$,$\vec{b}$)
$\vec{a}$
a_2
a_1
b_1

Figure 20.12 The Euclidean and Manhattan distance metrics for vectors $a = (a_1, a_2)$, and $b = (b_1, b_2)$, just to give the reader a grpahical intuition about the idea of distance between vectors; these particular metrics are generally not used for word similarity. See Ch. 9 for more on distance metrics.

Although the Euclidean and Manhattan distance metrics provide a nice geometric intuition for vector similarity and distance, these measures are rarely used for word similarity. This is because both measures turn out to be very sensitive to extreme values. Instead of these simple distance metrics, word similarity is based on closely related metrics from **information retrieval** and from **information theory**. The information retrieval methods seem to work better for word similarity, so we'll define a number of these in this section.

BINARY VECTOR
DOT PRODUCT
INNER PRODUCT

Let's begin with the intuition for a similarity metric in Fig. 20.9, in which the similarity between two binary vectors was just the number of features the two words had in common. If we assume a feature vector is a **binary vector**, we can define such a similarity metric as follows, using the **dot product** or **inner product** operator from linear algebra:

$$\text{sim}_{\text{dot-product}}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i \times w_i \tag{20.44}$$

In most cases, though, as we saw in the previous section, the values of our vector are not binary. Let's assume for the rest of this section that the entries in the co-occurrence vector are the **association** values between the target words and each of the features. In other words, let's define the vector for a target word $\vec{w}$ with N features $f_1..f_N$ as:

(20.45) $$\vec{w} = (\text{assoc}(w, f_1), \text{assoc}(w, f_2), \text{assoc}(w, f_3), \ldots, \text{assoc}(w, f_N))$$

Now we can apply $\text{sim}_{\text{dot-product}}$ to vectors with values defined as associations, to get the dot-product similarity between weighted values. This raw dot-product, however, has a problem as a similarity metric: it favors **long** vectors. The **vector length** is defined as:

VECTOR LENGTH

(20.46) $$|\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

A vector can be longer because it has more non-zero values, or because each dimension has a higher value. Both of these facts will increase the dot product. It turns out that both of these can occur as a by-product of word frequency. A vector from a very frequent word will have more non-zero co-occurrence association values, and will probably have higher values in each (even using association weights that control somewhat for frequency). The raw dot product thus favors frequent words.

We need to modify the dot product to normalize for the vector length. The simplest way is just to divide the dot product by the lengths of each of the two vectors. This **normalized dot product** turns out to be the same as the cosine of the angle between the two vectors. The **cosine** or normalized dot product similarity metric is thus:

NORMALIZED DOT PRODUCT

COSINE

(20.47) $$\text{sim}_{\text{cosine}}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i \times w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Because we have transformed the vectors to unit length, the cosine metric, unlike Euclidean or Manhattan distance, is no longer sensitive to long vectors from high-frequency words. The cosine value ranges from 1 for vectors pointing in the same direction, through 0 for vectors which are orthogonal (share no common terms), to -1 for vectors pointing in opposite directions, although in practice values tend to be positive.

Let's discuss two more similarity measures derived from information retrieval. The **Jaccard** (Jaccard, 1908, 1912) (also called **Tanimoto** or **min/max** (Dagan, 2000)) measure was originally designed for binary vectors. It was extended by Grefenstette (1994) to vectors of weighted associations as follows:

JACCARD

TANIMOTO

MIN/MAX

(20.48) $$\text{sim}_{\text{Jaccard}}(\vec{v}, \vec{w}) = \frac{\sum_{i=1}^{N} \min(v_i, w_i)}{\sum_{i=1}^{N} \max(v_i, w_i)}$$

The numerator of the Grefenstette/Jaccard function uses the min function, essentially computing the (weighted) number of overlapping features (since if either vector has a zero association value for an attribute, the result will be zero). The denominator can be viewed as a normalizing factor.

DICE

A very similar measure, the **Dice** measure, was similarly extended from binary vectors to vectors of weighted associations; one extension from Curran (2003) uses the Jaccard numerator, but uses as the denominator normalization factor the total weighted value of non-zero entries in the two vectors.

$$\text{sim}_{\text{Dice}}(\vec{v},\vec{w}) = \frac{2\times\sum_{i=1}^{N}\min(v_i,w_i)}{\sum_{i=1}^{N}(v_i+w_i)} \tag{20.49}$$

$$\begin{aligned}
\text{assoc}_{\text{prob}}(w,f) &= P(f|w) && (20.35)\\
\text{assoc}_{\text{PMI}}(w,f) &= \log_2\frac{P(w,f)}{P(w)P(f)} && (20.38)\\
\text{assoc}_{\text{Lin}}(w,f) &= \log_2\frac{P(w,f)}{P(w)P(r|w)P(w'|w)} && (20.39)\\
\text{assoc}_{\text{t-test}}(w,f) &= \frac{P(w,f)-P(w)P(f)}{\sqrt{P(f)P(w)}} && (20.41)
\end{aligned}$$

$$\begin{aligned}
\text{sim}_{\text{cosine}}(\vec{v},\vec{w}) &= \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N}v_i\times w_i}{\sqrt{\sum_{i=1}^{N}v_i^2}\sqrt{\sum_{i=1}^{N}w_i^2}} && (20.47)\\
\text{sim}_{\text{Jaccard}}(\vec{v},\vec{w}) &= \frac{\sum_{i=1}^{N}\min(v_i,w_i)}{\sum_{i=1}^{N}\max(v_i,w_i)} && (20.48)\\
\text{sim}_{\text{Dice}}(\vec{v},\vec{w}) &= \frac{2\times\sum_{i=1}^{N}\min(v_i,w_i)}{\sum_{i=1}^{N}(v_i+w_i)} && (20.49)\\
\text{sim}_{\text{JS}}(\vec{v}||\vec{w}) &= D(\vec{v}|\tfrac{\vec{v}+\vec{w}}{2})+D(\vec{w}|\tfrac{\vec{v}+\vec{w}}{2}) && (20.52)
\end{aligned}$$

Figure 20.13 Defining word similarity: measures of association between a target word w and a feature $f=(r,w')$ to another word w', and measures of vector similarity between word co-occurrence vectors $\vec{v}$ and $\vec{w}$.

Finally, there is a family of information-theoretic distributational similarity measures, (Pereira et al., 1993; Dagan et al., 1994, 1999; Lee, 1999), also based on the conditional probability association measure $P(f|w)$. The intuition of these models is that two vectors $\vec{v}$ and $\vec{w}$ are similar to the extent that their probability distributions $P(f|w)$ and $P(f|v)$ are similar. The basis of comparing two probability distributions P and Q is the **Kullback-Leibler divergence** or **KL divergence** or **relative entropy** (Kullback and Leibler, 1951) :

KL DIVERGENCE

$$D(P||Q) \;=\; \sum_x P(x)\log\frac{P(x)}{Q(x)} \tag{20.50}$$

Unfortunately, the KL-divergence is undefined when $Q(x)=0$ and $P(x)\neq 0$, which is a problem since these word distribution vectors are generally quite sparse. One alternative (Lee, 1999) is to use the **Jenson-Shannon divergence**, which represents the divergence of each distribution from the mean of the two, and doesn't have this problem with zeros:

JENSON-SHANNON DIVERGENCE

$$JS(P||Q) \;=\; D(P|\frac{P+Q}{2})+D(Q|\frac{P+Q}{2}) \tag{20.51}$$

Rephrased in terms of vectors $\vec{v}$ and $\vec{w}$,

$$\text{sim}_{\text{JS}}(\vec{v}||\vec{w}) = D(\vec{v}|\frac{\vec{v}+\vec{w}}{2}) + D(\vec{w}|\frac{\vec{v}+\vec{w}}{2}) \tag{20.52}$$

Fig. 20.13 summarizes the measures of association and of vector similarity that we have designed. See the history section for a summary of other vector similarity measures.

Finally, let's look at some of the results of distributional word similarity. The following are the ten most similar words to the different parts of speech of *hope* and *brief*, derived using the online dependency-based similarity tool (Lin, 2007); this tool defines the co-occurrence vector using all minipar grammatical relations, uses the $\text{assoc}_{\text{Lin}}$ measure of association, and a vector similarity metric from Lin (1998a).

- **hope (N):** optimism 0.141338, chance 0.136681, expectation 0.136559, prospect 0.125597, dream 0.119079, desire 0.117939, fear 0.116273, effort 0.111264, confidence 0.109136, promise 0.108269
- **hope (V):** would like 0.157988, wish 0.139532, plan 0.139349, say 0.136786, believe 0.135058, think 0.132673, agree 0.129985, wonder 0.129709, try 0.127047, decide 0.125387,
- **brief (N):** legal brief 0.139177, affidavit 0.103401, filing 0.0982636, petition 0.0864875, document 0.0835244, argument 0.0831851, letter 0.0785654, rebuttal 0.077766, memo 0.0768226, article 0.0758248
- **brief (A):** lengthy 0.256242, hour-long 0.191421, short 0.173561, extended 0.163085, frequent 0.162555, recent 0.15815, short-lived 0.154955, Prolonged 0.149289, week-long 0.149128, occasional 0.146385

20.7.4 Evaluating Distributional Word Similarity

Distributional similarity can be evaluated in the same ways as thesaurus-based similarity; we can compare intrinsically to human similarity scores, or we can evaluate it extrinsically as part of end-to-end applications. Besides word sense disambiguation and malapropism detection, similarity measures have been used as a part of systems for the grading of exams and essays(Landauer et al., 1997), or taking TOEFL multiple-choice exams (Landauer and Dumais, 1997; Turney et al., 2003).

Distributional algorithms are also often evaluated in a third intrinsic way: by comparison with a gold-standard thesaurus. This comparison can be direct with a single thesaurus (Grefenstette, 1994; Lin, 1998a) or by using precision and recall measure against an ensemble of thesauri (Curran and Moens, 2002). Let S be the set of words that are defined as similar in the thesaurus, by being in the same synset, or perhaps sharing the same hypernym, or being in the hypernym-hyponym relation. Let S' be the set of words that are classified as similar by some algorithm. We can define precision and recall as:

$$\text{precision} = \frac{|S \cap S'|}{|S'|} \text{recall} = \frac{|S \cap S'|}{|S|} \tag{20.53}$$

Curran (2003) evaluated a number of distributional algorithms using comparison with thesauri and found that the Dice and Jaccard methods performed best as measures of vector similarity, while t-test performed best as a measure of association. Thus the best metric weighted the associations with t-test, and then used either Dice or Jaccard to measure vector similarity.

20.8 Hyponymy and other word relations

Similarity is only one kind of semantic relation between words. As we discussed in Ch. 19, WordNet and MeSH both include **hyponymy/hypernymy**, as do many thesauruses for other languages, such as CiLin for Chinese (?). WordNet also includes **antonymy**, **meronymy**, and other relations. Thus if we want to know if two senses are related by one of these relations, and the senses occur in WordNet or MeSH, we can just look them up. But since many words are not in these resources, it is important to be able to learn new hypernym and meronym relations automatically.

Much work on automatic learning of word relations is based on a key insight first articulated by Hearst (1992), that the presence of certain lexico-syntactic patterns can indicate a particular semantic relationship between two nouns. Consider the following sentence extracted by Hearst from the Groliers encyclopedia:

(20.54) Agar is a substance prepared from a mixture of red algae, such as Gelidium, for laboratory or industrial use.

Hearst points out that most human readers will not know what *Gelidium* is, but that they can readily infer that it is a kind of (a **hyponym** of) *red algae*, whatever that is. She suggests that the following **lexico-syntactic pattern**

$$NP_0 \textit{ such as } NP_1\{,NP_2\ldots,(\textit{and}|\textit{or})NP_i\}, i \geq 1 \tag{20.55}$$

implies the following semantics

$$\forall NP_i, i \geq 1, \text{hyponym}(NP_i, NP_0) \tag{20.56}$$

allowing us to infer

$$\text{hyponym}(\text{Gelidium}, \text{red algae}) \tag{20.57}$$

$NP\{,NP\}*\{,\}$ (and$\mid$or) other NP_H	. . . temples, treasuries, and other important civic buildings.
NP_H such as $\{NP,\}$* (or$\mid$and) NP	red algae such as Gelidium
such NP_H as $\{NP,\}$* (or$\mid$and) NP	works by such authors as Herrick, Goldsmith, and Shakespeare
NP_H $\{,\}$ including $\{NP,\}$* (or$\mid$and) NP	All common-law countries, including Canada and England
NP_H $\{,\}$ especially $\{NP,\}$* (or$\mid$and) NP	. . . most European countries, especially France, England, and Spain

Figure 20.14 Hand-built lexico-syntactic patterns for finding hypernyms (Hearst, 1992, 1998)

Fig. 20.14 shows five patterns Hearst (1992, 1998) suggested for inferring the hyponym relation; we've shown NP_H as the parent/hyponym. There are a number of other attempts to extract different WordNet relations using such patterns; see the history section for more details.

Of course, the coverage of such pattern-based methods is limited by the number and accuracy of the available patterns. Unfortunately, once the obvious examples have been found, the process of creating patterns by hand becomes a difficult and slow process. Fortunately, we've already seen the solution to this kind of problem. We can find new patterns using **bootstrapping** methods that are common in information extraction (Riloff, 1996; Brin, 1998), and are also key to the Yarowsky method described earlier in Sec. 20.5.

The key insight for the use of bootstrapping in relational pattern discovery is that with a large corpus we can expect that words involved in a relation to show up with many different patterns that express that same relation. Therefore, in theory at least, we need only start with a small number of precise patterns to acquire a set of seed words involved in a given relation. These words can then be used to query a large corpus for sentences containing both terms in some kind of dependency relation; new patterns can then be extracted from these new sentences. The process can be repeated until the pattern set is large enough.

As an example of this process, consider the terms "red algae" and "Gelidium" discovered earlier using Hearst's simple pattern set. Among the results of a simple Google search using these as query terms is the following example:

(20.58) One example of a red algae is Gelidium.

Removing the seed words from such a sentence and replacing them with simple wildcards is the crudest kind of pattern generation. In this case, submitting the pattern "One example of a * is *" to Google currently yields nearly 500,000 hits, including the following example:

(20.59) One example of a boson is a photon.

We can also extract slightly more sophisticated patterns by parsing the extracted sentences and putting wildcards into the parse tree.

The key to the success of bootstrapping approaches is to avoid the *semantic drift* that tends to occur as part of repeated applications of bootstrapping. The further we get from the original set of seed words or patterns the more likely it is that we'll come across patterns with meanings quite different from what we set out to discover. We'll see methods for dealing with this drift when we discuss bootstrapping for information extraction in Ch. 22.

An alternative to bootstrapping is to use large lexical resources like WordNet as a source of training information, in which each WordNet hypernym/hyponym pair tells us something about kinds of words are in this relation, and we train a classifier to help find new words that exhibit this relation.

This hyponym learning algorithm of Snow et al. (2005), for example, relies on WordNet to help learn large numbers of weak hyponym patterns, and then combine them in a supervised classifier in 4 steps:

1. Collect all pairs of WordNet noun concepts c_i, c_j that are in the hypernym/hyponym relation.
2. For each noun pair, collect all sentences (in a 6 million word corpus) in which both nouns occur.

3. Parse the sentences and automatically extract every possible Hearst-style lexico-syntactic pattern from the parse tree
4. Use the large set of patterns as features in an logistic regression classifier
5. Given a pair of nouns in the test set, extract features and use the classifier to determine if the noun pair is related by the hypernym/hyponym relation or not.

Four of the new patterns automatically learned by this algorithm include:

NP_H like NP	NP_H called NP
NP is a NP_H	NP, a NP_H (appositive):

Snow et al. (2005) then showed good hypernym detection performance by using each of these patterns as a weak feature combined by a logistic regression classifier.

Another way to use WordNet to help address the hypernym problem is to model the task as choosing the place to insert unknown words into an otherwise complete hierarchy. It is possible to do this without using lexico-syntactic patterns. For example, we can use a similarity classifier (using distributional information, or morphological information) to find the words in the hierarchy that are most similar to an unknown word, using an approach like K-Nearest-Neighbors, and insert the new word there (Tseng, 2003). Or we can treat the task of hypernym labeling as a labeling task like named-entity tagging. Ciaramita and Johnson (2003) take this approach, using as tags 26 **supersenses**, from the 26 broad-category 'lexicographer class' labels from WordNet (*person*, *location*, *event*, *quantity*, etc). They use features such as surrounding part-of-speech tags, word bigram and trigram features, spelling and morphological features, and apply a multiclass perceptron classifier.

SUPERSENSES

Finding **meronyms** seems to be harder than hyponyms; here are some examples from Girju et al. (2003):

(20.60) The car's mail messenger is busy at work in the <PART>mail car</PART> as the <WHOLE>train</WHOLE> moves along.

(20.61) Through the open <PART>side door</PART> of the <WHOLE>car</WHOLE>, moving scenery can be seen.

Meronyms are hard to find because the lexico-syntactic patterns that characterize them are very ambiguous. For example the two most common patterns indicating meronymy are the English genitive constructions [NP_1 of NP_2] and [NP_1's NP_2], which also express many other meanings such as *possession*; see Girju et al. (2003, 2006) for discussion and possible algorithms.

THESAURUS INDUCTION

Learning individual relations between words is an important component of the general task of **thesaurus induction**. In thesaurus induction, we combine our estimates of word similarity with our hypernym or other relations to build an entire ontology or thesaurus. For example the two-step thesaurus induction algorithm of Caraballo (1999, 2001) first applies a bottom-up **clustering** algorithm to group together semantically similar words into an unlabeled word hierarchy. Recall from Sec. 20.10 that in agglomerative clustering, we start by assigning each word its own cluster. New clusters are then formed in a bottom-up fashion by successively merging the two clusters that are most similar; we can use any metric for semantic similarity, such as one of the distributional metrics described in the previous section. In the second step, given the unlabeled hierarchy, the algorithm uses a pattern-based hyponym classifier to assign a hypernym label to each cluster of words. See the history section for more recent work on thesaurus induction.

20.9 Semantic Role Labeling

SEMANTIC ROLE LABELING

The final task we'll discuss in this chapter links word meanings with sentence meanings. This is the task of **semantic role labeling**, sometimes called **thematic role labeling**, **case role assignment** or even **shallow semantic parsing**. Semantic role labeling is the task of automatically finding the **semantic roles** for each predicate in a sentence. More specifically, that means determining which constituents in a sentence are semantic arguments for a given predicate, and then determining the appropriate role for each of those arguments. Semantic role labeling has the potential to improve performance in any language understanding task, although to date its primary applications have been in question answering and information extraction.

Current approaches to semantic role labeling are based on supervised machine learning and hence require access to adequate amounts of training and testing materials. Over the last few years, both the FrameNet and PropBank resources discussed in Ch. 19 have played this role. That is, they have been used to specify what counts as a predicate, to define the set of roles used in the task and to provide training and test data. The SENSEVAL-3 evaluation used Framenet, while the CONLL evaluations in 2004 and 2005 were based on PropBank.

The following examples show the different representations from the two efforts. Recall that FrameNet (20.62) employs a large number of frame-specific frame elements as roles, while PropBank (20.63) makes use of a smaller number of numbered argument labels which can be interpreted as verb-specific labels.

(20.62)

[You]	can't	[blame]	[the program]	[for being unable to identify a processor]
COGNIZER		TARGET	EVALUEE	REASON

(20.63)

[The San Francisco Examiner]	issued	[a special edition]	[around noon yesterday]
ARG0	TARGET	ARG1	ARGM-TMP

A simplified semantic role labeling algorithm is sketched in Fig. 20.15. Following the very earliest work on semantic role analysis (Simmons, 1973), most work on semantic role labeling begins by parsing the sentence. Publicly available broad-coverage parsers (such as Collins (1996) or Charniak (1997)) are typically used to assign a parse to the input string. Fig. 20.16 shows a parse of (20.63) above. The resulting parse is then traversed to find all predicate-bearing words. For each of these predicates the tree is again traversed to determine which role, if any, each constituent in the parse plays with respect to that predicate. This judgment is made by first characterizing the constituent as a set of features with respect to the predicate. A classifier trained on an appropriate training set is then passed this feature set and makes the appropriate assignment.

Let's look in more detail at the simple set of features suggested by Gildea and Jurafsky (2000, 2002), which have been incorporated into most role-labeling systems. We'll

```
function SemanticRoleLabel(words) returns labeled tree

    parse ← Parse(words)
    for each predicate in parse do
        for each node in parse do
            featurevector ← ExtractFeatures(node, predicate, parse)
            ClassifyNode(node, featurevector, parse)
```

Figure 20.15 A generic semantic role labeling algorithm. The ClassifyNode component can be a simple 1-of-N classifier which assigns a semantic role (or NONE for non-role constituents). ClassifyNode can be trained on labeled data such as FrameNet or PropBank.

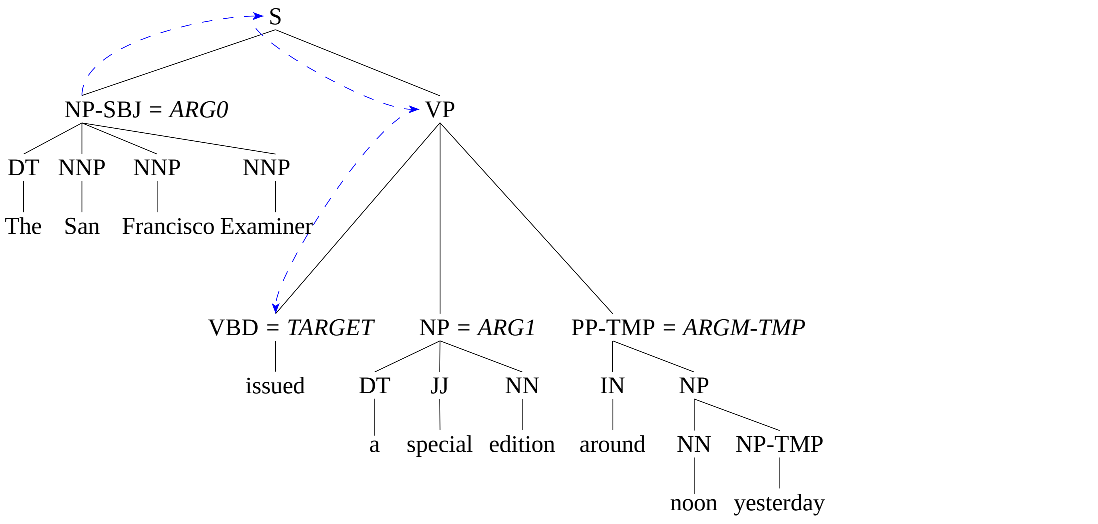


Figure 20.16 Parse tree for a PropBank sentence, showing the PropBank argument labels. The dotted line shows the **path** feature NP↑S↓VP↓VBD for ARG0, the NP-SBJ constituent *the San Francisco Examiner*.

extract them for the first *NP* in Fig. 20.16, the *NP-SBJ* constituent *the San Francisco Examiner*.

- The governing **predicate**, in this case the verb *issued*. For PropBank, the predicates are always verbs; FrameNet also has noun and adjective predicates. The predicate is a crucial feature, since both PropBank and FrameNet labels are defined only with respect to a particular predicate.
- The **phrase type** of the constituent, in this case *NP* (or *NP-SBJ*). This is simply the name of the parse node which dominates this constituent in the parse tree. Some semantic roles tend to appear as *NP*s, others as *S* or *PP*, and so on.
- The **head word** of the constituent, *Examiner*. The head word of a constituent can be computed using standard head rules, such as those given in Ch. 12 in

Fig. **??**. Certain head words (e.g. pronouns) place strong constraints on the possible semantic roles they are likely to fill.

- The **head word part-of-speech** of the constituent, *NNP*.
- The **path** in the parse tree from the constituent to the predicate. This path is marked by the dotted line in Fig. 20.16. Following (Gildea and Jurafsky, 2000), we can use a simple linear representation of the path, NP↑S↓VP↓VBD. ↑ and ↓ represent upward and downward movement in the tree respectively. The path is very useful as a compact representation of many kinds of grammatical function relationships between the constituent and the predicate.
- The **voice** of the clause in which the constituent appears, in this case **active** (as contrasted with **passive**). Passive sentences tend to have strongly different linkings of semantic roles to surface form than active ones.
- The binary **linear position** of the constituent with respect to the predicate, either **before** or **after**.
- The **sub-categorization** of the predicate. Recall from Ch. 12 that the subcategorization of a verb is the set of expected arguments that appear in the verb phrase. We can extract this information by using the phrase structure rule that expands the immediate parent of the predicate; VP → NP PP for the predicate in Fig. 20.16.

Many other features are generally extracted by semantic role labeling systems, such as named entity tags (it is useful to know if a constituent is a LOCATION or PERSON, for example), or more complex versions of the path features (the upward or downward halves, whether particular nodes occur in the path), the rightmost or leftmost words of the constituent, and so on.

We now have a set of observations like the following example, each with a vector of features; we have shown the features in the order described above (recall that most observations will have the value NONE rather than e.g., ARG0, since most constituents in the parse tree will not bear a semantic role):

ARG0: [issued, NP, Examiner, NNP, NP↑S↓VP↓VBD, active, before, VP → NP PP]

Just as we saw for word sense disambiguation, we can divide these observations into a training and a test set, use the training examples in any supervised machine learning algorithm, and build a classifier. SVM and Maximum Entropy classifiers have yielded good results on this task on standard evaluations. Once trained, the classifier can be used on unlabeled sentences to propose a role for each constituent in the sentence. More precisely, an input sentence is parsed and a procedure similar to that described earlier for training is employed.

Instead of training a single stage classifier, some role labeling algorithms do classification in multiple stages for efficiency:

- **Pruning:** to speed up execution, some constituents are eliminated from consideration as possible roles, based on simple rules
- **Identification:** a binary classification of each node as an ARG to be labeled or a NONE.

- **Classification:** a one-of-N classification of all the constituents that were labeled as ARG by the previous stage.

There are a number of complications that all semantic role labeling systems need to deal with. Constituents in FrameNet and PropBank are required to be non-overlapping. Thus if a system incorrectly labels two overlapping constituents as arguments, it needs to decide which of the two is correct. Additionally, the semantic roles of constituents are not independent; since PropBank does not allow multiple identical arguments, labeling one constituent as an ARG0 would greatly increase the probability of another constituent being labeled ARG1. Both these problems can be addressed by the two-stage approaches based on lattice or *N*-best rescoring discussed in Ch. 9: having the classifier assign multiple labels to each constituent, each with a probability, and using a second global optimization pass to pick the best label sequence.

Instead of using parses as input, it is also possible to do semantic role labeling directly from raw (or part-of-speech tagged) text by applying the chunking techniques used for named entity extraction or partial parsing. Such techniques are particularly useful in domains such as bioinformatics where it is unlikely that syntactic parsers trained on typical newswire text will perform well.

Finally, semantic role labeling systems have been generally evaluated by requiring that each argument label must be assigned to the exactly correct word sequence or parse constituent. Precision, recall, and F-measure can then be computed. A simple rule-based system can be used as a baseline, for example tagging the first NP before the predicate as ARG0 and the first NP after the predicate as ARG1, and switching these if the verb phrase is passive.

20.10 ADVANCED: UNSUPERVISED SENSE DISAMBIGUATION

Let's briefly return to the WSD task. It is expensive and difficult to build large corpora in which each word is labeled for its word sense. For this reason, unsupervised approaches to sense disambiguation are an exciting and important research area.

In unsupervised approaches, we don't use human-defined word senses. Instead, the set of 'senses' of each word are created automatically from the instances of each word in the training set. Let's introduce a simplified version of the methods of Schütze's (Schütze, 1992b, 1998) on unsupervised sense disambiguation. In Schütze's method, we first represent each instance of a word in the training set by distributional context feature-vectors that are a slight generalization of the feature vectors we defined in Sec. 20.7. (It is for this reason that we turned to unsupervised sense disambiguation only after introducing word similarity.)

As in Sec. 20.7 we will represent a word w as a vector based on frequencies of its neighboring words. For example for a given target word (type) w, we might select 1000 words that occur most frequently within 25 words of any instance of w. These 1000 words become the dimension of the vector. Let's define f_i to mean the frequency with which word i occurs in the context of word w. We define the word vector $\vec{w}$ (for a given token (observation) of w) as:

$$\vec{w} = (f_1, f_2, f_3, \cdots, f_{1000})$$

So far this is just a version of the distributional context we saw in Sec. 20.7. We can also use a slightly more complex version of the distributional context. For example Schuetze defines the **context vector** of a word w not as this first-order vector, but instead by its **second order co-occurrence**. That is, the context vector for a word w is built by taking each word x in the context of w, for each x computing its word vector $\vec{x}$, and then taking the centroid (average) of the vectors $\vec{x}$.

Let's see how we use these context vectors (whether first-order or second-order) in unsupervised sense disambiguation of a word w. In training, we'll need only 3 steps:

1. For each token w_i of word w in a corpus, compute a context vector $\vec{c}$.
2. Use a **clustering algorithm** to **cluster** these word token context vectors $\vec{c}$ into a predefined number of groups or clusters. Each cluster defines a sense of w.
3. Compute the **vector centroid** of each cluster. Each vector centroid $\vec{s_j}$ is a **sense vector** representing that sense of w.

Since this is an unsupervised algorithm we won't have names for each of these 'senses' of w; we just refer to the jth sense of w.

Now how do we disambiguate a particular token t of w? Again we have three steps:

1. Compute a context vector $\vec{c}$ for t as discussed above.
2. Retrieve all sense vectors s_j for w.
3. Assign t to the sense represented by the sense vector s_j that is closest to t.

All we need is a clustering algorithm, and a distance metrics between vectors. Fortunately, clustering is a well-studied problem with a wide number of standard algorithms that can be applied to inputs structured as vectors of numerical values (Duda and Hart, 1973). A frequently used technique in language applications is known as **agglomerative clustering**. In this technique, each of the N training instances is initially assigned to its own cluster. New clusters are then formed in a bottom-up fashion by successively merging the two clusters that are most similar. This process continues until either a specified number of clusters is reached, or some global goodness measure among the clusters is achieved. In cases where the number of training instances makes this method too expensive, random sampling can be used on the original training set (Cutting et al., 1992) to achieve similar results.

AGGLOMERATIVE CLUSTERING

How can we evaluate unsupervised sense disambiguation approaches? As usual, the best way is to do extrinsic or in vivo evaluation, in which the WSD algorithm is embedded in some end-to-end system. Intrinsic evaluation can also be useful, though, if we have some way to map the automatically derived sense classes into some hand-labeled gold standard set, so that we can compare a hand-labeled test set with a set labeled by our unsupervised classifier. One way of doing this mapping is to map each sense cluster to a pre-defined sense by choosing the sense that (in some training set) has the most word tokens overlapping with the cluster. Another is to consider all pairs of words in the test set, testing for each whether both the system and the hand-labeling put both members of the pair in the same cluster or not.

Bibliographical and Historical Notes

Word sense disambiguation traces its roots to some of the earliest applications of digital computers. We saw above Warren Weaver's (1955) suggestion to disambiguate a word by looking at a small window around it, in the context of machine translation. Other notions first proposed in this early period include the use of a thesaurus for disambiguation (Masterman, 1957), supervised training of Bayesian models for disambiguation (Madhu and Lytel, 1965), and the use of clustering in word sense analysis (Sparck Jones, 1986).

An enormous amount of work on disambiguation has been conducted within the context of early AI-oriented natural language processing systems. While most natural language analysis systems of this type exhibited some form of lexical disambiguation capability, a number of these efforts made word sense disambiguation a larger focus of their work. Among the most influential efforts were the efforts of Quillian (1968) and Simmons (1973) with semantic networks, the work of Wilks with *Preference Semantics* Wilks (1975c, 1975b, 1975a), and the work of Small and Rieger (1982) and Riesbeck (1975) on word-based understanding systems. Hirst's ABSITY system (Hirst and Charniak, 1982; Hirst, 1987, 1988), which used a technique based on semantic networks called marker passing, represents the most advanced system of this type. As with these largely symbolic approaches, most connectionist approaches to word sense disambiguation have relied on small lexicons with hand-coded representations (Cottrell, 1985; Kawamoto, 1988).

Considerable work on sense disambiguation has been conducted in the areas of Cognitive Science and psycholinguistics. Appropriately enough, it is generally described using a different name: lexical ambiguity resolution. Small et al. (1988) present a variety of papers from this perspective.

The earliest implementation of a robust empirical approach to sense disambiguation is due to Kelly and Stone (1975) who directed a team that hand-crafted a set of disambiguation rules for 1790 ambiguous English words. Lesk (1986) was the first to use a machine readable dictionary for word sense disambiguation. Wilks et al. (1996) describe extensive explorations of the use of machine readable dictionaries. The problem of dictionary senses being too fine-grained or lacking an appropriate organization has been addressed with models of clustering word senses Dolan (1994), Peters et al. (1998), Chen and Chang (1998), Mihalcea and Moldovan (2001), Agirre and de Lacalle (2003), Chklovski and Mihalcea (2003), Palmer et al. (2004), McCarthy (2006), Navigli (2006), Snow et al. (2007); corpora with clustered word senses for training clustering algorithms include Palmer et al. (2006) and **OntoNotes** (Hovy et al., 2006).

ONTONOTES

Modern interest in supervised machine learning approaches to disambiguation began with Black (1988), who applied decision tree learning to the task. The need for large amounts of annotated text in these methods led to investigations into the use of bootstrapping methods (Hearst, 1991; Yarowsky, 1995). The problem of how to weigh and combine disparate sources of evidence is explored in Ng and Lee (1996), McRoy (1992), and Stevenson and Wilks (2001).

Among the semi-supervised methods, more recent models of selectional preference include Li and Abe (1998), Ciaramita and Johnson (2000), McCarthy and Carroll (2003), Light and Greiff (2002). Diab and Resnik (2002) give a semi-supervised algorithm for sense disambiguation based on aligned parallel corpora in two languages. For example, the fact that the French word *catastrophe* might be translated as English *disaster* in one instance and *tragedy* in another instance can be used to disambiguate the senses of the two English words (i.e. to choose senses of *disaster* and *tragedy* that are similar). Abney (2002, 2004) explores the mathematical foundations of the Yarowsky algorithm and its relation to co-training. The most-frequent-sense heuristic is an extremely powerful one, but requires large amounts of supervised training data. McCarthy et al. (2004) propose an unsupervised way to automatically estimate the most frequent sense, based on the thesaurus similarity metrics defined in Sec. 20.6.

The earliest attempt to use clustering in the study of word senses is due to Sparck Jones (1986). Zernik (1991) successfully applied a standard information retrieval clustering algorithm to the problem, and provided an evaluation based on improvements in retrieval performance. More extensive recent work on clustering can be found in Pedersen and Bruce (1997) and Schütze (1997, 1998).

A few algorithms have attempted to exploit the power of mutually disambiguating all the words in a sentence, either by multiple passes (Kelly and Stone, 1975) to take advantage of easily disambiguated words, or by parallel search (Cowie et al., 1992; Veronis and Ide, 1990).

Recent work has focused on ways to use the web for training data for word sense disambiguation, either unsupervised (Mihalcea and Moldovan, 1999) or by using volunteers to label data (Chklovski and Mihalcea, 2002).

Resnik (2006) describes potential applications of WSD. One recent application has been to improve machine translation Chan et al. (2007), Carpuat and Wu (2007).

Agirre and Edmonds (2006) is a comprehensive edited volume that summarizes the state of the art in WSD. Ide and Veronis (1998a) provide a comprehensive review of the history of word sense disambiguation up to 1998. Ng and Zelle (1997) provide a more focused review from a machine learning perspective. Wilks et al. (1996) describe dictionary and corpus experiments, along with detailed descriptions of very early work.

The models of distributional word similarity we discussed arose out of research in linguistics and psychology of the 1950's. The idea that meaning was related to distribution of words in context was widespread in linguistic theory of the 1950's; even before the well-known Firth (1957) and Harris (1968) dictums discussed earlier, Joos (1950) stated that

> the linguist's 'meaning' of a morpheme…is by definition the set of conditional probabilities of its occurrence in context with all other morphemes'

The related idea that the meaning of a word could be modeled as a point in a Euclidean space, and that the similarity of meaning between two words could be modeled as the distance between these points, was proposed in psychology by Osgood et al. (1957). The application of these ideas in a computational framework was first made by Sparck Jones (1986), and became a core principle of information retrieval, from whence it came into broader use in speech and language processing.

There are a wide variety of other weightings and methods for word similarity. The largest class of methods not discussed in this chapter are the variants to and details of the **information-theoretic** methods like Jensen-Shannon divergence, KL-divergence and α-skew divergence that we briefly introduced (Pereira et al., 1993; Dagan et al., 1994, 1999; Lee, 1999, 2001); there are also other metrics from Hindle (1990) and Lin (1998a). Alternative paradigms include the **co-occurrence retrieval** model (Weeds, 2003; Weeds and Weir, 2005). Manning and Schütze (1999, Chapter 5 and 8) give collocation measures and other related similarity measures. A commonly used weighting is **weighted mutual information** (Fung and McKeown, 1997) in which the pointwise mutual information is weighted by the joint probability. In information retrieval the **TF/IDF** weight is widely used, as we will see in Ch. 23. See Dagan (2000), Mohammad and Hirst (2005), Curran (2003) and Weeds (2003) for good summaries of distributional similarity.

WEIGHTED MUTUAL INFORMATION

An alternative vector space model of semantic similarity, **Latent Semantic Indexing** (LSI) or **Latent Semantic Analysis** (**LSA**), uses **singular value decomposition** to reduce the dimensionality of the vector space with the intent of discovering higher-order regularities (Deerwester et al., 1990). We have already discussed Schütze (1992b), another semantic similarity model based on singular value decomposition.

LATENT SEMANTIC INDEXING

LSA

There is a wide variety of recent literature on other lexical relations and thesaurus induction. The use of distributional word similarity for thesaurus induction was explored systematically by Grefenstette (1994). A wide variety of distributional clustering algorithms have been applied to the task of discovering groupings of semantically similar words, including hard clustering (Brown et al., 1992), soft clustering (Pereira et al., 1993), as well as new algorithms like **Clustering By Committee** (CBC) (Lin and Pantel, 2002). For particular relations, Lin et al. (2003) applied hand-crafted patterns to find **antonyms**, with the goal of improving synonym-detection. The distributional word similarity algorithms from Sec. 20.7 often incorrectly assign high similarity to antonyms. Lin et al. (2003) showed that words appearing in the patterns *from X to Y* or *either X or Y* tended to be antonyms. Girju et al. (2003, 2006) show improvements in **meronym** extraction by learning generalizations about the semantic superclasses of the two nouns. Chklovski and Pantel (2004) used hand-built patterns to extract fine-grained relations between verbs such as **strength**. Much recent work has focused on thesaurus induction by combining different relation extractors. Pantel and Ravichandran (2004), for example, extend Caraballo's algorithm for combining similarity and hyponymy information, while Snow et al. (2006) integrate multiple relation extractors to compute the most probable thesaurus structure. Recent work on similarity focuses on the use of the Web, for example relying on Wikipedia Strube and Ponzetto (2006), Gabrilovich and Markovitch (2007); this Web-based work is also closely related to unsupervised information extraction; see Ch. 22 and references like Etzioni et al. (2005).

While not as old a field as word similarity or sense disambiguation, semantic role labeling has a long history in computational linguistics. The earliest work on semantic role labeling (Simmons, 1973) first parsed a sentence using an ATN parser. Each verb then had a set of rules specifying how the parse should be mapped to semantic roles. These rules mainly made reference to grammatical functions (subject, object, complement of specific prepositions), but also checked constituent-internal features such as the animacy of head nouns.

Statistical work in the area revived in 2000 after the FrameNet and PropBank project had created databases large enough and consistent enough to make training and testing possible. Many popular features used for role labeling are defined in Gildea and

Jurafsky (2002), Chen and Rambow (2003), Surdeanu et al. (2003), Xue and Palmer (2004), Pradhan et al. (2003, 2005).

To avoid the need for huge labeled training sets, recent work has focused on unsupervised approaches for semantic role labeling (Swier and Stevenson, 2004).

The semantic labeling work described above focuses on labeling each sentence token in a corpus with semantic roles. An alternative approach to semantic role labeling focuses on lexicon learning, using unsupervised learning on a corpus to learn the kinds of semantic classes a verb can belong to in terms of its possible semantic roles or argument alternation patterns (Stevenson and Merlo, 1999; Schulte im Walde, 2000; Merlo and Stevenson, 2001; Merlo et al., 2001; Grenager and Manning, 2006).

EXERCISES

20.1 Collect a small corpus of example sentences of varying lengths from any newspaper or magazine. Using WordNet, or any standard dictionary, determine how many senses there are for each of the open-class words in each sentence. How many distinct combinations of senses are there for each sentence? How does this number seem to vary with sentence length?

20.2 Using WordNet, or a standard reference dictionary, tag each open-class word in your corpus with its correct tag. Was choosing the correct sense always a straightforward task. Report on any difficulties you encountered.

20.3 Using the same corpus, isolate the words taking part in all the verb-subject and verb-object relations. How often does it appear to be the case that the words taking part in these relations could be disambiguated using only information about the words in the relation?

20.4 Between the words *eat* and *find* which would you expect to be more effective in selectional restriction-based sense disambiguation? Why?

20.5 Using your favorite dictionary, simulate the Original Lesk word overlap disambiguation algorithm described on page 11 on the phrase *Time flies like an arrow*. Assume that the words are to be disambiguated one at a time, from left to right, and that the results from earlier decisions are used later in the process.

20.6 Build an implementation of your solution to the previous exercise. Using WordNet, implement the Original Lesk word overlap disambiguation algorithm described on page 11 on the phrase *Time flies like an arrow*.

20.7 Implement and experiment with a decision-list sense disambiguation system. As a model, use the kinds of features shown in Figure 20.2. Use one of the publicly available decision-list packages like WEKA (or see Russell and Norvig (1995) for more details on implementing decision-list learning yourself). To facilitate evaluation of your system, you should obtain one of the freely available sense-tagged corpora.

20.8 Evaluate two or three of the similarity methods from the publicly available `Wordnet::Similarity` package (Pedersen et al., 2004). You might do this by hand-labeling some word pairs with similarity scores and seeing how well the algorithms approximate your hand labels.

20.9 Implement a distributional word similarity algorithm that can take different measures of association and different measures of vector similarity. Now evaluate two measures of association and two measures of vector similarity from Fig. 20.13. Again, you might do this by hand-labeling some word pairs with similarity scores and seeing how well the algorithms approximate your hand labels.

Abney, S. P. (2002). Bootstrapping. In *ACL-02*.

Abney, S. P. (2004). Understanding the Yarowsky algorithm. *Computational Linguistics*, *30*(3), 365–395.

Agirre, E. and de Lacalle, O. L. (2003). Clustering wordnet word senses. In *RANLP 2003*.

Agirre, E. and Edmonds, P. (Eds.). (2006). *Word Sense Disambiguation: Algorithms and Applications*. Kluwer.

Atkins, S. (1993). Tools for computer-aided corpus lexicography: The Hector project. *Acta Linguistica Hungarica*, *41*, 5–72.

Banerjee, S. and Pedersen, T. (2003). Extended gloss overlaps as a measure of semantic relatedness. In *IJCAI 2003*, pp. 805–810.

Black, E. (1988). An experiment in computational discrimination of English word senses. *IBM Journal of Research and Development*, *32*(2), 185–194.

Brin, S. (1998). Extracting patterns and relations from the World Wide Web. In *Proceedings World Wide Web and Databases International Workshop, Number 1590 in LNCS*, pp. 172–183. Springer.

Brown, P. F., Della Pietra, V. J., de Souza, P. V., Lai, J. C., and Mercer, R. L. (1992). Class-based *n*-gram models of natural language. *Computational Linguistics*, *18*(4), 467–479.

Bruce, R. and Wiebe, J. (1994). Word-sense disambiguation using decomposable models. In *Proceedings of the 32nd ACL*, Las Cruces, NM, pp. 139–145.

Budanitsky, A. and Hirst, G. (2001). Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. In *Proceedings of the NAACL 2001 Workshop on WordNet and Other Lexical Resources*, Pittsburgh, PA.

Budanitsky, A. and Hirst, G. (2006). Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, *32*(1), 13–47.

Caraballo, S. A. (1999). Automatic construction of a hypernym-labeled noun hierarchy from text. In *ACL-99*, College Park, MD. ACL.

Caraballo, S. A. (2001). *Automatic Acquisition of a hypernym-labeled noun hierarchy from text*. Ph.D. thesis, Brown University.

Carpuat, M. and Wu, D. (2007). Improving statistical machine translation using word sense disambiguation. In *EMNLP/CoNLL 2007*, Prague, Czech Republic, pp. 61–72.

Chan, Y. S., Ng, H. T., and Chiang, D. (2007). Word sense disambiguation improves statistical machine translation. In *ACL-07*, Prague, Czech Republic, pp. 33–40.

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *AAAI-97*, Menlo Park, pp. 598–603. AAAI Press.

Chen, J. N. and Chang, J. S. (1998). Topical clustering of MRD senses based on information retrieval techniques. *Computational Linguistics*, *24*(1), 61–96.

Chen, J. and Rambow, O. (2003). Use of deep linguistic features for the recognition and labeling of semantic arguments. In *EMNLP 2003*, pp. 41–48.

Chklovski, T. and Mihalcea, R. (2003). Exploiting Agreement and Disagreement of Human Annotators for Word Sense Disambiguation. In *RANLP 2003*.

Chklovski, T. and Mihalcea, R. (2002). Building a sense tagged corpus with open mind word expert. In *ACL-02 Workshop on Word Sense Disambiguation: Recent Successes and Future Directions*, pp. 116–122.

Chklovski, T. and Pantel, P. (2004). Verb ocean: Mining the Web for fine-grained semantic verb relations. In *EMNLP 2004*, pp. 25–26.

Church, K. W. and Hanks, P. (1989). Word association norms, mutual information, and lexicography. In *Proceedings of the 27th ACL*, Vancouver, B.C., pp. 76–83. ACL.

Church, K. W. and Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational Linguistics*, *16*(1), 22–29.

Ciaramita, M. and Johnson, M. (2000). Explaining away ambiguity: learning verb selectional preference with Bayesian networks. In *COLING-00*, pp. 187–193. ACL.

Ciaramita, M. and Johnson, M. (2003). Supersense tagging of unknown nouns in WordNet. In *EMNLP-2003*, pp. 168–175. ACL.

Collins, M. (1996). A new statistical parser based on bigram lexical dependencies. In *ACL-96*, Santa Cruz, California, pp. 184–191.

Cottrell, G. W. (1985). *A Connectionist Approach to Word Sense Disambiguation*. Ph.D. thesis, University of Rochester, Rochester, NY. Revised version published in the same title by Pitman in 1989.

Cowie, J., Guthrie, J. A., and Guthrie, L. M. (1992). Lexical disambiguation using simulated annealing. In *COLING-92*, Nantes, France, pp. 359–365.

Curran, J. R. (2003). *From Distributional to Semantic Similarity*. Ph.D. thesis, University of Edinburgh.

Curran, J. R. and Moens, M. (2002). Improvements in automatic thesaurus extraction. In *Proceedings of the ACL-02 workshop on Unsupervised Lexical Acquisition*, Philadelphia, PA, pp. 59–66. ACL.

Cutting, D., Karger, D. R., Pedersen, J., and Tukey, J. W. (1992). Scatter/gather: A cluster-based approach to browsing large document collections. In *SIGIR-92*, Copenhagen, Denmark, pp. 318–329. ACM.

Dagan, I. (2000). Contextual word similarity. In Dale, R., Moisl, H., and Somers, H. (Eds.), *A Handbook of Natural Language Processing: Techniques and applications for the processing of language as text*. Marcel Dekker.

Dagan, I., Lee, L., and Pereira, F. C. N. (1999). Similarity-based models of cooccurrence probabilities. *Machine Learning*, *34*(1–3), 43–69.

Dagan, I., Marcus, S., and Markovitch, S. (1993). Contextual word similarity and estimation from sparse data. In *Proceedings of the 31st ACL*, Columbus, Ohio, pp. 164–171.

Dagan, I., Pereira, F. C. N., and Lee, L. (1994). Similarity-base estimation of word cooccurrence probabilities. In *Proceedings of the 32nd ACL*, Las Cruces, NM, pp. 272–278.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, *41*, 391–407.

Diab, M. and Resnik, P. (2002). An unsupervised method for word sense tagging using parallel corpora. In *ACL-02*, pp. 255–262. ACL.

Dolan, W. B. (1994). Word sense ambiguation: Clustering related senses. In *COLING-94*, Kyoto, Japan, pp. 712–716. ACL.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York.

Etzioni, O., Cafarella, M., Downey, D., Popescu, A., Shaked, T., Soderland, S., Weld, D., and Yates, A. (2005). Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, *165*(1), 91–134.

Fano, R. M. (1961). *Transmission of information; A statistical theory of communications*. MIT Press.

Firth, J. R. (1957). A synopsis of linguistic theory 1930–1955. In *Studies in Linguistic Analysis*. Philological Society, Oxford. Reprinted in Palmer, F. (ed.) 1968. Selected Papers of J. R. Firth. Longman, Harlow.

Fung, P. and McKeown, K. R. (1997). A technical word and term translation aid using noisy parallel corpora across language groups. *Machine Translation*, *12*(1-2), 53–87.

Gabrilovich, E. and Markovitch, S. (2007). Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis. In *IJCAI-07*.

Gale, W. A., Church, K. W., and Yarowsky, D. (1992a). Work on statistical methods for word sense disambiguation. In Goldman, R. (Ed.), *Proceedings of the 1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language*.

Gale, W. A., Church, K. W., and Yarowsky, D. (1992b). Estimating upper and lower bounds on the performance of word-sense disambiguation programs. In *Proceedings of the 30th ACL*, Newark, DE, pp. 249–256.

Gale, W. A., Church, K. W., and Yarowsky, D. (1992c). One sense per discourse. In *Proceedings DARPA Speech and Natural Language Workshop*, pp. 233–237. Morgan Kaufmann.

Gaustad, T. (2001). Statistical corpus-based word sense disambiguation: Pseudowords vs. real ambiguous words. In *ACL/EACL 2001 – Student Research Workshop*, pp. 255–262. ACL.

Gildea, D. and Jurafsky, D. (2000). Automatic labeling of semantic roles. In *ACL-00*, Hong Kong, pp. 512–520.

Gildea, D. and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational Linguistics*, *28*(3), 245–288.

Girju, R., Badulescu, A., and Moldovan, D. (2006). Automatic discovery of part-whole relations. *Computational Linguistics*, *31*(1).

Girju, R., Badulescu, A., and Moldovan, D. (2003). Learning semantic constraints for the automatic discovery of part-whole relations. In *HLT-NAACL-03*, Edmonton, Canada, pp. 1–8. ACL.

Gould, S. J. (1980). *The Panda's Thumb*. Penguin Group, London.

Grefenstette, G. (1994). *Explorations in Automatic Thesaurus Discovery*. Kluwer, Norwell, MA.

Grenager, T. and Manning, C. D. (2006). Unsupervised Discovery of a Statistical Verb Lexicon. In *EMNLP 2006*.

Harris, Z. S. (1968). *Mathematical Structures of Language*. John Wiley.

Hearst, M. A. (1991). Noun homograph disambiguation. In *Proceedings of the 7th Annual Conference of the University of Waterloo Centre for the New OED and Text Research*, Oxford, pp. 1–19.

Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In *COLING-92*, Nantes, France.

Hearst, M. A. (1998). Automatic discovery of wordnet relations. In Fellbaum, C. (Ed.), *Wordnet: An Electronic Lexical Database*. MIT Press.

Hindle, D. (1990). Noun classification from predicate-argument structures. In *Proceedings of the 28th ACL*, Pittsburgh, PA, pp. 268–275. ACL.

Hirst, G. (1987). *Semantic Interpretation and the Resolution of Ambiguity*. Cambridge University Press.

Hirst, G. (1988). Resolving lexical ambiguity computationally with spreading activation and polaroid words. In Small, S. L., Cottrell, G. W., and Tanenhaus, M. K. (Eds.), *Lexical ambiguity resolution: Perspectives from psycholinguistics, neuropsychology, and artificial intelligence*, pp. 73–108. Morgan Kaufmann.

Hirst, G. and Budanitsky, A. (2005). Correcting real-word spelling errors by restoring lexical cohesion. *Natural Language Engineering*, *11*, 87–111.

Hirst, G. and Charniak, E. (1982). Word sense and case slot disambiguation. In *AAAI-82*, pp. 95–98.

Hovy, E. H., Marcus, M. P., Palmer, M., Ramshaw, L. A., and Weischedel, R. (2006). Ontonotes: The 90% solution. In *HLT-NAACL-06*.

Ide, N. M. and Veronis, J. (Eds.). (1998a). *Computational Linguistics: Special Issue on Word Sense Disambiguation*, Vol. 24. MIT Press.

Ide, N. M. and Véronis, J. (1998b). Introduction to the special issue on word sense disambiguation. *Computational Linguistics*, *24*(1), 1–40.

Jaccard, P. (1908). Nouvelles recherches sur la distribution florale. *Bulletin de la Société Vaudoise des Sciences Naturelles*, *44*, 223–227.

Jaccard, P. (1912). The distribution of the flora of the alpine zone. *New Phytologist*, *11*, 37–50.

Jiang, J. J. and Conrath, D. W. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. In *ROCLING X*, Taiwan.

Joos, M. (1950). Description of language design. *Journal of the Acoustical Society of America*, *22*, 701–708.

Katz, J. J. and Fodor, J. A. (1963). The structure of a semantic theory. *Language*, *39*, 170–210.

Kawamoto, A. H. (1988). Distributed representations of ambiguous words and their resolution in connectionist networks. In Small, S. L., Cottrell, G. W., and Tanenhaus, M. (Eds.), *Lexical Ambiguity Resolution*, pp. 195–228. Morgan Kaufman.

Kelly, E. F. and Stone, P. J. (1975). *Computer Recognition of English Word Senses*. North-Holland, Amsterdam.

Kilgarriff, A. (2001). English lexical sample task description. In *Proceedings of Senseval-2: Second International Workshop on Evaluating Word Sense Disambiguation Systems*, Toulouse, France, pp. 17–20.

Kilgarriff, A. and Palmer, M. (Eds.). (2000). *Computing and the Humanities: Special Issue on SENSEVAL*, Vol. 34. Kluwer.

Kilgarriff, A. and Rosenzweig, J. (2000). Framework and results for English SENSEVAL. *Computers and the Humanities*, *34*(1-2).

Krovetz, R. (1998). More than one sense per discourse. In *Proceedings of the ACL-SIGLEX SENSEVAL Workshop*.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, *22*, 79–86.

Landauer, T. K. and Dumais, S. T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, *104*, 211–240.

Landauer, T. K., Laham, D., Rehder, B., and Schreiner, M. E. (1997). How well can passage meaning be derived without using word order: A comparison of latent semantic analysis and humans. In *COGSCI-97*, Stanford, CA, pp. 412–417. Lawrence Erlbaum.

Landes, S., Leacock, C., and Tengi, R. I. (1998). Building semantic concordances. In Fellbaum, C. (Ed.), *WordNet: An Electronic Lexical Database*, pp. 199–216. MIT Press.

Leacock, C. and Chodorow, M. S. (1998). Combining local context and WordNet similarity for word sense identification. In Fellbaum, C. (Ed.), *Wordnet: An Electronic Lexical Database*, pp. 265–283. MIT Press.

Leacock, C., Towell, G., and Voorhees, E. (1993). Corpus-based statistical sense resolution. In *Proceedings of the ARPA Human Language Technology Workshop*, pp. 260–265.

Lee, L. (1999). Measures of distributional similarity. In *ACL-99*, pp. 25–32.

Lee, L. (2001). On the effectiveness of the skew divergence for statistical language analysis. In *Artificial Intelligence and Statistics*, pp. 65–72.

Lesk, M. E. (1986). Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the Fifth International Conference on Systems Documentation*, Toronto, CA, pp. 24–26. ACM.

Li, H. and Abe, N. (1998). Generalizing case frames using a thesaurus and the MDL principle. *Computational Linguistics*, *24*(2), 217–244.

Light, M. and Greiff, W. (2002). Statistical models for the induction and use of selectional preferences. *Cognitive Science*, *87*, 1–13.

Lin, D. (1998a). Automatic retrieval and clustering of similar words. In *COLING/ACL-98*, Montreal, pp. 768–774.

Lin, D. (1998b). An information-theoretic definition of similarity. In *ICML 1998*, San Francisco, pp. 296–304.

Lin, D. (2007). Dependency-based word similarity demo. `http://www.cs.ualberta.ca/~lindek/demos.htm`.

Lin, D. and Pantel, P. (2002). Concept discovery from text. In *COLING-02*, pp. 1–7.

Lin, D., Zhao, S., Qin, L., and Zhou, M. (2003). Identifying synonyms among distributionally similar words. In *IJCAI-03*, pp. 1492–1493.

Madhu, S. and Lytel, D. (1965). A figure of merit technique for the resolution of non-grammatical ambiguity. *Mechanical Translation*, *8*(2), 9–13.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Masterman, M. (1957). The thesaurus in syntax and semantics. *Mechanical Translation*, *4*(1), 1–2.

McCarthy, D. (2006). Relating wordnet senses for word sense disambiguation. In *Proceedings of ACL Workshop on Making Sense of Sense*.

McCarthy, D. and Carroll, J. (2003). Disambiguating nouns, verbs, and adjectives using automatically acquired selectional preferences. *Computational Linguistics*, *29*(4), 639–654.

McCarthy, D., Koeling, R., Weeds, J., and Carroll, J. (2004). Finding predominant word senses in untagged text. In *ACL-04*, pp. 279–286.

McRoy, S. (1992). Using multiple knowledge sources for word sense discrimination. *Computational Linguistics*, *18*(1), 1–30.

Merlo, P. and Stevenson, S. (2001). Automatic verb classification based on statistical distribution of argument structure. *Computational Linguistics*, *27*(3), 373–408.

Merlo, P., Stevenson, S., Tsang, V., and Allaria, G. (2001). A multilingual paradigm for automatic verb classification. In *ACL-02*, pp. 207–214.

Mihalcea, R. and Moldovan, D. (2001). Automatic generation of a coarse grained WordNet. In *NAACL Workshop on WordNet and Other Lexical Resources*.

Mihalcea, R. and Moldovan, D. (1999). An automatic method for generating sense tagged corpora. In *Proceedings of AAAI*, pp. 461–466.

Miller, G. A. and Charles, W. G. (1991). Contextual correlates of semantics similarity. *Language and Cognitive Processes*, *6*(1), 1–28.

Miller, G. A., Leacock, C., Tengi, R., and Bunker, R. T. (1993). A semantic concordance. In *Proceedings ARPA Workshop on Human Language Technology*, pp. 303–308. ACL.

Mohammad, S. and Hirst, G. (2005). Distributional measures as proxies for semantic relatedness. Submitted.

Nakov, P. I. and Hearst, M. A. (2003). Category-based pseudowords. In *HLT-NAACL-03*, Edmonton, Canada. ACL.

Navigli, R. (2006). Meaningful clustering of senses helps boost word sense disambiguation performance. In *COLING/ACL 2006*, pp. 105–112.

Ng, H. T. and Lee, H. B. (1996). Integrating multiple knowledge sources to disambiguate word senses: An exemplar-based approach. In *ACL-96*, Santa Cruz, CA, pp. 40–47. ACL.

Ng, H. T. and Zelle, J. (1997). Corpus-based approaches to semantic interpretation in NLP. *AI Magazine*, *18*(4), 45–64.

Osgood, C. E., Suci, G. J., and Tannenbaum, P. H. (1957). *The Measurement of Meaning*. University of Illinois Press, Urbana, IL.

Palmer, M., Dang, H. T., and Fellbaum, C. (2006). Making fine-grained and coarse-grained sense distinctions, both manually and automatically. *Natural Language Engineering*, *13*(2), 137–163.

Palmer, M., Babko-Malaya, O., and Dang, H. T. (2004). Different sense granularities for different applications. In *HLT-NAACL Workshop on Scalable Natural Language Understanding*, Boston, MA, pp. 49–56.

Palmer, M., Fellbaum, C., Cotton, S., Delfs, L., and Dang, H. T. (2001). English tasks: All-words and verb lexical sample. In *Proceedings of Senseval-2: Second International Workshop on Evaluating Word Sense Disambiguation Systems*, Toulouse, France, pp. 21–24.

Palmer, M., Ng, H. T., and Dang, H. T. (2006). Evaluation of wsd systems. In Agirre, E. and Edmonds, P. (Eds.), *Word Sense Disambiguation: Algorithms and Applications*. Kluwer.

Pantel, P. and Ravichandran, D. (2004). Automatically labeling semantic classes. In *HLT-NAACL-04*, Boston, MA.

Patwardhan, S., Banerjee, S., and Pedersen, T. (2003). Using measures of semantic relatedness for word sense disambiguation. In *Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics*, pp. 241–257. Springer.

Pedersen, T. and Bruce, R. (1997). Distinguishing word senses in untagged text. In *EMNLP 1997*, Providence, RI.

Pedersen, T., Patwardhan, S., and Michelizzi, J. (2004). WordNet::Similarity – Measuring the relatedness of concepts. In *HLT-NAACL-04*.

Pereira, F. C. N., Tishby, N., and Lee, L. (1993). Distributional clustering of English words. In *Proceedings of the 31st ACL*, Columbus, Ohio, pp. 183–190.

Peters, W., Peters, I., , and Vossen, P. (1998). Automatic sense clustering in EuroWordNet. In *LREC-98*, Granada, Spain, pp. 447–454.

Pradhan, S., Hacioglu, K., Ward, W., Martin, J., and Jurafsky, D. (2003). Semantic role parsing: Adding semantic structure to unstructured text. In *Proceedings of the International Conference on Data Mining (ICDM-2003)*.

Pradhan, S., Ward, W., Hacioglu, K., Martin, J., and Jurafsky, D. (2005). Semantic role labeling using different syntactic views. In *ACL-05*, Ann Arbor, MI. ACL.

Quillian, M. R. (1968). Semantic memory. In Minsky, M. (Ed.), *Semantic Information Processing*, pp. 227–270. MIT Press.

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxanomy. In *International Joint Conference for Artificial Intelligence (IJCAI-95)*, pp. 448–453.

Resnik, P. (1996). Selectional constraints: An information-theoretic model and its computational realization. *Cognition*, *61*, 127–159.

Resnik, P. (1997). Selectional preference and sense disambiguation. In *Proceedings of ACL SIGLEX Workshop on Tagging Text with Lexical Semantics*, Washington, D.C., pp. 52–57.

Resnik, P. (1998). Wordnet and class-based probabilities. In Fellbaum, C. (Ed.), *WordNet: An Electronic Lexical Database*. MIT Press.

Resnik, P. (2006). Word sense disambiguation in nlp appplications. In Agirre, E. and Edmonds, P. (Eds.), *Word Sense Disambiguation: Algorithms and Applications*. Kluwer.

Riesbeck, C. K. (1975). Conceptual analysis. In Schank, R. C. (Ed.), *Conceptual Information Processing*, pp. 83–156. American Elsevier, New York.

Riloff, E. (1996). Automatically generating extraction patterns from untagged text. In *AAAI-96*, pp. 117–124.

Rivest, R. L. (1987). Learning decision lists. *Machine Learning*, *2*(3), 229–246.

Rubenstein, H. and Goodenough, J. B. (1965). Contextual correlates of synonymy. *Communications of the ACM*, *8*(10), 627–633.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Schulte im Walde, S. (2000). Clustering verbs semantically according to their alternation behaviour. In *COLING-00*, Saarbrücken, Germany, pp. 747–753.

Schütze, H. (1992a). Context space. In Goldman, R. (Ed.), *Proceedings of the 1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language*.

Schütze, H. (1992b). Dimensions of meaning. In *Proceedings of Supercomputing '92*, pp. 787–796. IEEE, IEEE Press.

Schütze, H. (1997). *Ambiguity Resolution in Language Learning: Computational and Cognitive Models*. CSLI Publications, Stanford, CA.

Schütze, H. (1998). Automatic word sense discrimination. *Computational Linguistics*, *24*(1), 97–124.

Simmons, R. F. (1973). Semantic networks: Their computation and use for understanding English sentences. In Schank, R. C. and Colby, K. M. (Eds.), *Computer Models of Thought and Language*, pp. 61–113. W.H. Freeman and Co., San Francisco.

Small, S. L., Cottrell, G. W., and Tanenhaus, M. (Eds.). (1988). *Lexical Ambiguity Resolution*. Morgan Kaufman.

Small, S. L. and Rieger, C. (1982). Parsing and comprehending with Word Experts. In Lehnert, W. G. and Ringle, M. H. (Eds.), *Strategies for Natural Language Processing*, pp. 89–147. Lawrence Erlbaum.

Snow, R., Jurafsky, D., and Ng, A. Y. (2005). Learning syntactic patterns for automatic hypernym discovery. In Saul, L. K., Weiss, Y., and Bottou, L. (Eds.), *NIPS 17*, pp. 1297–1304. MIT Press.

Snow, R., Jurafsky, D., and Ng, A. Y. (2006). Semantic taxonomy induction from heterogenous evidence. In *COLING/ACL 2006*.

Snow, R., Prakash, S., Jurafsky, D., and Ng, A. Y. (2007). Learning to merge word senses. In *EMNLP/CoNLL 2007*, pp. 1005–1014.

Sparck Jones, K. (1986). *Synonymy and Semantic Classification*. Edinburgh University Press, Edinburgh. Republication of 1964 PhD Thesis.

Stevenson, M. and Wilks, Y. (2001). The interaction of knowledge sources in word sense disambiguation. *Computational Linguistics*, *27*(3), 321–349.

Stevenson, S. and Merlo, P. (1999). Automatic verb classification using distributions of grammatical features. In *EACL-99*, Bergen, Norway, pp. 45–52.

Strube, M. and Ponzetto, S. P. (2006). WikiRelate! Computing semantic relatedness using Wikipedia. In *AAAI-06*, pp. 1419–1424.

Surdeanu, M., Harabagiu, S., Williams, J., and Aarseth, P. (2003). Using predicate-argument structures for information extraction. In *ACL-03*, pp. 8–15.

Swier, R. and Stevenson, S. (2004). Unsupervised semantic role labelling. In *EMNLP 2004*, pp. 95–102.

Tseng, H. (2003). Semantic classification of Chinese unknown words. In *ACL-03*, pp. 72–79. ACL.

Turney, P., Littman, M., Bigham, J., and Shnayder, V. (2003). Combining independent modules to solve multiple-choice synonym and analogy problems. In *Proceedings of RANLP-03*, Borovets, Bulgaria, pp. 482–489.

Vasilescu, F., Langlais, P., and Lapalme, G. (2004). Evaluating variants of the lesk approach for disambiguating words. In *LREC-04*, Lisbon, Portugal, pp. 633–636. ELRA.

Veronis, J. and Ide, N. M. (1990). Word sense disambiguation with very large neural networks extracted from machine readable dictionaries. In *COLING-90*, Helsinki, Finland, pp. 389–394.

Weaver, W. (1949/1955). Translation. In Locke, W. N. and Boothe, A. D. (Eds.), *Machine Translation of Languages*, pp. 15–23. MIT Press. Reprinted from a memorandum written by Weaver in 1949.

Weeds, J. (2003). *Measures and Applications of Lexical Distributional Similarity*. Ph.D. thesis, University of Sussex.

Weeds, J. and Weir, D. (2005). Co-occurrence retrieval: a general framework for lexical distributional similarity. *Computational Linguistics*, *31*(4), 439–476.

Wilks, Y. (1975a). An intelligent analyzer and understander of English. *Communications of the ACM*, *18*(5), 264–274.

Wilks, Y. (1975b). Preference semantics. In Keenan, E. L. (Ed.), *The Formal Semantics of Natural Language*, pp. 329–350. Cambridge Univ. Press.

Wilks, Y. (1975c). A preferential, pattern-seeking, semantics for natural language inference. *Artificial Intelligence*, *6*(1), 53–74.

Wilks, Y. (1978). Making preferences more active. *Artificial Intelligence*, *11*(3), 197–223.

Wilks, Y., Slator, B. M., and Guthrie, L. M. (1996). *Electric Words: Dictionaries, Computers, and Meanings*. MIT Press.

Wu, Z. and Palmer, M. (1994). Verb semantics and lexical selection. In *Proceedings of the 32nd ACL*, Las Cruces, NM, pp. 133–138.

Xue, N. and Palmer, M. (2004). Calibrating features for semantic role labeling. In *EMNLP 2004*.

Yarowsky, D. (1994). Decision lists for lexical ambiguity resolution: Application to accent restoration in Spanish and French. In *Proceedings of the 32nd ACL*, Las Cruces, NM, pp. 88–95. ACL.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *ACL-95*, Cambridge, MA, pp. 189–196. ACL.

Yarowsky, D. (1997). Homograph disambiguation in text-to-speech synthesis. In van Santen, J. P. H., Sproat, R., Olive, J. P., and Hirschberg, J. (Eds.), *Progress in Speech Synthesis*, pp. 157–172. Springer.

Yuret, D. (2004). Some experiments with a Naive Bayes WSD system. In *Senseval-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*.

Zernik, U. (1991). Train1 vs. train2: Tagging word senses in corpus. In *Lexical Acquisition: Exploiting On-Line Resources to Build a Lexicon*, pp. 91–112. Lawrence Erlbaum.

21 COMPUTATIONAL DISCOURSE

> Gracie: Oh yeah. . . and then Mr. and Mrs. Jones were having matrimonial trouble, and my brother was hired to watch Mrs. Jones.
> George: Well, I imagine she was a very attractive woman.
> Gracie: She was, and my brother watched her day and night for six months.
> George: Well, what happened?
> Gracie: She finally got a divorce.
> George: Mrs. Jones?
> Gracie: No, my brother's wife.
>
> George Burns and Gracie Allen in *The Salesgirl*

Orson Welles' movie *Citizen Kane* was groundbreaking in many ways, perhaps most notably in its structure. The story of the life of fictional media magnate Charles Foster Kane, the movie does not proceed in chronological order through Kane's life. Instead, the film begins with Kane's death, (famously murmuring "*Rosebud*"), and is structured around flashbacks to his life inserted among scenes of a reporter investigating his death. The novel idea that the structure of a movie does not have to linearly follow the structure of the real timeline made apparent for 20th century cinematography the infinite possibilities and impact of different kinds of coherent narrative structures.

But coherent structure is not just a fact about movies, or works of art. Up to this point of the book, we have focused primarily on language phenomena that operate at the word or sentence level. But just like movies, language does not normally consist of isolated, unrelated sentences, but instead of collocated, structured, **coherent** groups of sentences. We refer to such a coherent structured group of sentences as a **discourse**. DISCOURSE

The chapter you are now reading is an example of a discourse. It is in fact a discourse of a particular sort: a **monologue**. Monologues are characterized by a *speaker* (a term which will be used to include writers, as it is here), and a *hearer* (which, analogously, includes readers). The communication flows in only one direction in a monologue, that is, from the speaker to the hearer. MONOLOGUE

After reading this chapter, you may have a conversation with a friend about it, which would consist of a much freer interchange. Such a discourse is called a **dialogue**, specifically a **human-human dialogue**. In this case, each participant periodically takes DIALOGUE turns being a speaker and hearer. Unlike a typical monologue, dialogues generally consist of many different types of communicative acts: asking questions, giving answers, making corrections, and so forth.

HCI

You may also, for some purposes, such as booking an airline or train trip, have a conversation with a computer **conversational agent**. This use of **human-computer dialogue** for *human-computer interaction*, or **HCI** has properties that distinguish it from normal human-human dialogue, in part due to the present-day limitations on the ability of computer systems to participate in free, unconstrained conversation.

While many discourse processing problems are common to these three forms of discourse, they differ in enough respects that different techniques have often been used to process them. This chapter focuses on techniques commonly applied to the interpretation of monologues; techniques for conversational agents and other dialogues will be described in Ch. 24.

Language is rife with phenomena that operate at the discourse level. Consider the discourse shown in example (21.1).

(21.1) The Tin Woodman went to the Emerald City to see the Wizard of Oz and ask for a heart. After he asked for it, the Woodman waited for the Wizard's response.

What do pronouns such as *he* and *it* denote? No doubt the reader had little trouble figuring out that *he* denotes the Tin Woodman and not the Wizard of Oz, and that *it* denotes the heart and not the Emerald City. Furthermore, it is clear to the reader that *the Wizard* is the same entity as *the Wizard of Oz*, and *the Woodman* is the same as *the Tin Woodman*.

But doing this disambiguation automatically is a difficult task. This goal of deciding what pronouns and other noun phrases refer to is called **coreference resolution**. Coreference resolution is important for **information extraction**, **summarization**, and for **conversational agents**. In fact, it turns out that just about any conceivable language processing application requires methods for determining the denotations of pronouns and related expressions.

There are other important discourse structures beside the relationships between pronouns and other nouns. Consider the task of **summarizing** the following news passage:

(21.2) First Union Corp is continuing to wrestle with severe problems. According to industry insiders at Paine Webber, their president, John R. Georgius, is planning to announce his retirement tomorrow.

We might want to extract a summary like the following:

(21.3) First Union President John R. Georgius is planning to announce his retirement tomorrow.

In order to build such a summary, we need to know that the second sentence is the more important of the two, and that the first sentence is subordinate to it, just giving background information. Relationships of this sort between sentences in a discourse are called **coherence relations**, and determining the coherence structures between discourse sentences is an important discourse task.

Since **coherence** is also a property of a good text, automatically detecting coherence relations is also useful for tasks that measure text quality, like **automatic essay**

grading. In automatic essay grading, short student essays are assigned a grade by measuring the internal coherence of the essay as well as comparing its content to source material and hand-labeled high-quality essays. Coherence is also used to evaluate the output quality of natural language generation systems.

Discourse structure and coreference are related in deep ways. Notice that in order to perform the summary above, a system must correctly identify *First Union Corp* as the denotation of *their* (as opposed to *Paine Webber*, for instance). Similarly, it turns out that determining the discourse structure can help in determining coreference.

Coherence

Let's conclude this introduction by discussing what it means for a text to be **coherent**. Assume that you have collected an arbitrary set of well-formed and independently interpretable utterances, for instance, by randomly selecting one sentence from each of the previous chapters of this book. Do you have a discourse? Almost certainly not. The reason is that these utterances, when juxtaposed, will not exhibit **coherence**. Consider, for example, the difference between passages (21.4) and (21.5).

COHERENCE

(21.4) John hid Bill's car keys. He was drunk.

(21.5) ?? John hid Bill's car keys. He likes spinach.

While most people find passage (21.4) to be rather unremarkable, they find passage (21.5) to be odd. Why is this so? Like passage (21.4), the sentences that make up passage (21.5) are well formed and readily interpretable. Something instead seems to be wrong with the fact that the sentences are juxtaposed. The hearer might ask, for instance, what hiding someone's car keys has to do with liking spinach. By asking this, the hearer is questioning the coherence of the passage.

Alternatively, the hearer might try to construct an explanation that makes it coherent, for instance, by conjecturing that perhaps someone offered John spinach in exchange for hiding Bill's car keys. In fact, if we consider a context in which we had known this already, the passage now sounds a lot better! Why is this? This conjecture allows the hearer to identify John's liking spinach as the cause of his hiding Bill's car keys, which would explain how the two sentences are connected. The very fact that hearers try to identify such connections is indicative of the need to establish coherence as part of discourse comprehension.

In passage (21.4), or in our new model of passage (21.5), the second sentence offers the reader an EXPLANATION or CAUSE for the first sentence. These examples show that a coherent discourse must have meaningful connections between its utterances, connections like EXPLANATION that are often called **coherence relations** and will be introduced in Sec. 21.2.

COHERENCE RELATIONS

Let's introduce a second aspect of coherence by considering the following two texts from Grosz et al. (1995a):

(21.6) a. John went to his favorite music store to buy a piano.
b. He had frequented the store for many years.
c. He was excited that he could finally buy a piano.
d. He arrived just as the store was closing for the day.

(21.7) a. John went to his favorite music store to buy a piano.
b. It was a store John had frequented for many years.
c. He was excited that he could finally buy a piano.
d. It was closing just as John arrived.

While these two texts differ only in how the two entities (John and the store) are realized in the sentences, the discourse in (21.6) is intuitively more coherent than the one in (21.7). As Grosz et al. (1995a) point out, this is because the discourse in (21.6) is clearly about one individual, John, describing his actions and feelings. The discourse in (21.7), by contrast, focuses first on John, then the store, then back to John, then to the store again. It lacks the 'aboutness' of the first discourse.

These examples show that for a discourse to be coherent it must exhibit certain kinds of relationships with the entities it is about, introducing them and following them in a focused way. This kind of coherence can be called **entity-based coherence**, We will introduce the **Centering** model of entity-based coherence in Sec. 21.6.2.

In the rest of the chapter we'll study aspects of both discourse structure and discourse entities. We begin in Sec. 21.1 with the simplest kind of discourse structure: simple **discourse segmentation** of a document into a linear sequence of multiparagraph passages. In Section 21.2, we then introduce more fine-grained discourse structure, the **coherence relation**, and give some algorithms for interpreting these relations. Finally, in Section 21.3, we turn to entities, describing methods for interpreting *referring expressions* such as pronouns.

21.1 Discourse Segmentation

The first kind of discourse task we examine is an approximation to the global or high-level structure of a text or discourse. Many genres of text are associated with particular conventional structures. Academic articles might be divided into sections like Abstract, Introduction, Methodology, Results, Conclusion. A newspaper story is often described as having an inverted pyramid structure, in which the opening paragraphs (the **lede**) contains the most important information. Spoken patient reports are dictated by doctors in four sections following the standard SOAP format (Subjective, Objective, Assessment, Plan).

LEDE

Automatically determining all of these types of structures for a large discourse is a difficult and unsolved problem. But some kinds of discourse structure detection algorithms exist. This section introduces one such algorithm, for the simpler problem of **discourse segmentation**; separating a document into a linear sequence of subtopics. Such segmentation algorithms are unable to find sophisticated hierarchical structure. Nonetheless, linear discourse segmentation can be important for **information retrieval**, for example, for automatically segmenting a TV news broadcast or a long news story into a sequence of stories so as to find a relevant story, or for **text summarization** algorithms which need to make sure that different segments of the document are summarized correctly, or for **information extraction** algorithms which tend to extract information from inside a single discourse segment.

DISCOURSE SEGMENTATION

In the next two sections we introduce both an unsupervised and a supervised algorithm for discourse segmentation.

21.1.1 Unsupervised Discourse Segmentation

LINEAR SEGMENTATION

Let's consider the task of segmenting a text into multi-paragraph units that represent subtopics or passages of the original text. As we suggested above, this task is often called **linear segmentation**, to distinguish it from the task of deriving more sophisticated hierarchical discourse structure. The goal of a segmenter, given raw text, might be to assign subtopic groupings such as the ones defined by Hearst (1997) for the following 21-paragraph science news article called *Stargazers* on the existence of life on earth and other planets (numbers indicate paragraphs):

l-3	Intro - the search for life in space
4–5	The moon's chemical composition
6-8	How early earth-moon proximity shaped the moon
9–12	How the moon helped life evolve on earth
13	Improbability of the earth-moon system
14–16	Binary/trinary star systems make life unlikely
17–18	The low probability of nonbinary/trinary systems
19–20	Properties of earth's sun that facilitate life
21	Summary

COHESION

LEXICAL COHESION

An important class of unsupervised algorithms for the linear discourse segmentation task rely on the concept of **cohesion** (Halliday and Hasan, 1976). **Cohesion** is the use of certain linguistic devices to link or tie together textual units. **Lexical cohesion** is cohesion indicated by relations between words in the two units, such as use of an identical word, a synonym, or a hypernym. For example the fact that the words *house*, *shingled*, and *I* occur in both of the two sentences in (21.8ab), is a cue that the two are tied together as a discourse:

(21.8)
- Before winter **I** built a chimney, and **shingled** the sides of my **house**...
- **I** have thus a tight **shingled** and plastered **house**

In Ex. (21.9), lexical cohesion between the two sentences is indicated by the hypernym relation between *fruit* and the words *pears* and *apples*.

(21.9) Peel, core and slice **the pears and the apples**. Add **the fruit** to the skillet.

There are also non-lexical cohesion relations, such as the use of **anaphora**, shown here between *Woodhouses* and *them* (we will define and discuss anaphora in detail in Sec. 21.6):

(21.10) **The Woodhouses** were first in consequence there. All looked up to **them**.

COHESION CHAIN

In addition to single examples of lexical cohesion between two words, we can have a **cohesion chain**, in which cohesion is indicated by a whole sequence of related words:

(21.11) Peel, core and slice **the pears and the apples**. Add **the fruit** to the skillet. When **they** are soft...

Coherence and **cohesion** are often confused; let's review the difference. **Cohesion** refers to the way textual units are tied or linked together. A cohesive relation is like a kind of glue grouping together two units into a single unit. **Coherence** refers to the *meaning* relation between the two units. A coherence relation explains how the meaning of different textual units can combine to jointly build a discourse meaning for the larger unit.

The intuition of the cohesion-based approach to segmentation is that sentences or paragraphs in a subtopic are cohesive with each other, but not with paragraphs in a neighboring subtopic. Thus if we measured the cohesion between every neighboring sentence, we might expect a 'dip' in cohesion at subtopic boundaries.

TEXTTILING

Let's look at one such cohesion-based approach, the **TextTiling** algorithm (Hearst, 1997). The algorithm has three steps: **tokenization**, **lexical score determination**, and **boundary identification**. In the tokenization stage, each space-delimited word in the input is converted to lower-case, words in a stop list of function words are thrown out, and the remaining words are morphologically stemmed. The stemmed words are grouped into pseudo-sentences of length $w = 20$ (equal-length pseudo-sentences are used rather than real sentences).

Now we look at each gap between pseudo-sentences, and compute a **lexical cohesion score** across that gap. The cohesion score is defined as the average similarity of the words in the pseudo-sentences before gap to the pseudo-sentences after the gap. We generally use a block of $k = 10$ pseudo-sentences on each side of the gap. To compute similarity, we create a word vector b from the block before the gap, and a vector a from the block after the gap, where the vectors are of length N (the total number of non-stop words in the document) and the ith element of the word vector is the frequency of the word w_i. Now we can compute similarity by the cosine (= normalized dot product) measure defined in Eq. (**??**) from Ch. 20, rewritten here:

$$\text{sim}_{\text{cosine}}(\vec{b}, \vec{a}) = \frac{\vec{b} \cdot \vec{a}}{|\vec{b}||\vec{a}|} = \frac{\sum_{i=1}^{N} b_i \times a_i}{\sqrt{\sum_{i=1}^{N} b_i^2}\sqrt{\sum_{i=1}^{N} a_i^2}} \tag{21.12}$$

This similarity score (measuring how similar pseudo-sentences $i - k$ to i are to sentences $i + 1$ to $i + k + 1$) is computed for each gap i between pseudo-sentences. Let's look at the example in Fig. 21.1, where $k = 2$. Fig. 21.1a shows a schematic view of four pseudo-sentences. Each 20-word pseudo-sentence might have multiple true sentences in it; we've shown each with two true sentences. The figure also indicates the computation of the dot-product between successive pseudosentences. Thus for example in the first pseudo-sentence, consisting of sentences 1 and 2, the word A occurs twice, B once, C twice, and so on. The dot product between the first two pseudosentences is $2 \times 1 + 1 \times 1 + 2 \times 1 + 1 \times 1 + 2 \times 1 = 8$. What is the cosine between these first two, assuming all words not shown have zero count?

Finally, we compute a **depth score** for each gap, measuring the depth of the 'similarity valley' at the gap. The depth score is the distance from the peaks on both sides of the valley to the valley; In Fig. 21.1(b), this would be $(y_{a_1} - y_{a_2}) + (y_{a_3} - y_{a_2})$.

Boundaries are assigned at any valley which is deeper than a cutoff threshold (such as $\bar{s} - \sigma$, i.e. one standard deviation deeper than the mean valley depth).

Instead of using these depth score thresholds, more recent cohesion-based segmenters use **divisive clustering** (Choi, 2000; Choi et al., 2001); see the end of the chapter for more information.

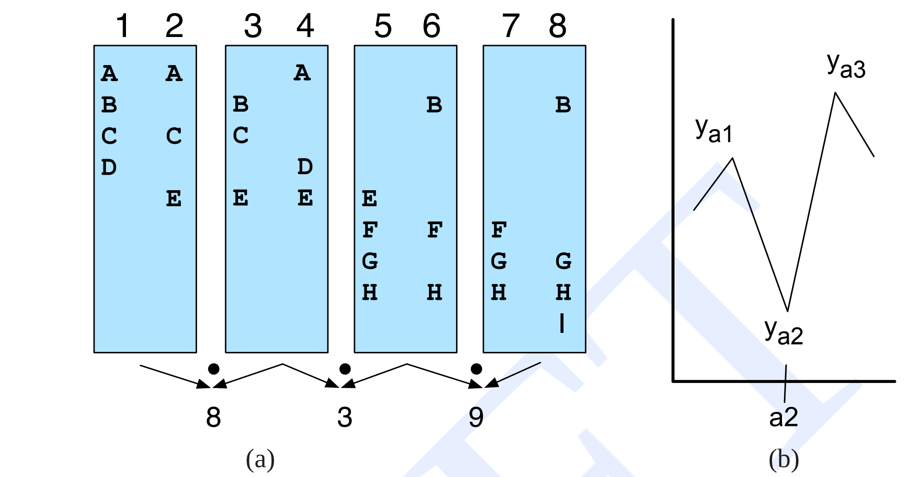


Figure 21.1 The TextTiling algorithm, showing (a) the dot-product computation of similarity between two sentences (1 and 2) and 2 following sentences (3 and 4); capital letters (A, B, C, etc) indicate occurrences of words. (b) shows the computation of the depth score of a valley. After Hearst (1997).

21.1.2 Supervised Discourse Segmentation

We've now seen a method for segmenting discourses when no hand-labeled segment boundaries exist. For some kinds of discourse segmentation tasks, however, it is relatively easy to acquire boundary-labeled training data.

PARAGRAPH SEGMENTATION

Consider the spoken discourse task of segmentation of broadcast news. In order to do summarization of radio or TV broadcasts, we first need to assign boundaries between news stories. This is a simple discourse segmentation task, and training sets with hand-labeled news story boundaries exist. Similarly, for speech recognition of monologues like lectures or speeches, we often want to automatically break the text up into paragraphs. For the task of **paragraph segmentation**, it is trivial to find labeled training data from the web (marked with <p>) or other sources.

Every kind of classifier has been used for this kind of supervised discourse segmentation. For example, we can use a binary classifier (SVM, decision tree) and make a yes-no boundary decision between any two sentences. We can also use a sequence classifier (HMM, CRF), making it easier to incorporate sequential constraints.

The features in supervised segmentation are generally a superset of those used in unsupervised classification. We can certainly use cohesion features such as word overlap, word cosine, LSA, lexical chains, coreference, and so on.

DISCOURSE MARKERS

CUE WORDS

A key additional feature that is often used for supervised segmentation is the presence of **discourse markers** or **cue words**. A discourse marker is a word or phrase that functions to signal discourse structure. Discourse markers will play an important role throughout this chapter. For the purpose of broadcast news segmentation, important discourse markers might include a phrase like *good evening, I'm* ⟨*PERSON*⟩, which tends to occur at the beginning of broadcasts, or the word *joining*, which tends to occur in the phrase *joining us now is* ⟨*PERSON*⟩, which often occurs at beginnings of specific segments. Similarly, the cue phrase *coming up* often appears at the end of segments (Reynar, 1999; Beeferman et al., 1999).

Discourse markers tend to be very domain-specific. For the task of segmenting newspaper articles from the Wall Street Journal, for example, the word *incorporated* is a useful feature, since Wall Street Journal articles often start by introducing a company with the full name *XYZ Incorporated*, but later using just *XYZ*. For the task of segmenting out real estate ads, Manning (1998) used discourse cue features like *'is the following word a neighborhood name?'*, *'is previous word a phone number?'* and even punctuation cues like *'is the following word capitalized?'*.

It is possible to write hand-written rules or regular expressions to identify discourse markers for a given domain. Such rules often refer to named entities (like the PERSON examples above), and so a named entity tagger must be run as a preprocessor. Automatic methods for finding discourse markers for segmentation also exist. They first encode all possible words or phrases as features to a classifier, and then doing some sort of **feature selection** on the training set to find only the words that are the best indicators of a boundary (Beeferman et al., 1999; Kawahara et al., 2004).

21.1.3 Evaluating Discourse Segmentation

Discourse segmentation is generally evaluated by running the algorithm on a test set in which boundaries have been labeled by humans. The performance of the algorithm is computed by comparing the automatic and human boundary labels using the *WindowDiff* (Pevzner and Hearst, 2002) or P_k (Beeferman et al., 1999) metrics.

We generally don't use precision, recall and F-measure for evaluating segmentation because they are not sensitive to near misses. Using standard F-measure, if our algorithm was off by one sentence in assigning each boundary, it would get as bad a score as an algorithm which assigned boundaries nowhere near the correct locations. Both *WindowDiff* and P_k assign partial credit. We will present WindowDiff, since it is a more recent improvement to P_k.

WindowDiff compares a reference (human labeled) segmentation with a hypothesis segmentation by sliding a probe, a moving window of length k, across the hypothesis segmentation. At each position in the hypothesis string, we compare the number of **reference** boundaries that fall within the probe (r_i) to the number of **hypothesized** boundaries that fall within the probe (h_i). The algorithm penalizes any hypothesis for which $r_i \neq h_i$, i.e. for which $|r_i - h_i| \neq 0$. The window size k is set as half the average segment in the reference string. Fig. 21.2 shows a schematic of the computation.

More formally, if $b(i,j)$ is the number of boundaries between positions i and j in a text, and N is the number of sentences in the text:

$$\text{WindowDiff}(\mathit{ref},\mathit{hyp}) = \frac{1}{N-k}\sum_{i=1}^{N-k}(|b(\mathit{ref}_i,\mathit{ref}_{i+k}) - b(\mathit{hyp}_i,\mathit{hyp}_{i+k})| \neq 0) \tag{21.13}$$

WindowDiff returns a value between 0 and 1, where 0 indicates that all boundaries are assigned correctly.

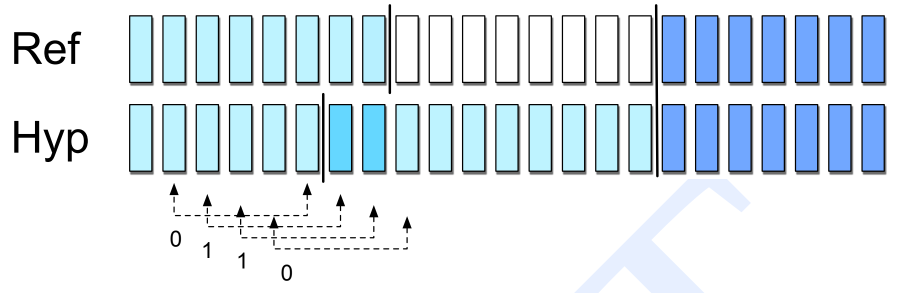


Figure 21.2 The WindowDiff algorithm, showing the moving window sliding over the hypothesis string, and the computation of $|r_i - h_i|$ at four positions. After Pevzner and Hearst (2002).

21.2 Text Coherence

The previous section showed that cohesive devices, like lexical repetition, can be used to find structure in a discourse. The existence of such devices alone, however, does not satisfy a stronger requirement that a discourse must meet, that of being *coherent*. We briefly introduced coherence in the introduction. In this section we offer more details on what it means for a text to be coherent, and computational mechanisms for determining coherence. We will focus on **coherence relations** and reserve **entity-based coherence** for discussion in Sec. 21.6.2.

Recall from the introduction the difference between passages (21.14) and (21.15).

(21.14) John hid Bill's car keys. He was drunk.

(21.15) ?? John hid Bill's car keys. He likes spinach.

The reason (21.14) is more coherent is that the reader can form a connection between the two utterances, in which the second utterance provides a potential CAUSE or EXPLANATION for the first utterance. This link is harder to form for (21.15). The possible connections between utterances in a discourse can be specified as a set of **coherence relations**. A few such relations, proposed by Hobbs (1979), are given below. The terms S_0 and S_1 represent the meanings of the two sentences being related.

COHERENCE RELATIONS

Result: Infer that the state or event asserted by S_0 causes or could cause the state or event asserted by S_1.

(21.16) The Tin Woodman was caught in the rain. His joints rusted.

Explanation: Infer that the state or event asserted by S_1 causes or could cause the state or event asserted by S_0.

(21.17) John hid Bill's car keys. He was drunk.

Parallel: Infer $p(a_1, a_2, ...)$ from the assertion of S_0 and $p(b_1, b_2, ...)$ from the assertion of S_1, where a_i and b_i are similar, for all i.

(21.18) The Scarecrow wanted some brains. The Tin Woodman wanted a heart.

Elaboration: Infer the same proposition P from the assertions of S_0 and S_1.

(21.19) Dorothy was from Kansas. She lived in the midst of the great Kansas prairies.

Occasion: A change of state can be inferred from the assertion of S_0, whose final state can be inferred from S_1, or a change of state can be inferred from the assertion of S_1, whose initial state can be inferred from S_0.

(21.20) Dorothy picked up the oil-can. She oiled the Tin Woodman's joints.

We can also talk about the coherence of an entire discourse, by considering the hierarchical structure between coherence relations. Consider passage (21.21).

(21.21) John went to the bank to deposit his paycheck. (S1)
He then took a train to Bill's car dealership. (S2)
He needed to buy a car. (S3)
The company he works for now isn't near any public transportation. (S4)
He also wanted to talk to Bill about their softball league. (S5)

Intuitively, the structure of passage (21.21) is not linear. The discourse seems to be primarily about the sequence of events described in sentences S1 and S2, whereas sentences S3 and S5 are related most directly to S2, and S4 is related most directly to S3. The coherence relationships between these sentences result in the discourse structure shown in Figure 21.3.

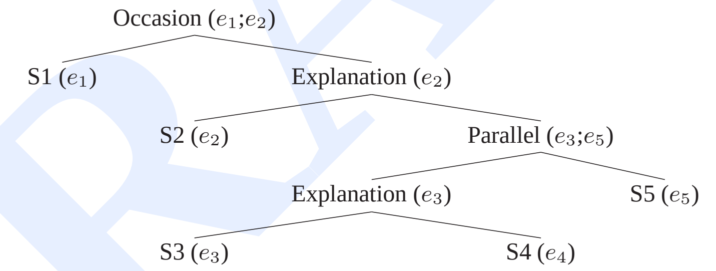


Figure 21.3 The discourse structure of passage (21.21).

DISCOURSE SEGMENT

Each node in the tree represents a group of locally coherent clauses or sentences, called a **discourse segment**. Roughly speaking, one can think of discourse segments as being analogous to constituents in sentence syntax.

Now that we've seen examples of coherence, we can see more clearly how a coherence relation can play a role in summarization or information extraction. For example, discourses that are coherent by virtue of the Elaboration relation are often characterized by a summary sentence followed by one or more sentences adding detail to it, as in passage (21.19). Although there are two sentences describing events in this passage, the Elaboration relation tells us that the same event is being described in each. Automatic labeling of the Elaboration relation could thus tell an information extraction or summarization system to merge the information from the sentences and produce a single event description instead of two.

21.2.1 Rhetorical Structure Theory

RHETORICAL STRUCTURE THEORY
RST

Another theory of coherence relations that has received broad usage is **Rhetorical Structure Theory** (**RST**), a model of text organization that was originally proposed for the study of text generation (Mann and Thompson, 1987).

NUCLEUS
SATELLITE

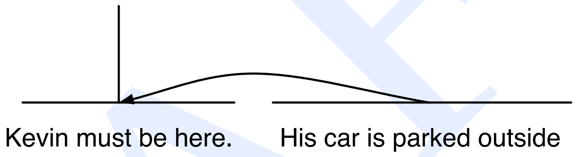
RST is based on a set of 23 *rhetorical relations* that can hold between spans of text within a discourse. Most relations hold between two text spans (often clauses or sentences), a **nucleus** and a **satellite**. The nucleus is the unit that is more central to the writer's purpose, and that is interpretable independently; the satellite is less central, and generally is only interpretable with respect to the nucleus.

EVIDENCE

Consider the **Evidence** relation, in which a satellite presents evidence for the proposition or situation expressed in the nucleus:

(21.22) Kevin must be here. His car is parked outside.

RST relations are traditionally represented graphically; the asymmetric Nucleus-Satellite relation is represented with an arrow from the satellite to the nucleus:



In the original (Mann and Thompson, 1987) formulation, an RST relation is formally defined by a set of **constraints** on the nucleus and satellite, having to do with the goals and beliefs of the writer (W) and reader (R), and by the **effect** on the reader (R). The Evidence relation, for example, is defined as follows:

Relation Name:	Evidence
Constraints on N:	R might not believe N to a degree satisfactory to W
Constraints on S:	R believes S or will find it credible
Constraints on N+S:	R's comprehending S increases R's belief of N
Effects:	R's belief of N is increased

There are many different sets of rhetorical relations in RST and related theories and implementations. The RST TreeBank (Carlson et al., 2001), for example, defines 78 distinct relations, grouped into 16 classes. Here are some common RST relations, with definitions adapted from Carlson and Marcu (2001).

Elaboration: There are various kinds of elaboration relations; in each one, the satellite gives further information about the content of the nucleus:

[$_N$ The company wouldn't elaborate,] [$_S$ citing competitive reasons]

Attribution: The satellite gives the source of attribution for an instance of reported speech in the nucleus.

[$_S$ Analysts estimated,] [$_N$ that sales at U.S. stores declined in the quarter, too]

Contrast: This is a multinuclear relation, in which two or more nuclei contrast along some important dimension:

[$_N$ The priest was in a very bad temper,] [$_N$ but the lama was quite happy.]

List: In this multinuclear relation, a series of nuclei is given, without contrast or explicit comparison:

[$_N$ Billy Bones was the mate;] [$_N$ Long John, he was quartermaster]

Background: The satellite gives context for interpreting the nucleus:

[$_S$ T is the pointer to the root of a binary tree.] [$_N$ Initialize T.]

Just as we saw for the Hobbs coherence relations, RST relations can be hierarchically organized into an entire discourse tree. Fig. 21.4 shows one from Marcu (2000a) for the text in (21.23) from the Scientific American magazine.

(21.23) With its distant orbit–50 percent farther from the sun than Earth–and slim atmospheric blanket, Mars experiences frigid weather conditions. Surface temperatures typically average about -60 degrees Celsius (-76 degrees Fahrenheit) at the equator and can dip to -123 degrees C near the poles. Only the midday sun at tropical latitudes is warm enough to thaw ice on occasion, but any liquid water formed in this way would evaporate almost instantly because of the low atmospheric pressure.

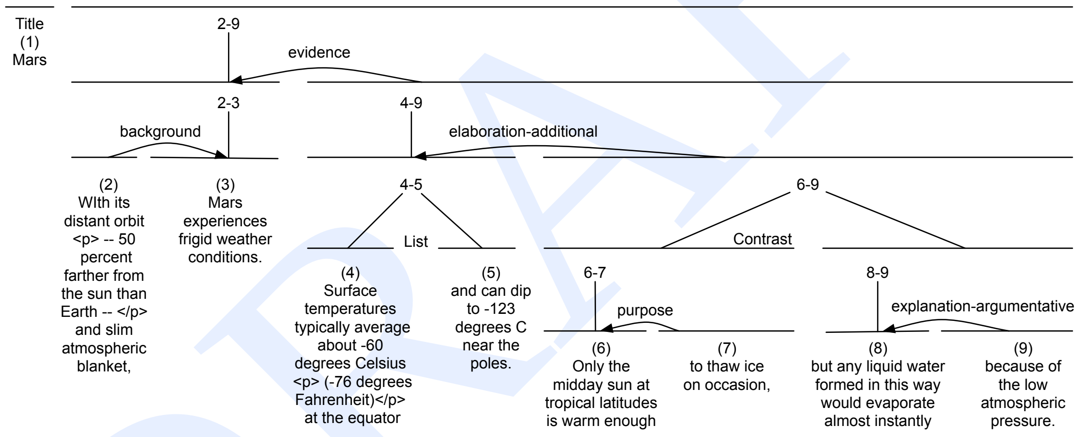

Title
(1)
Mars
2-9
evidence
2-3
background
4-9
elaboration-additional
(2)
WIth its distant orbit <p> -- 50 percent farther from the sun than Earth -- </p> and slim atmospheric blanket,
(3)
Mars experiences frigid weather conditions.
4-5
List
6-9
Contrast
(4)
Surface temperatures typically average about -60 degrees Celsius <p> (-76 degrees Fahrenheit)</p> at the equator
(5)
and can dip to -123 degrees C near the poles.
6-7
purpose
8-9
explanation-argumentative
(6)
Only the midday sun at tropical latitudes is warm enough
(7)
to thaw ice on occasion,
(8)
but any liquid water formed in this way would evaporate almost instantly
(9)
because of the low atmospheric pressure.

Figure 21.4 A discourse tree for the Scientific American text in (21.23), from Marcu (2000a). Note that asymmetric relations are represented with a curved arrow from the satellite to the nucleus.

See the end of the chapter for pointers to other theories of coherence relations and related corpora, and Ch. 23 for the application of RST and similar coherence relations to summarization.

21.2.2 Automatic Coherence Assignment

Given a sequence of sentences, how can we automatically determine the coherence relations between them? Whether we use RST, Hobbs, or one of the many other sets of relations (see the end of the chapter), we call this task **coherence relation assignment**. If we extend this task from assigning a relation between two sentences to the larger goal of extracting a tree or graph representing an entire discourse, the term **discourse parsing** is often used.

DISCOURSE PARSING

Both of these tasks are quite difficult, and remain unsolved open research problems. Nonetheless, a variety of methods have been proposed, and in this section we describe shallow algorithms based on **cue phrases**. In the following section we sketch a more sophisticated but less robust algorithm based on **abduction**.

A shallow cue-phrase-based algorithm for coherence extraction has three stages:

1. Identify the cue phrases in a text
2. Segment the text into discourse segments, using cue phrases
3. Classify the relationship between each consecutive discourse segment, using cue phrases.

CUE PHRASE
DISCOURSE MARKER

We said earlier that a **cue phrase** (or **discourse marker** or **cue word**) is a word or phrase that functions to signal discourse structure, especially by linking together discourse segments. In Sec. 21.1 we mentioned cue phrases or features like *joining us now is* ⟨*PERSON*⟩ (for broadcast news segmentation) or *following word is the name of a neighborhood* (for real estate ad segmentation). For extracting coherence relations, we rely on cue phrases called **connectives**, which are often conjunctions or adverbs, and which give us a 'cue' to the coherence relations that hold between segments. For example, the connective *because* strongly suggests the EXPLANATION relation in passage (21.24).

CONNECTIVES

(21.24) John hid Bill's car keys because he was drunk.

Other such cue phrases include *although*, *but*, *for example*, *yet*, *with*, and *and*. Discourse markers can be quite ambiguous between these **discourse** uses and non-discourse related **sentential** uses. For example, the word *with* can be used as a cue phrase as in (21.25), or in a sentential use as in (21.26)[1]:

SENTENTIAL

(21.25) **With** its distant orbit, Mars exhibits frigid weather conditions

(21.26) We can see Mars **with** an ordinary telescope.

Some simple disambiguation of the discourse versus sentential use of a cue phrase can be done with simple regular expressions, once we have sentence boundaries. For example, if the words *With* or *Yet* are capitalized and sentence-initial, they tend to be discourse markers. The words *because* or *where* tend to be discourse markers if preceded by a comma. More complete disambiguation requires the WSD techniques of Ch. 20 using many other features. If speech is available, for example, discourse markers often bear different kinds of pitch accent than sentential uses (Hirschberg and Litman, 1993).

The second step in determining the correct coherence relation is to segment the text into **discourse segments**. Discourse segments generally correspond to clauses or sentences, although sometimes they are smaller than clauses. Many algorithms approximate segmentation by using entire sentences, employing the sentence segmentation algorithm of Fig. **??** (page **??**), or the algorithm of Sec. **??**.

Often, however, a clause or clause-like unit is a more appropriate size for a discourse segment, as we see in the following examples from Sporleder and Lapata (2004):

(21.27) [We can't win] [but we must keep trying] (CONTRAST)

[1] Where perhaps it will be a cue instead for the semantic role INSTRUMENT

(21.28) [The ability to operate at these temperature is advantageous], [because the devices need less thermal insulation] (EXPLANATION)

One way to segment these clause-like units is to use hand-written segmentation rules based on individual cue phrases. For example, if the cue-phrase *Because* occurs sentence-initially and is eventually followed by a comma (as in (21.29)), it may begin a segment (terminated by the comma) that relates to the clause after the comma. If *because* occurs sentence-medially, it may divide the sentence into a previous and following discourse segment (as in (21.30)). These cases can be distinguished by hand-written rules based on punctuation and sentence boundaries.

(21.29) [Because of the low atmospheric pressure,] [any liquid water would evaporate instantly]

(21.30) [Any liquid water would evaporate instantly] [because of the low atmospheric pressure.]

If a syntactic parser is available, we can write more complex segmentation rules making use of syntactic phrases.

The third step in coherence extraction is to automatically classify the relation between each pair of neighboring segments. We can again write rules for each discourse marker, just as we did for determining discourse segment boundaries. Thus a rule could specify that a segmenting beginning with sentence-initial *Because* is a satellite in a CAUSE relationship with a nucleus segment that follows the comma.

In general, the rule-based approach to coherence extraction does not achieve extremely high accuracy. Partly this is because cue phrases are ambiguous; *because*, for example, can indicate both CAUSE and EVIDENCE, *but* can indicate CONTRAST, ANTITHESIS, and CONCESSION, and so on. We need additional features than just the cue phrases themselves. But a deeper problem with the rule-based method is that many coherence relations are not signaled by cue phrases at all. In the RST corpus of Carlson et al. (2001), for example, Marcu and Echihabi (2002) found that only 61 of the 238 CONTRAST relations, and only 79 of the 307 EXPLANATION-EVIDENCE relations, were indicated by explicit cue phrases. Instead, many coherence relations are signalled by more implicit cues. For example, the following two sentences are in the CONTRAST relation, but there is no explicit *in contrast* or *but* connective beginning the second sentence:

(21.31) The $6 billion that some 40 companies are looking to raise in the year ending March 31 compares with only $2.7 billion raised on the capital market in the previous fiscal year

(21.32) In fiscal 1984 before Mr. Gandhi came to power, only $810 million was raised.

How can we extract coherence relations between discourse segments if no cue phrases exist? There are certainly many implicit cues that we could use. Consider the following two discourse segments:

(21.33) [I don't want a truck;] [I'd prefer a convertible.]

The CONTRAST relation between these segments is signalled by their syntactic parallelism, by the use of negation in the first segment, and by the lexical coordinate relation between *convertible* and *truck*. But many of these features are quite lexical, requiring a large number of parameters which couldn't be trained on the small amount of labeled coherence relation data that currently exists.

This suggests the use of **bootstrapping** to automatically label a larger corpus with coherence relations that could then be used to train these more expensive features. We can do this by relying on discourse markers that are very strong unambiguous cues for particular relations. For example *consequently* is an unambiguous signal for RESULT, *in other words* for SUMMARY, *for example* for ELABORATION, and *secondly* for CONTINUATION. We write regular expressions to extract pairs of discourse segments surrounding these cue phrases, and then remove the cue phrases themselves. The resulting sentence pairs, without the cue phrases, are used as a supervised training set for these coherence relations.

Given this labeled training set, any supervised machine learning method may be used. Marcu and Echihabi (2002), for example, use a naive Bayes classifier based only on word-pair features (w_1, w_2), where the first word w_1 occurs in the first discourse segment, and the second w_2 occurs in the following segment. This feature captures lexical relations like *convertible*/*truck* above. Sporleder and Lascarides (2005) include other features, including individual words, parts of speech, or stemmed words in the left and right discourse segment. They found, for example, that words like *other*, *still*, and *not* were chosen by feature selection as good cues for CONTRAST. Words like *so*, *indeed*, and *undoubtedly* were chosen as cues for RESULT.

21.3 Reference Resolution

> and even Stigand, the patriotic archbishop of Canterbury, found it advisable–"'
> 'Found WHAT?' said the Duck.
> 'Found IT,' the Mouse replied rather crossly: 'of course you know what "it" means.'
> 'I know what "it" means well enough, when I find a thing,' said the Duck: 'it's generally a frog or a worm. The question is, what did the archbishop find?
> Lewis Carroll, Alice in Wonderland

In order to interpret the sentences of any discourse, we need to know who or what entity is being talked about. Consider the following passage:

(21.34) Victoria Chen, Chief Financial Officer of Megabucks Banking Corp since 2004, saw her pay jump 20%, to $1.3 million, as the 37-year-old also became the Denver-based financial-services company's president. It has been ten years since she came to Megabucks from rival Lotsabucks.

In this passage, each of the underlined phrases is used by the speaker to denote one person named Victoria Chen. We refer to this use of linguistic expressions like *her* or *Victoria Chen* to denote an entity or individual as **reference**. In the next few sections of this chapter we study the problem of **reference resolution**. Reference resolution is the task of determining what entities are referred to by which linguistic expressions.

REFERENCE

REFERENCE RESOLUTION

We first define some terminology. A natural language expression used to perform reference is called a **referring expression**, and the entity that is referred to is called the **referent**. Thus, *Victoria Chen* and *she* in passage (21.34) are referring expressions, and Victoria Chen is their referent. (To distinguish between referring expressions and

REFERRING EXPRESSION

REFERENT

COREFER
ANTECEDENT
ANAPHORA
ANAPHORIC

their referents, we italicize the former.) As a convenient shorthand, we will sometimes speak of a referring expression referring to a referent, e.g., we might say that *she* refers to Victoria Chen. However, the reader should keep in mind that what we really mean is that the speaker is performing the act of referring to Victoria Chen by uttering *she*. Two referring expressions that are used to refer to the same entity are said to **corefer**; thus *Victoria Chen* and *she* corefer in passage (21.34). There is also a term for a referring expression that licenses the use of another, in the way that the mention of *John* allows John to be subsequently referred to using *he*. We call *John* the **antecedent** of *he*. Reference to an entity that has been previously introduced into the discourse is called **anaphora**, and the referring expression used is said to be **anaphoric**. In passage (21.34), the pronouns *she* and *her*, and the definite NP *the 37-year-old* are therefore anaphoric.

DISCOURSE CONTEXT
SITUATIONAL CONTEXT

Natural languages provide speakers with a variety of ways to refer to entities. Say that your friend has a 1961 Ford Falcon automobile and you want to refer to it. Depending on the operative **discourse context**, you might say *it, this, that, this car, that car, the car*, *the Ford*, *the Falcon*, or *my friend's car*, among many other possibilities. However, you are not free to choose between any of these alternatives in any context. For instance, you cannot simply say *it* or *the Falcon* if the hearer has no prior knowledge of your friend's car, it has not been mentioned before, and it is not in the immediate surroundings of the discourse participants (i.e., the **situational context** of the discourse).

DISCOURSE MODEL

The reason for this is that each type of referring expression encodes different signals about the place that the speaker believes the referent occupies within the hearer's set of beliefs. A subset of these beliefs that has a special status form the hearer's mental model of the ongoing discourse, which we call a **discourse model** (Webber, 1978). The discourse model contains representations of the entities that have been referred to in the discourse and the relationships in which they participate. Thus, there are two components required by a system to successfully interpret (or produce) referring expressions: a method for constructing a discourse model that evolves with the dynamically-changing discourse it represents, and a method for mapping between the signals that various referring expressions encode and the hearer's set of beliefs, the latter of which includes this discourse model.

EVOKED
ACCESSED

We will speak in terms of two fundamental operations to the discourse model. When a referent is first mentioned in a discourse, we say that a representation for it is **evoked** into the model. Upon subsequent mention, this representation is **accessed** from the model. The operations and relationships are illustrated in Figure 21.5. As we will see in Sec. 21.8, the discourse model plays an important role in how coreference algorithms are evaluated.

COREFERENCE RESOLUTION
COREFERENCE CHAIN

We are now ready to introduce two reference resolution tasks: **coreference resolution** and **pronominal anaphora resolution**. Coreference resolution is the task of finding referring expressions in a text that refer to the same entity, i.e. finding expressions that **corefer**. We call the set of coreferring expressions a **coreference chain**. For example, in processing (21.34), a coreference resolution algorithm would need to find four coreference chains:

1. { *Victoria Chen, Chief Financial Officer of Megabucks Banking Corp since 1994, her, the*

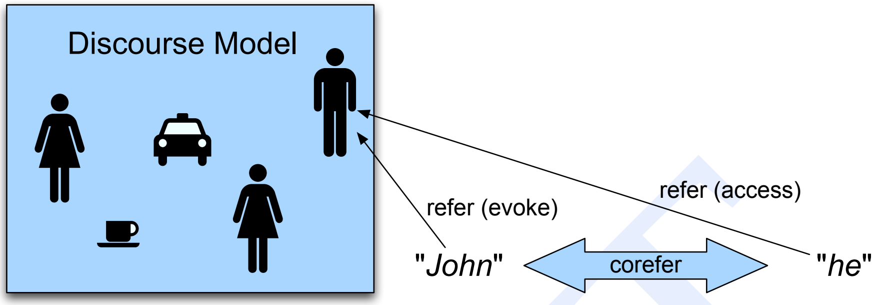


Figure 21.5 Reference operations and relationships with respect to the discourse model.

37-year-old, the Denver-based financial-services company's president, She}

2. { *Megabucks Banking Corp, the Denver-based financial-services company, Megabucks* }
3. { *her pay* }
4. { *Lotsabucks* }

Coreference resolution thus requires finding all referring expressions in a discourse, and grouping them into coreference chains. By contrast, **pronominal anaphora resolution** is the task of finding the antecedent for a single pronoun; for example, given the pronoun *her*, our task is to decide that the antecedent of *her* is *Victoria Chen*. Thus pronominal anaphora resolution can be viewed as a subtask of coreference resolution.[2]

PRONOMINAL ANAPHORA RESOLUTION

In the next section we introduce different kinds of reference phenomena. We then give various algorithms for reference resolution. Pronominal anaphora has received a lot of attention in speech and language processing, and so we will introduce three algorithms for pronoun processing: the **Hobbs** algorithm, a **Centering** algorithm, and a **log-linear** (MaxEnt) algorithm. We then give an algorithm for the more general coreference resolution task.

We will see that each of these algorithms focuses on resolving reference to entities or individuals. It is important to note, however, that discourses do include reference to many other types of referents than entities. Consider the possibilities in example (21.35), adapted from Webber (1991).

(21.35) According to Doug, Sue just bought a 1961 Ford Falcon.

a. But *that* turned out to be a lie.
b. But *that* was false.
c. *That* struck me as a funny way to describe the situation.
d. *That* caused a financial problem for Sue.

The referent of *that* is a speech act (see Ch. 24) in (21.35a), a proposition in (21.35b), a manner of description in (21.35c), and an event in (21.35d). The field awaits the development of robust methods for interpreting these types of reference.

[2] Although technically there are cases of anaphora that are not cases of coreference; see van Deemter and Kibble (2000) for more discussion.

21.4 Reference Phenomena

The set of referential phenomena that natural languages provide is quite rich indeed. In this section, we provide a brief description of several basic reference phenomena, surveying five types of referring expression: *indefinite noun phrases*, *definite noun phrases*, *pronouns*, *demonstratives*, and *names*. We then summarize the way these referring expressions are used to encode **given** and **new** information, along the way introducing two types of referents that complicate the reference resolution problem: *inferrables* and *generics*.

21.4.1 Five Types of Referring Expressions

Indefinite Noun Phrases Indefinite reference introduces entities that are new to the hearer into the discourse context. The most common form of indefinite reference is marked with the determiner *a* (or *an*), but it can also be marked by a quantifier such as *some* or even the determiner *this*:

(21.36) (a) Mrs. Martin was so very kind as to send Mrs. Goddard *a beautiful goose*.
(b) He had gone round one day to bring her *some walnuts*.
(c) I saw *this beautiful Ford Falcon* today.

Such noun phrases evoke a representation for a new entity that satisfies the given description into the discourse model.

The indefinite determiner *a* does not indicate whether the entity is identifiable to the speaker, which in some cases leads to a *specific*/*non-specific* ambiguity. Example (21.36a) only has the specific reading, since the speaker has a particular goose in mind, particularly the one Mrs. Martin sent. In sentence (21.37), on the other hand, both readings are possible.

(21.37) I am going to the butchers to buy a goose.

That is, the speaker may already have the goose picked out (specific), or may just be planning to pick one out that is to her liking (nonspecific).

Definite Noun Phrases Definite reference is used to refer to an entity that is identifiable to the hearer. An entity can be identifiable to the hearer because it has been mentioned previously in the text, and thus is already represented in the discourse model:

(21.38) It concerns a white stallion which I have sold to an officer. But the pedigree of *the white stallion* was not fully established.

Alternatively, an entity can be identifiable because it is contained in the hearer's set of beliefs about the world, or the uniqueness of the object is implied by the description itself, in which case it evokes a representation of the referent into the discourse model, as in (21.39):

(21.39) I read about it in *The New York Times*.

Pronouns Another form of definite reference is pronominalization, illustrated in example (21.40).

(21.40) Emma smiled and chatted as cheerfully as *she* could, SALIENCE

The constraints on using pronominal reference are stronger than for full definite noun phrases, requiring that the referent have a high degree of activation or **salience** in the discourse model. Pronouns usually (but not always) refer to entities that were introduced no further than one or two sentences back in the ongoing discourse, whereas definite noun phrases can often refer further back. This is illustrated by the difference between sentences (21.41d) and (21.41d').

(21.41)
a. John went to Bob's party, and parked next to a classic Ford Falcon.
b. He went inside and talked to Bob for more than an hour.
c. Bob told him that he recently got engaged.
d. ?? He also said that he bought *it* yesterday.
d.' He also said that he bought *the Falcon* yesterday.

By the time the last sentence is reached, the Falcon no longer has the degree of salience required to allow for pronominal reference to it.

CATAPHORA

Pronouns can also participate in **cataphora**, in which they are mentioned before their referents are, as in example (21.42).

(21.42) Even before *she* saw *it*, Dorothy had been thinking about the Emerald City every day.

Here, the pronouns *she* and *it* both occur *before* their referents are introduced.

BOUND

Pronouns also appear in quantified contexts in which they are considered to be **bound**, as in example (21.43).

(21.43) Every dancer brought *her* left arm forward.

Under the relevant reading, *her* does not refer to some woman in context, but instead behaves like a variable bound to the quantified expression *every dancer*. We will not be concerned with the bound interpretation of pronouns in this chapter.

PROXIMAL DEMONSTRATIVE
DISTAL DEMONSTRATIVE

Demonstratives Demonstrative pronouns, like *this* and *that*, behave somewhat differently than simple definite pronouns like *it*. They can appear either alone or as determiners, for instance, *this ingredient*, *that spice*. *This* and *that* differ in lexical meaning; (*this*, the **proximal demonstrative**, indicating literal or metaphorical closeness, while *that*, the **distal demonstrative** indicating literal or metaphorical distance (further away in time, as in the following example)):

(21.44) I just bought a copy of Thoreau's *Walden*. I had bought one five years ago. *That one* had been very tattered; *this one* was in much better condition.

Note that *this NP* is ambiguous; in colloquial spoken English, it can be indefinite, as in (21.36), or definite, as in (21.44).

Names Names are a very common form of referring expression, including names of people, organizations, and locations, as we saw in the discussion of named entities in Sec. **??**. Names can be used to refer to both new and old entities in the discourse:

(21.45)
a. **Miss Woodhouse** certainly had not done him justice.
b. **International Business Machines** sought patent compensation from Amazon; **I.B.M.** had previously sued other companies.

21.4.2 Information Status

We noted above that the same referring expressions (such as many indefinite NPs) can be used to introduce new referents, while other expressions (such as many definite NPs, or pronouns) can be used to refer anaphorically to old referents. This idea of studying the way different referential forms are used to provide new or old information is called **information status** or **information structure**.

INFORMATION STATUS
INFORMATION STRUCTURE

GIVENNESS HIERARCHY

There are a variety of theories that express the relation between different types of referential form and the informativity or saliency of the referent in the discourse. For example, the **givenness hierarchy** (Gundel et al., 1993) is a scale representing six kinds of information status that different referring expression are used to signal:

The givenness hierarchy:

in focus >	activated >	familiar >	uniquely identifiable >	referential >	type identifiable
{it}	{ *that* *this* *this* N }	{that N}	{the N}	{indef. *this* N}	{*a* N}

ACCESSIBILITY SCALE

The related **accessibility scale** of Ariel (2001) is based on the idea that referents that are more salient will be easier for the hearer to call to mind, and hence can be referred to with less linguistic material. By contrast, less salient entities will need a longer and more explicit referring expression to help the hearer recover the referent. The following shows a sample scale going from low to high accessibility:

> *Full name* > *long definite description* > *short definite description* > *last name* > *first name* > *distal demonstrative* > *proximate demonstrative* > *NP* > *stressed pronoun* > *unstressed pronoun*

Note that accessibility correlates with length, with less accessible NPs tending to be longer. Indeed, if we follow a coreference chain in a discourse, we will often find longer NPs (for example long definition descriptions with relative clauses) early in the discourse, and much shorter ones (for example pronouns) later in the discourse.

Another perspective, based on the work of (Prince, 1992), is to analyze information status in terms of two crosscutting dichotomies: *hearer status* and *discourse status*. The *hearer status* of a referent expresses whether it is previously known to the hearer, or whether it is new. The *discourse status* expresses whether the referent has been previously mentioned in the discourse.

The relationship between referring expression form and information status can be complicated; we summarize below three such complicating factors (the use of **inferrables**, **generics**, and **non-referential forms**):

INFERRABLES
BRIDGING INFERENCES
MEDIATED

Inferrables: In some cases, a referring expression does not refer to an entity that has been explicitly evoked in the text, but instead one that is inferentially related to an evoked entity. Such referents are called **inferrables**, **bridging inferences**, or **mediated**(Haviland and Clark, 1974; Prince, 1981; Nissim et al., 2004) Consider the expressions *a door* and *the engine* in sentence (21.46).

(21.46) I almost bought a 1961 Ford Falcon today, but *a door* had a dent and *the engine* seemed noisy.

The indefinite noun phrase *a door* would normally introduce a new door into the discourse context, but in this case the hearer is to infer something more: that it is not just any door, but one of the doors of the Falcon. Similarly, the use of the definite noun phrase *the engine* normally presumes that an engine has been previously evoked or is otherwise uniquely identifiable. Here, no engine has been explicitly mentioned, but the hearer makes a **bridging inference** to infer that the referent is the engine of the previously mentioned Falcon.

Generics: Another kind of expression that does not refer back to an entity explicitly evoked in the text is *generic* reference. Consider example (21.47).

(21.47) I'm interested in buying a Mac laptop. *They* are very stylish.

Here, *they* refers, not to a particular laptop (or even a particular set of laptops), but instead to the class of Mac laptops in general. Similarly, the pronoun *you* can be used generically in the following example:

(21.48) In March in Boulder *you* have to wear a jacket.

Non-referential uses: Finally, some non-referential forms bear a confusing superficial resemblance to referring expressions. For example in addition to its referring usages, the word *it* can be used in **pleonastic** cases like *it is raining*, in idioms like *hit it off*, or in particular syntactic situations like **clefts** (21.49a) or **extraposition** (21.49b):

PLEONASTIC

CLEFTS

(21.49) (a) *It* was Frodo who carried the ring.
(b) *It* was good that Frodo carried the ring.

21.5 Features for Pronominal Anaphora Resolution

We now turn to the task of resolving pronominal reference. In general, this problem is formulated as follows. We are given a single pronoun (*he*, *him*, *she*, *her*, *it*, and sometimes *they/them*), together with the previous context. Our task is to find the antecedent of the pronoun in this context. We present three systems for this task; but first we summarize useful constraints on possible referents.

We begin with five relatively hard-and-fast morphosyntactic features that can be used to filter the set of possible referents: **number**, **person**, **gender**, and **binding theory** constraints.

Number Agreement: Referring expressions and their referents must agree in number; for English, this means distinguishing between *singular* and *plural* references. English *she/her/he/him/his/it* are singular, *we/us/they/them* are plural, and *you* is unspecified for number. Some illustrations of the constraints on number agreement:

John has a Ford Falcon. It is red.	* John has a Ford Falcon. They are red.
John has three Ford Falcons. They are red.	* John has three Ford Falcons. It is red.

We cannot always enforce a very strict grammatical notion of number agreement, since sometimes semantically plural entities can be referred to by either *it* or *they*:

(21.50) IBM announced a new machine translation product yesterday. *They* have been working on it for 20 years.

Person Agreement: English distinguishes between three forms of person: first, second, and third. The antecedent of a pronoun must agree with the pronoun in number. A first person pronoun (*I*, *me*, *my*) must have a first person antecedent (*I*, *me*, or *my*). A second person pronoun (*you* or *your*) must have a second person antecedent (*you* or *your*). A third person pronoun (*he*, *she*, *they*, *him*, *her*, *them*, *his*, *her*, *their*) must have a third person antecedent (one of the above or any other noun phrase).

Gender Agreement: Referents also must agree with the gender specified by the referring expression. English third person pronouns distinguish between *male*, (*he*, *him*, *his*), *female*, (*she*, *her*) and *nonpersonal* (*it*) genders. Unlike in some languages, English male and female pronoun genders only apply to animate entities; inanimate entities are always nonpersonal/neuter. Some examples:

(21.51) John has a Ford. He is attractive. (he=John, not the Ford)

(21.52) John has a Ford. It is attractive. (it=the Ford, not John)

Binding Theory Constraints: Reference relations may also be constrained by the syntactic relationships between a referential expression and a possible antecedent noun phrase when both occur in the same sentence. For instance, the pronouns in all of the following sentences are subject to the constraints indicated in brackets.

(21.53) John bought himself a new Ford. [himself=John]

(21.54) John bought him a new Ford. [him≠John]

(21.55) John said that Bill bought him a new Ford. [him≠Bill]

(21.56) John said that Bill bought himself a new Ford. [himself=Bill]

(21.57) He said that he bought John a new Ford. [He≠John; he≠John]

REFLEXIVES

English pronouns such as *himself*, *herself*, and *themselves* are called **reflexives**. Oversimplifying the situation, a reflexive corefers with the subject of the most immediate clause that contains it (ex. 21.53), whereas a nonreflexive cannot corefer with this subject (ex. 21.54). That this rule applies only for the subject of the most immediate clause is shown by examples (21.55) and (21.56), in which the opposite reference pattern is manifest between the pronoun and the subject of the higher sentence. On the other hand, a full noun phrase like *John* cannot corefer with the subject of the most immediate clause nor with a higher-level subject (ex. 21.57).

BINDING THEORY

These constraints are often called the **binding theory** (Chomsky, 1981), and quite complicated versions of these constraints have been proposed. A complete statement of the constraints requires reference to semantic and other factors, and cannot be stated purely in terms of syntactic configuration. Nonetheless, for the algorithms discussed later in this chapter we will assume a simple syntactic account of restrictions on intrasentential coreference.

Selectional Restrictions: The selectional restrictions that a verb places on its arguments (see Ch. 19) may be responsible for eliminating referents, as in example (21.58).

(21.58) John parked his car in the garage after driving it around for hours.

There are two possible referents for *it*, the car and the garage. The verb *drive*, however, requires that its direct object denote something that can be driven, such as a car, truck, or bus, but not a garage. Thus, the fact that the pronoun appears as the object of *drive* restricts the set of possible referents to the car. Selectional restrictions can be implemented by storing a dictionary of probabilistic dependencies between the verb associated with the pronoun and the potential referent.

Recency: We next turn to features for predicting the referent of a pronoun that are less hard-and-fast. Entities introduced in recent utterances tend to be more salient than those introduced from utterances further back. Thus, in example (21.59), the pronoun *it* is more likely to refer to Jim's map than the doctor's map.

(21.59) The doctor found an old map in the captain's chest. Jim found an even older map hidden on the shelf. It described an island.

Grammatical Role: Many theories specify a salience hierarchy of entities that is ordered by the grammatical position of the expressions which denote them. These typically treat entities mentioned in subject position as more salient than those in object position, which are in turn more salient than those mentioned in subsequent positions.

Passages such as (21.60) and (21.61) lend support for such a hierarchy. Although the first sentence in each case expresses roughly the same propositional content, the preferred referent for the pronoun *he* varies with the subject in each case – John in (21.60) and Bill in (21.61).

(21.60) Billy Bones went to the bar with Jim Hawkins. He called for a glass of rum.
[he = Billy]

(21.61) Jim Hawkins went to the bar with Billy Bones. He called for a glass of rum.
[he = Jim]

Repeated Mention: Some theories incorporate the idea that entities that have been focused on in the prior discourse are more likely to continue to be focused on in subsequent discourse, and hence references to them are more likely to be pronominalized. For instance, whereas the pronoun in example (21.61) has Jim as its preferred interpretation, the pronoun in the final sentence of example (21.62) may be more likely to refer to Billy Bones.

(21.62) Billy Bones had been thinking about a glass of rum ever since the pirate ship docked. He hobbled over to the Old Parrot bar. Jim Hawkins went with him. He called for a glass of rum. [he = Billy]

Parallelism: There are also strong preferences that appear to be induced by parallelism effects, as in example (21.63).

(21.63) Long John Silver went with Jim to the Old Parrot. Billy Bones went with him to the Old Anchor Inn. [him = Jim]

The grammatical role hierarchy described above ranks Long John Silver as more salient than Jim, and thus should be the preferred referent of *him*. Furthermore, there is no semantic reason that Long John Silver cannot be the referent. Nonetheless, *him* is instead understood to refer to Jim.

Verb Semantics Certain verbs appear to place a semantically-oriented emphasis on one of their argument positions, which can have the effect of biasing the manner in which subsequent pronouns are interpreted. Compare sentences (21.64) and (21.65).

(21.64) John telephoned Bill. He lost the laptop.

(21.65) John criticized Bill. He lost the laptop.

These examples differ only in the verb used in the first sentence, yet the subject pronoun in passage (21.64) is typically resolved to John, whereas the pronoun in passage (21.65) is resolved to Bill. It has been argued that this effect results from what the "implicit causality" of a verb: the implicit cause of a "criticizing" event is considered to be its object, whereas the implicit cause of a "telephoning" event is considered to be its subject. This emphasis results in a higher degree of salience for the entity in this argument position.

21.6 Three algorithms for pronominal anaphora resolution

21.6.1 Pronominal Anaphora Baseline: The Hobbs Algorithm

HOBBS ALGORITHM

The first of the three algorithms we present for pronominal anaphora resolution is the **Hobbs algorithm**. The Hobbs algorithm (the simpler of two algorithms presented originally in Hobbs (1978)) depends only on a syntactic parser plus a morphological gender and number checker. For this reason it is often used as a baseline when evaluating new pronominal anaphora resolution algorithms.

The input to the Hobbs algorithm is a pronoun to be resolved, together with a syntactic parse of the sentences up to and including the current sentence. The algorithm searches for an antecedent noun phrase in these trees. The intuition of the algorithm is to start with the target pronoun and walk up the parse tree to the root S. For each NP or S node that it finds, it does a breadth-first left-to-right search of the node's children to the left of the target. As each candidate noun phrase is proposed, it is checked for gender, number, and person agreement with the pronoun. If no referent is found, the algorithm performs the same breadth-first search on preceding sentences.

The Hobbs algorithm does not capture all the constraints and preferences on pronominalization described above. It does, however, approximate the *binding theory*, *recency*, and *grammatical role* preferences by the order in which the search is performed, and the *gender*, *person*, and *number* constraints by a final check.

An algorithm that searches parse trees must also specify a grammar, since the assumptions regarding the structure of syntactic trees will affect the results. A fragment for English that the algorithm uses is given in Figure 21.6. The steps of the **Hobbs algorithm** are as follows:

1. Begin at the noun phrase (NP) node immediately dominating the pronoun.
2. Go up the tree to the first NP or sentence (S) node encountered. Call this node X, and call the path used to reach it p.

$$
\begin{array}{l}
S \;\rightarrow\; NP\ VP \\
NP \;\rightarrow\; \left\{ \begin{array}{l} (Det)\quad Nominal \quad \left(\left\{ \begin{array}{l} PP \\ Rel \end{array} \right\} \right)^{*} \\ pronoun \end{array} \right\} \\
Det \;\rightarrow\; \left\{ \begin{array}{l} determiner \\ NP\text{ 's} \end{array} \right\} \\
PP \;\rightarrow\; preposition\ NP \\
Nominal \;\rightarrow\; noun\ (PP)^{*} \\
Rel \;\rightarrow\; wh\text{-}word\ S \\
VP \;\rightarrow\; verb\ NP\ (PP)^{*}
\end{array}
$$

Figure 21.6 A grammar fragment for the Tree Search algorithm.

3. Traverse all branches below node X to the left of path p in a left-to-right, breadth-first fashion. Propose as the antecedent any NP node that is encountered which has an NP or S node between it and X.
4. If node X is the highest S node in the sentence, traverse the surface parse trees of previous sentences in the text in order of recency, the most recent first; each tree is traversed in a left-to-right, breadth-first manner, and when an NP node is encountered, it is proposed as antecedent. If X is not the highest S node in the sentence, continue to step 5.
5. From node X, go up the tree to the first NP or S node encountered. Call this new node X, and call the path traversed to reach it p.
6. If X is an NP node and if the path p to X did not pass through the Nominal node that X immediately dominates, propose X as the antecedent.
7. Traverse all branches below node X to the *left* of path p in a left-to-right, breadth-first manner. Propose any NP node encountered as the antecedent.
8. If X is an S node, traverse all branches of node X to the *right* of path p in a left-to-right, breadth-first manner, but do not go below any NP or S node encountered. Propose any NP node encountered as the antecedent.
9. Go to Step 4.

Demonstrating that this algorithm yields the correct coreference assignments for an example sentence is left as Exercise 21.2.

Most parsers return number information (singular or plural), and person information is easily encoded by rule for the first and second person pronouns. But parsers for English rarely return gender information for common or proper nouns. Thus the only additional requirement to implementing the Hobbs algorithm, besides a parser, is an algorithm for determining gender for each antecedent noun phrase.

One common way to assign gender to a noun phrase is to extract the head noun, and then use WordNet (Ch. 19) to look at the hypernyns of the head noun. Ancestors like *person* or *living thing* indicate an animate noun. Ancestors like *female* indicate a female noun. A list of personal names associated with genders, or patterns like `Mr.` can also be used (Cardie and Wagstaff, 1999).

More complex algorithms exist, such as that of Bergsma and Lin (2006); Bergsma and Lin also make freely available a large list of nouns and their (automatically extracted) genders.

21.6.2 A Centering Algorithm for Anaphora Resolution

CENTERING THEORY

The Hobbs algorithm does not use an explicit representation of a discourse model. By contrast **Centering theory**, (Grosz et al., 1995b, henceforth GJW) is a family of models which has an explicit representation of a discourse model, and incorporates an additional claim: that there is a single entity being "centered" on at any given point in the discourse which is to be distinguished from all other entities that have been evoked. Centering theory has been applied to many problems in discourse, such as the computation of **entity-based coherence**; in this section we see its application to anaphora resolution.

BACKWARD LOOKING CENTER

FORWARD LOOKING CENTERS

There are two main representations tracked in the Centering theory discourse model. In what follows, take U_n and U_{n+1} to be two adjacent utterances. The **backward looking center** of U_n, denoted as $C_b(U_n)$, represents the entity currently being focused on in the discourse after U_n is interpreted. The **forward looking centers** of U_n, denoted as $C_f(U_n)$, form an ordered list containing the entities mentioned in U_n, all of which could serve as the C_b of the following utterance. In fact, $C_b(U_{n+1})$ is by definition the most highly ranked element of $C_f(U_n)$ mentioned in U_{n+1}. (The C$_b$ of the first utterance in a discourse is undefined.) As for how the entities in the $C_f(U_n)$ are ordered, for simplicity's sake we can use the grammatical role hierarchy below.[3]

> subject $>$ existential predicate nominal $>$ object $>$ indirect object or oblique
> $>$ demarcated adverbial PP

As a shorthand, we will call the highest-ranked forward-looking center C_p (for "preferred center").

We describe a centering-based algorithm for pronoun interpretation due to Brennan et al. (1987, henceforth BFP). (See also Walker et al. (1994) and the end of the chapter for other centering algorithms). In this algorithm, preferred referents of pronouns are computed from relations that hold between the forward and backward looking centers in adjacent sentences. Four intersentential relationships between a pair of utterances U_n and U_{n+1} are defined which depend on the relationship between $C_b(U_{n+1})$, $C_b(U_n)$, and $C_p(U_{n+1})$; these are shown in Figure 21.7.

	$C_b(U_{n+1}) = C_b(U_n)$ or undefined $C_b(U_n)$	$C_b(U_{n+1}) \neq C_b(U_n)$
$C_b(U_{n+1}) = C_p(U_{n+1})$	Continue	Smooth-Shift
$C_b(U_{n+1}) \neq C_p(U_{n+1})$	Retain	Rough-Shift

Figure 21.7 Transitions in the BFP algorithm.

The following rules are used by the algorithm:

[3] This is an extended form of the hierarchy used in Brennan et al. (1987), described below.

- Rule 1: If any element of $C_f(U_n)$ is realized by a pronoun in utterance U_{n+1}, then $C_b(U_{n+1})$ must be realized as a pronoun also.
- Rule 2: Transition states are ordered. Continue is preferred to Retain is preferred to Smooth-Shift is preferred to Rough-Shift.

Having defined these concepts and rules, the algorithm is defined as follows.

1. Generate possible C_b-C_f combinations for each possible set of reference assignments .
2. Filter by constraints, e.g., syntactic coreference constraints, selectional restrictions, centering rules and constraints.
3. Rank by transition orderings.

The pronominal referents that get assigned are those which yield the most preferred relation in Rule 2, assuming that Rule 1 and other coreference constraints (gender, number, syntactic, selectional restrictions) are not violated.

Let us step through passage (21.66) to illustrate the algorithm.

(21.66) John saw a beautiful 1961 Ford Falcon at the used car dealership. (U_1)
He showed it to Bob. (U_2)
He bought it. (U_3)

Using the grammatical role hierarchy to order the C$_f$, for sentence U_1 we get:

$C_f(U_1)$: {John, Ford, dealership}
$C_p(U_1)$: John
$C_b(U_1)$: undefined

Sentence U_2 contains two pronouns: *he*, which is compatible with John, and *it*, which is compatible with the Ford or the dealership. John is by definition $C_b(U_2)$, because he is the highest ranked member of $C_f(U_1)$ mentioned in U_2 (since he is the only possible referent for *he*). We compare the resulting transitions for each possible referent of *it*. If we assume *it* refers to the Falcon, the assignments would be:

$C_f(U_2)$: {John, Ford, Bob}
$C_p(U_2)$: John
$C_b(U_2)$: John
Result: Continue ($C_p(U_2)$=$C_b(U_2)$; $C_b(U_1)$ undefined)

If we assume *it* refers to the dealership, the assignments would be:

$C_f(U_2)$: {John, dealership, Bob}
$C_p(U_2)$: John
$C_b(U_2)$: John
Result: Continue ($C_p(U_2)$=$C_b(U_2)$; $C_b(U_1)$ undefined)

Since both possibilities result in a Continue transition, the algorithm does not say which to accept. For the sake of illustration, we will assume that ties are broken in terms of the ordering on the previous C_f list. Thus, we will take *it* to refer to the Falcon instead of the dealership, leaving the current discourse model as represented in the first possibility above.

In sentence U_3, *he* is compatible with either John or Bob, whereas *it* is compatible with the Ford. If we assume *he* refers to John, then John is $C_b(U_3)$ and the assignments would be:

$C_f(U_3)$: {John, Ford}
$C_p(U_3)$: John
$C_b(U_3)$: John
Result: Continue $(C_p(U_3)$=$C_b(U_3)$=$C_b(U_2))$

If we assume *he* refers to Bob, then Bob is $C_b(U_3)$ and the assignments would be:

$C_f(U_3)$: {Bob, Ford}
$C_p(U_3)$: Bob
$C_b(U_3)$: Bob
Result: Smooth-Shift $(C_p(U_3)$=$C_b(U_3)$; $C_b(U_3)$≠$C_b(U_2))$

Since a Continue is preferred to a Smooth-Shift per Rule 2, John is correctly taken to be the referent.

The main salience factors that the centering algorithm implicitly incorporates include the grammatical role, recency, and repeated mention preferences. The manner in which the grammatical role hierarchy affects salience is indirect, since it is the resulting transition type that determines the final reference assignments. In particular, a referent in a low-ranked grammatical role will be preferred to one in a more highly ranked role if the former leads to a more highly ranked transition. Thus, the centering algorithm may incorrectly resolve a pronoun to a low salience referent. For instance, in example (21.67),

(21.67) Bob opened up a new dealership last week. John took a look at the Fords in his lot. He ended up buying one.

the centering algorithm will assign Bob as the referent of the subject pronoun *he* in the third sentence – since Bob is $C_b(U_2)$, this assignment results in a Continue relation whereas assigning John results in a Smooth-Shift relation. On the other hand, the Hobbs algorithm will correctly assign John as the referent.

Like the Hobbs algorithm, the centering algorithm requires a full syntactic parse as well as morphological detectors for gender.

Centering theory is also a model of entity coherence, and hence has implications for other discourse applications like summarization; see the end of the chapter for pointers.

21.6.3 A Log-Linear model for Pronominal Anaphora Resoluton

As our final model of pronominal anaphora resolution, we present a simple supervised machine learning approach, in which we train a log-linear classifier on a corpus in which the antecedents are marked for each pronoun. Any supervised classifier can be used for this purpose; log-linear models are popular, but Naive Bayes and other classifiers have been used as well.

For training, the system relies on a hand-labeled corpus in which each pronoun has been linked by hand with the correct antecedent. The system needs to extract positive and negative examples of anaphoric relations. Positive examples occur directly in the training set. Negative examples can be found by pairing each pronoun with some other noun phrase. Features (discussed in the next section) are extracted for each training observation, and a classifier is trained to predict *1* for the true pronoun-antecedent pairs, and *0* for the incorrect pronoun-antecedent pairs.

For testing, just as we saw with as with the Hobbs and Centering classifiers, the log-linear classifier takes as input a pronoun (*he, him, his, she, her, it, they, them, their*), together with the current and preceding sentences.

In order to deal with non-referential pronouns, we first filter out pleonastic pronouns (like the pleonastic *it is raining*), using hand-written rules based on frequent lexical patterns.

The classifier then extracts all potential antecedents by doing a parse of the current and previous sentences, either using a full parser or a simple chunker. Next, each NP in the parse is considered a potential antecedent for each following pronoun. Each pronoun-potential antecedent pair is then presented to the classifier.

21.6.4 Features

Some commonly used features for pronominal anaphora resolution between a pronoun *Pro*$_i$ and a potential referent *NP*$_j$ include:

1. **strict gender [true** or **false]**. True if there is a strict match in gender (e.g. male pronoun Pro_i with male antecedent NP_j).
2. **compatible gender [true** or **false]**. True if Pro_i and NP_j are merely compatible (e.g. male pronoun Pro_i with antecedent NP_j of unknown gender).
3. **strict number [true** or **false]** True if there is a strict match in number (e.g. singular pronoun with singular antecedent)
4. **compatible number [true** or **false]**. True if Pro_i and NP_j are merely compatible (e.g. singular pronoun Pro_i with antecedent NP_j of unknown number).
5. **sentence distance [0, 1, 2, 3,...]**. The number of sentences between pronoun and potential antecedent.
6. **Hobbs distance [0, 1, 2, 3,...]**. The number of noun groups that the Hobbs algorithm has to skip, starting backwards from the pronoun Pro_i, before the potential antecedent NP_j is found.
7. **grammatical role [subject, object, PP]**. Whether the potential antecedent is a syntactic subject, direct object, or is embedded in a PP.
8. **linguistic form [proper, definite, indefinite, pronoun]**. Whether the potential antecedent NP_j is a proper name, definite description, indefinite NP, or a pronoun.

Fig. 21.8 shows some sample feature values for potential antecedents for the final *He* in U_3:

(21.68) John saw a beautiful 1961 Ford Falcon at the used car dealership. (U_1)
He showed it to Bob. (U_2)
He bought it. (U_3)

The classifier will learn weights indicating which of these features are more likely to be good predictors of a successful antecedent (e.g. being nearby the pronoun, in

	He (U_2)	it (U_2)	Bob (U_2)	John (U_1)
strict number	1	1	1	1
compatible number	1	1	1	1
strict gender	1	0	1	1
compatible gender	1	0	1	1
sentence distance	1	1	1	2
Hobbs distance	2	1	0	3
grammatical role	subject	object	PP	subject
linguistic form	pronoun	pronoun	proper	proper

Figure 21.8 Feature values in log-linear classifier, for various pronouns from (21.68).

subject position, agreeing in gender and number). Thus where the Hobbs and Centering algorithms rely on hand-built heuristics for antecedent selection, the machine learning classifiers learn the importance of these different features based on their co-occurrence in the training set.

21.7 Coreference Resolution

In the previous few sections, we concentrated on interpreting a particular subclass of the reference phenomena that we outlined in Sec. 21.4: the personal pronouns such as *he*, *she*, and *it*. But for the general coreference task we'll need to decide whether any pair of noun phrases corefer. This means we'll need to deal with the other types of referring expressions from Sec. 21.4, the most common of which are *definite noun phrases* and *names*. Let's return to our coreference example, repeated below:

(21.69) Victoria Chen, Chief Financial Officer of Megabucks Banking Corp since 2004, saw her pay jump 20%, to $1.3 million, as the 37-year-old also became the Denver-based financial-services company's president. It has been ten years since she came to Megabucks from rival Lotsabucks.

Recall that we need to extract four coreference chains from this data:

1. { *Victoria Chen, Chief Financial Officer of Megabucks Banking Corp since 1994, her, the 37-year-old, the Denver-based financial-services company's president, She*}
2. { *Megabucks Banking Corp, the Denver-based financial-services company, Megabucks* }
3. { *her pay* }
4. { *Lotsabucks* }

As before, we have to deal with pronominal anaphora (figuring out that *her* refers to *Victoria Chen*). And we still need to filter out non-referential pronouns like the pleonastic *It* in *It has been ten years*), as we did for pronominal anaphora.

But for full NP coreference we'll also need to deal with definite noun phrases, to figure out that *the 37-year-old* is coreferent with *Victoria Chen*, and *the Denver-based financial-services company* is the same as *Megabucks*. And we'll need to deal with names, to realize that *Megabucks* is the same as *Megabucks Banking Corp*.

An algorithm for coreference resolution can use the same log-linear classifier architecture we saw for pronominal anaphora. Thus we'll build a binary classifier which is given an anaphor and a potential antecedent and returns true (the two are coreferential) or false (the two are not coreferential). We'll use this classifier in the resolution algorithm as follows. We process a document from left to right. For each NP_j we encounter, we'll search backwards through the document examining each previous NP. For each such potential antecedent NP_i, we'll run our classifier, and if it returns true, we successfully coindex NP_i and NP_j. The process for each NP_j terminates when we either find a successful antecedent NP_i or reach the beginning of the document. We then move on to the next anaphor NP_j.

In order to train our binary coreference classifier, just as for pronoun resolution, we'll need a labeled training set in which each anaphor NP_i has been linked by hand with the correct antecedent. In order to build a classifier, we'll need both positive and negative training examples of coreference relations. A positive examples for NP_i is the noun phrase NP_j which is marked as coindexed. We get negative examples by pairing the anaphor NP_j with the intervening NPs NP_{i+1}, NP_{i+2} which occur between the true antecedent NP_i and the anaphor NP_j.

Next features are extracted for each training observation, and a classifier is trained to predict whether an (NP_j,NP_i) pair corefer or not. Which features should we use in the binary coreference classifier? We can use all the features we used for anaphora resolution; number, gender, syntactic position, and so on. But we will also need to add new features to deal with phenomena that are specific to names and definite noun phrases. For example, we'll want a feature representing the fact that *Megabucks* and *Megabucks Banking Corp* share the word *Megabucks*, or that *Megabucks Banking Corp* and *the Denver-based financial-services company* both end in words (*Corp* and *company*) indicating a corporate organization.

Here are some commonly used features for coreference between an anaphor *NP*$_i$ and a potential antecedent *NP*$_j$ (in addition to the features for pronominal anaphora resolution listed on page 29):

1. **anaphor edit distance [0,1,2,...,]**. The character **minimum edit distance** from the potential antecedent to the anaphor. Recall from Ch. 3 that the character minimum edit distance is the minimum number of character editing operations (insertions, substitutions, deletions) necessary to turn one string into another. More formally,
$$100 \times \frac{m-(s+i+d)}{m}$$
given the antecedent length m, and the number of substitutions s, insertions i, and deletions d.
2. **antecedent edit distance [0,1,2,...,]**. The **minimum edit distance** from the anaphor to the antecedent. Given the anaphor length n:
$$100 \times \frac{n-(s+i+d)}{n}$$
3. **alias [true or false]**: A multi-part feature proposed by Soon et al. (2001) which requires a **named entity tagger**. Returns true if NP_i and NP_j are both named

entities of the same type, and NP_i is an **alias** of NP_j. The meaning of **alias** depends on the types; two dates are aliases of each other if they refer to the same date. For type PERSON, prefixes like *Dr.* or *Chairman* are stripped off and then the NPs are checked to see if they are identical. For type ORGANIZATION, the alias function checks for acronyms (e.g., *IBM* for *International Business Machines Corp*).

4. **appositive [true** or **false]**: True if the anaphor is in the syntactic apposition relation to the antecedent. For example the NP *Chief Financial Officer of Megabucks Banking Corp* is in apposition to the NP *Victoria Chen*. These can be detected using a parser, or more shallowly by looking for commas and requiring that neither NP have a verb and one of them be a name.

5. **linguistic form [proper, definite, indefinite, pronoun]**. Whether the potential anaphor NP_j is a proper name, definite description, indefinite NP or a pronoun.

21.8 Evaluating Coreference Resolution

One standard way of evaluating coreference is the Model-Theoretic coreference scoring scheme (Vilain et al., 1995), originally proposed for the MUC-6 and MUC-7 information extraction evaluation (Sundheim, 1995).

The evaluation is based on a human-labeled gold standard for coreference between referring expressions. We can represent this gold information as a set of identity links between referring expressions. For example, the fact that referring expression A and referring expression B are coreferent could be represented as a link A-B. If A, B, and C are coreferent, this can be represented as the two links A-B, B-C (or alternatively as A-C, B-C). We can call this set of correct links the **reference** or **key** set of links. Similarly, the **hypothesis** or **response** from a coreference algorithm can be viewed as a set of links.

What we'd like to do is compute the precision and recall of the **response** links against the **key** links. But recall that if entities A, B, and C are coreferent in the key, this can be represented either via (A-B, B-C) or via (A-C, B-C). As long as our coreference system correctly figures out that A, B, and C are coreferent, we don't want to penalize it for representing this fact in a different set of links than happen to be in the key.

For example, suppose that A, B, C, and D are coreferent, and this happens to be represented in the key by links (A-B, B-C, C-D). Suppose further that a particular coreference algorithm returns (A-B, C-D). What score should be given to this response? Intuitively the precision should be 1 (since both links correctly join referring expressions that indeed corefer). The recall should be 2/3, since intuitively it takes three links to correctly indicate that 4 expressions are coreferent, and the algorithm returned two of these three links. The details of this intuition are fleshed out in the Vilain et al. (1995) algorithm, which is based on computing the number of equivalence classes of expressions generated by the key.

21.9 ADVANCED: INFERENCE-BASED COHERENCE RESOLUTION

The algorithms we have seen in this chapter for the resolution of coherence and coreference have relied solely on shallow information like cue phrases and other lexical and simple syntactic cues. But many problems in resolution seem to require much more sophisticated kinds of knowledge. Consider the following example of coreference, adapted from Winograd (1972):

(21.70) The city council denied the demonstrators a permit because

a. they feared violence.

b. they advocated violence.

Determining the correct antecedent for the pronoun *they* requires understanding first that the second clause is intended as an **Explanation** of the first clause, and also that city councils are perhaps more likely than demonstrators to fear violence, and demonstrators might be more likely to advocate violence. A more advanced method for coherence resolution might assign this Explanation relation and in doing so help us figure out the referents of both pronouns.

We might perform this kind of more sophisticated coherence resolution by relying on the semantic constraints that are associated with each coherence relation, assuming a parser that could assign a reasonable semantics to each clause.

DEDUCTION

Applying these constraints requires a method for performing inference. Perhaps the most familiar type of inference is **deduction**; recall from Sec. **??** that the central rule of deduction is modus ponens:

$$\begin{array}{c} \alpha \Rightarrow \beta \\ \alpha \\ \hline \beta \end{array}$$

An example of modus ponens is the following:

$$\begin{array}{c} \text{All Falcons are fast.} \\ \text{John's car is an Falcon.} \\ \hline \text{John's car is fast.} \end{array}$$

SOUND INFERENCE

Deduction is a form of **sound inference**: if the premises are true, then the conclusion must be true.

ABDUCTION

However, much of language understanding is based on inferences that are not sound. While the ability to draw unsound inferences allows for a greater range of inferences to be made, it can also lead to false interpretations and misunderstandings. A method for such inference is logical **abduction** (Peirce, 1955). The central rule of abductive inference is:

$$\begin{array}{c} \alpha \Rightarrow \beta \\ \beta \\ \hline \alpha \end{array}$$

Whereas deduction runs an implication relation forward, abduction runs it backward, reasoning from an effect to a potential cause. An example of abduction is the following:

All Falcons are fast.
John’s car is fast.
John’s car is an Falcon.

Obviously, this may be an incorrect inference: John’s car may be made by another manufacturer yet still be fast.

In general, a given effect β may have many potential causes α_i. We generally will not want to merely reason from a fact to a *possible* explanation of it, we want to identify the *best* explanation of it. To do this, we need a method for comparing the quality of alternative abductive proofs. This can be done with probabilistic models (Charniak and Goldman, 1988; Charniak and Shimony, 1990), or with heuristic strategies (Charniak and McDermott, 1985, Chapter 10), such as preferring the explanation with the smallest number of assumptions, or the most specific explanation. We will illustrate a third approach to abductive interpretation, due to Hobbs et al. (1993), which applies a more general cost-based strategy that combines features of the probabilistic and heuristic approaches. To simplify the discussion, however, we will largely ignore the cost component of the system, keeping in mind that one is nonetheless necessary.

Hobbs et al. (1993) apply their method to a broad range of problems in language interpretation; here we focus on its use in establishing discourse coherence, in which world and domain knowledge are used to determine the most plausible coherence relation holding between utterances. Let us step through the analysis that leads to establishing the coherence of passage (21.4). First, we need axioms about coherence relations themselves. Axiom (21.71) states that a possible coherence relation is the Explanation relation; other relations would have analogous axioms.

(21.71)

$$\forall e_i, e_j \; Explanation(e_i, e_j) \;\Rightarrow\; CoherenceRel(e_i, e_j)$$

The variables e_i and e_j represent the events (or states) denoted by the two utterances being related. In this axiom and those given below, quantifiers always scope over everything to their right. This axiom tells us that, given that we need to establish a coherence relation between two events, one possibility is to abductively assume that the relation is Explanation.

The Explanation relation requires that the second utterance express the cause of the effect that the first sentence expresses. We can state this as axiom (21.72).

(21.72)

$$\forall e_i, e_j \; cause(e_j, e_i) \;\Rightarrow\; Explanation(e_i, e_j)$$

In addition to axioms about coherence relations, we also need axioms representing general knowledge about the world. The first axiom we use says that if someone is drunk, then others will not want that person to drive, and that the former causes the latter (for convenience, the state of not wanting is denoted by the *diswant* predicate).

(21.73)

$$\begin{aligned}&\forall x, y, e_i\ drunk(e_i, x) \ \Rightarrow\\ &\qquad \exists e_j, e_k\ diswant(e_j, y, e_k) \wedge drive(e_k, x) \wedge cause(e_i, e_j)\end{aligned}$$

Before we move on, a few notes are in order concerning this axiom and the others we will present. First, axiom (21.73) is stated using universal quantifiers to bind several of the variables, which essentially says that in all cases in which someone is drunk, all people do not want that person to drive. Although we might hope that this is generally the case, such a statement is nonetheless too strong. The way in which this is handled in the Hobbs et al. system is by including an additional relation, called an *etc* predicate, in the antecedent of such axioms. An *etc* predicate represents all the other properties that must be true for the axiom to apply, but which are too vague to state explicitly. These predicates therefore cannot be proven, they can only be assumed at a corresponding cost. Because rules with high assumption costs will be dispreferred to ones with low costs, the likelihood that the rule applies can be encoded in terms of this cost. Since we have chosen to simplify our discussion by ignoring costs, we will similarly ignore the use of *etc* predicates.

Second, each predicate has what may look like an “extra” variable in the first argument position; for instance, the *drive* predicate has two arguments instead of one. This variable is used to reify the relationship denoted by the predicate so that it can be referred to from argument places in other predicates. For instance, reifying the *drive* predicate with the variable e_k allows us to express the idea of not wanting someone to drive by referring to it in the final argument of the *diswant* predicate.

Picking up where we left off, the second world knowledge axiom we use says that if someone does not want someone else to drive, then they do not want this person to have his car keys, since car keys enable someone to drive.

(21.74)

$$\begin{aligned}&\forall x, y, e_j, e_k\ diswant(e_j, y, e_k) \wedge drive(e_k, x) \ \Rightarrow\\ &\qquad \exists z, e_l, e_m\ diswant(e_l, y, e_m) \wedge have(e_m, x, z)\\ &\qquad\qquad\qquad \wedge carkeys(z, x) \wedge cause(e_j, e_l)\end{aligned}$$

The third axiom says that if someone doesn’t want someone else to have something, he might hide it from him.

(21.75)

$$\begin{aligned}&\forall x, y, z, e_l, e_m\ diswant(e_l, y, e_m) \wedge have(e_m, x, z) \ \Rightarrow\\ &\qquad \exists e_n\ hide(e_n, y, x, z) \wedge cause(e_l, e_n)\end{aligned}$$

The final axiom says simply that causality is transitive, that is, if e_i causes e_j and e_j causes e_k, then e_i causes e_k.

(21.76)

$$\forall e_i, e_j, e_k\ cause(e_i, e_j) \wedge cause(e_j, e_k) \ \Rightarrow\ cause(e_i, e_k)$$

Finally, we have the content of the utterances themselves, that is, that John hid Bill’s car keys (from Bill), (21.77) $hide(e_1, John, Bill, ck) \wedge carkeys(ck, Bill)$

and that someone described using the pronoun “he” was drunk; we will represent the pronoun with the free variable *he*.

(21.78) $drunk(e_2, he)$

We can now see how reasoning with the content of the utterances along with the aforementioned axioms allows the coherence of passage (21.4) to be established under the Explanation relation. The derivation is summarized in Figure 21.9; the sentence interpretations are shown in boxes. We start by assuming there is a coherence relation, and using axiom (21.71) hypothesize that this relation is Explanation,

(21.79) $Explanation(e_1, e_2)$

which, by axiom (21.72), means we hypothesize that

(21.80) $cause(e_2, e_1)$

holds. By axiom (21.76), we can hypothesize that there is an intermediate cause e_3,

(21.81) $cause(e_2, e_3) \wedge cause(e_3, e_1)$

and we can repeat this again by expanding the first conjunct of (21.81) to have an intermediate cause e_4.

(21.82) $cause(e_2, e_4) \wedge cause(e_4, e_3)$

We can take the *hide* predicate from the interpretation of the first sentence in (21.77) and the second *cause* predicate in (21.81), and, using axiom (21.75), hypothesize that John did not want Bill to have his car keys:

(21.83) $diswant(e_3, John, e_5) \wedge have(e_5, Bill, ck)$

From this, the *carkeys* predicate from (21.77), and the second *cause* predicate from (21.82), we can use axiom (21.74) to hypothesize that John does not want Bill to drive:

(21.84) $diswant(e_4, John, e_6) \wedge drive(e_6, Bill)$

From this, axiom (21.73), and the second *cause* predicate from (21.82), we can hypothesize that Bill was drunk:

(21.85) $drunk(e_2, Bill)$

But now we find that we can “prove” this fact from the interpretation of the second sentence if we simply assume that the free variable *he* is bound to Bill. Thus, the establishment of coherence has gone through, as we have identified a chain of reasoning between the sentence interpretations – one that includes unprovable assumptions about axiom choice and pronoun assignment – that results in $cause(e_2, e_1)$, as required for establishing the Explanation relationship.

This derivation illustrates a powerful property of coherence establishment, namely its ability to cause the hearer to infer information about the situation described by the discourse that the speaker has left unsaid. In this case, the derivation required the assumption that John hid Bill’s keys because he did not want him to drive (presumably out of fear of him having an accident, or getting stopped by the police), as opposed to some other explanation, such as playing a practical joke on him. This cause is not stated anywhere in passage (21.4); it arises only from the inference process triggered

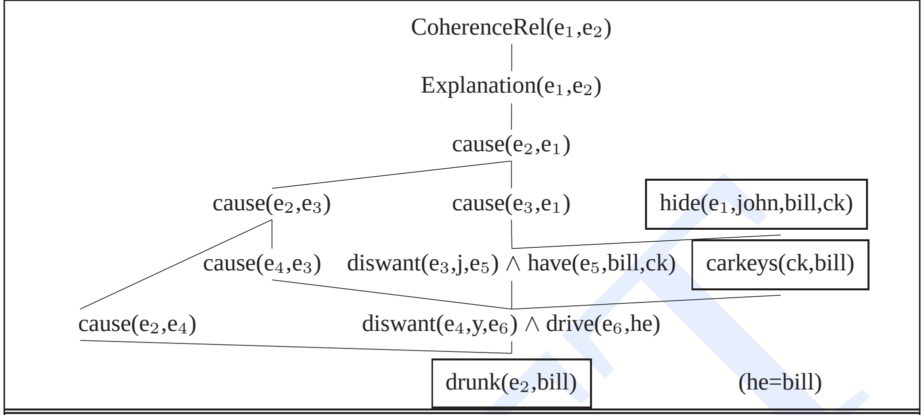


Figure 21.9 Establishing the coherence of passage (21.4).

by the need to establish coherence. In this sense, the meaning of a discourse is greater than the sum of the meanings of its parts. That is, a discourse typically communicates far more information than is contained in the interpretations of the individual sentences that comprise it.

We now return to passage (21.5), repeated below as (21.87), which was notable in that it lacks the coherence displayed by passage (21.4), repeated below as (21.86).

(21.86) John hid Bill's car keys. He was drunk.

(21.87) ?? John hid Bill's car keys. He likes spinach.

We can now see why this is: there is no analogous chain of inference capable of linking the two utterance representations, in particular, there is no causal axiom analogous to (21.73) that says that liking spinach might cause someone to not want you to drive. Without additional information that can support such a chain of inference (such as the aforementioned scenario in which someone promised John spinach in exchange for hiding Bill's car keys), the coherence of the passage cannot be established.

DEFEASIBLE

Because abduction is a form of unsound inference, it must be possible to subsequently retract the assumptions made during abductive reasoning, that is, abductive inferences are **defeasible**. For instance, if passage (21.86) was followed by sentence (21.88),

(21.88) Bill's car isn't here anyway; John was just playing a practical joke on him.

the system would have to retract the original chain of inference connecting the two clauses in (21.86), and replace it with one utilizing the fact that the hiding event was part of a practical joke.

In a more general knowledge base designed to support a broad range of inferences, one would want axioms that are more general than those we used to establish the coherence of passage (21.86). For instance, consider axiom (21.74), which says that if you do not want someone to drive, then you do not want them to have their car keys. A more general form of the axiom would say that if you do not want someone to perform an action, and an object enables them to perform that action, then you do not want them to have the object. The fact that car keys enable someone to drive would then be encoded separately, along with many other similar facts. Likewise, axiom (21.73) says that if someone is drunk, you don't want them to drive. We might replace this with an axiom that says that if someone does not want something to happen, then they don't want something that will likely cause it to happen. Again, the facts that people typically don't want other people to get into car accidents, and that drunk driving causes accidents, would be encoded separately.

While it is important to have computational models that shed light on the coherence establishment problem, large barriers remain for employing this and similar methods on a wide-coverage basis. In particular, the large number of axioms that would be required to encode all of the necessary facts about the world, and the lack of a robust mechanism for constraining inference with such a large set of axioms, makes these methods largely impractical in practice. Nonetheless, approximations to these kinds of knowledge and inferential rules can already play an important role in natural language understanding systems.

21.10 Psycholinguistic Studies of Reference and Coherence

To what extent do the techniques described in this chapter model human discourse comprehension? We summarize here a few selected results from the substantial body of psycholinguistic research; for reasons of space we focus here solely on anaphora resolution.

READING TIME EXPERIMENTS

A significant amount of work has been concerned with the extent to which people use the preferences described in Section 21.5 to interpret pronouns, the results of which are often contradictory. Clark and Sengal (1979) studied the effects that sentence recency plays in pronoun interpretation using a set of **reading time experiments**. After receiving and acknowledging a three sentence context to read, human subjects were given a target sentence containing a pronoun. The subjects pressed a button when they felt that they understood the target sentence. Clark and Sengal found that the reading time was significantly faster when the referent for the pronoun was evoked from the most recent clause in the context than when it was evoked from two or three clauses back. On the other hand, there was no significant difference between referents evoked from two clauses and three clauses back, leading them to claim that "the last clause processed grants the entities it mentions a privileged place in working memory".

QUESTION ANSWERING

REFERENT NAMING TASK

Crawley et al. (1990) compared the grammatical role parallelism preference with a grammatical role preference, in particular, a preference for referents evoked from the subject position of the previous sentence over those evoked from object position. Unlike previous studies which conflated these preferences by considering only subject-to-subject reference effects, Crawley et al. studied pronouns in object position to see if they tended to be assigned to the subject or object of the last sentence. They found that in two task environments – a **question answering task** which revealed how the human subjects interpreted the pronoun, and a **referent naming task** in which the subjects identified the referent of the pronoun directly – the human subjects resolved pronouns to the subject of the previous sentence more often than the object.

However, Smyth (1994) criticized the adequacy of Crawley et al.'s data for evaluating the role of parallelism. Using data that met more stringent requirements for assessing parallelism, Smyth found that subjects overwhelmingly followed the parallelism preference in a referent naming task. The experiment supplied weaker support for the preference for subject referents over object referents, which he posited as a default strategy when the sentences in question are not sufficiently parallel.

Caramazza et al. (1977) studied the effect of the "implicit causality" of verbs on pronoun resolution. Verbs were categorized in terms of having subject bias or object bias using a **sentence completion task**. Subjects were given sentence fragments such as (21.89).

SENTENCE COMPLETION TASK

(21.89) John telephoned Bill because he

The subjects provided completions to the sentences, which identified to the experimenters what referent for the pronoun they favored. Verbs for which a large percentage of human subjects indicated a grammatical subject or object preference were categorized as having that bias. A sentence pair was then constructed for each biased verb: a "congruent" sentence in which the semantics supported the pronoun assignment suggested by the verb's bias, and an "incongruent" sentence in which the semantics supported the opposite prediction. For example, sentence (21.90) is congruent for the subject-bias verb "telephoned", since the semantics of the second clause supports assigning the subject *John* as the antecedent of *he*, whereas sentence (21.91) is incongruent since the semantics supports assigning the object *Bill*.

(21.90) John telephoned Bill because he wanted some information.

(21.91) John telephoned Bill because he withheld some information.

In a referent naming task, Caramazza et al. found that naming times were faster for the congruent sentences than for the incongruent ones. Perhaps surprisingly, this was even true for cases in which the two people mentioned in the first clause were of different genders, thus rendering the reference unambiguous.

Matthews and Chodorow (1988) analyzed the problem of intrasentential reference and the predictions of syntactically-based search strategies. In a question answering task, they found that subjects exhibited slower comprehension times for sentences in which a pronoun antecedent occupied an early, syntactically deep position than for sentences in which the antecedent occupied a late, syntactically shallow position. This result is consistent with the search process used in Hobbs's tree search algorithm.

There has also been psycholinguistic work concerned with testing the principles of centering theory. In a set of reading time experiments, Gordon et al. (1993) found that reading times were slower when the current backward-looking center was referred to using a full noun phrase instead of a pronoun, even though the pronouns were ambiguous and the proper names were not. This effect – which they called a **repeated name penalty** – was found only for referents in subject position, suggesting that the C_b is preferentially realized as a subject. Brennan (1995) analyzed how choice of linguistic form correlates with centering principles. She ran a set of experiments in which a human subject watched a basketball game and had to describe it to a second person. She found that the human subjects tended to refer to an entity using a full noun phrase in subject position before subsequently pronominalizing it, even if the referent had already been introduced in object position.

REPEATED NAME PENALTY

21.11 Summary

In this chapter, we saw that many of the problems that natural language processing systems face operate between sentences, that is, at the *discourse* level. Here is a summary of some of the main points we discussed:

- Discourses, like sentences, have hierarchical structure. In the simplest kind of structure detection, we assume a simpler linear structure, and segment a discourse on topic or other boundaries. The main cues for this are **lexical cohesion** as well as discourse markers/cue phrases.
- Discourses are not arbitrary collections of sentences; they must be *coherent*. Among the factors that make a discourse coherent are coherence relations between the sentences and entity-based coherence.
- Various sets of **coherence relations** and rhetorical relations have been proposed. Algorithms for detecting these coherence relations can use surface-based cues (cue phrases, syntactic information).
- Discourse interpretation requires that one build an evolving representation of discourse state, called a *discourse model*, that contains representations of the entities that have been referred to and the relationships in which they participate.
- Natural languages offer many ways to refer to entities. Each form of reference sends its own signals to the hearer about how it should be processed with respect to her discourse model and set of beliefs about the world.
- Pronominal reference can be used for referents that have an adequate degree of *salience* in the discourse model. There are a variety of lexical, syntactic, semantic, and discourse factors that appear to affect salience.
- The Hobbs, Centering, and Log-linear models for pronominal anaphora offer different ways of drawing on and combining various of these constraints.
- The full NP coreference task also has to deal with names and definite NPs. String edit distance is a useful features for these.
- Advanced algorithms for establishing coherence apply constraints imposed by one or more coherence relations, often leads to the inference of additional information left unsaid by the speaker. The unsound rule of logical *abduction* can be used for performing such inference.

Bibliographical and Historical Notes

Building on the foundations set by early systems for natural language understanding (Woods et al., 1972; Winograd, 1972; Woods, 1978), much of the fundamental work in computational approaches to discourse was performed in the late 70's. Webber's (1978, 1983) work provided fundamental insights into how entities are represented in the discourse model and the ways in which they can license subsequent reference. Many of the examples she provided continue to challenge theories of reference to this day. Grosz (1977) addressed the focus of attention that conversational participants maintain as the discourse unfolds. She defined two levels of focus; entities relevant to the entire discourse were said to be in *global* focus, whereas entities that are locally in focus (i.e., most central to a particular utterance) were said to be in *immediate* focus. Sidner (1979, 1983) described a method for tracking (immediate) discourse foci and their use in resolving pronouns and demonstrative noun phrases. She made a distinction between the current discourse focus and potential foci, which are the predecessors to the backward and forward looking centers of centering theory respectively.

The roots of the centering approach originate from papers by Joshi and Kuhn (1979) and Joshi and Weinstein (1981), who addressed the relationship between immediate focus and the inferences required to integrate the current utterance into the discourse model. Grosz et al. (1983) integrated this work with the prior work of Sidner and Grosz. This led to a manuscript on centering which, while widely circulated since 1986, remained unpublished until Grosz et al. (1995b). A series of papers on centering based on this manuscript/paper were subsequently published (Kameyama, 1986; Brennan et al., 1987; Di Eugenio, 1990; Walker et al., 1994; Di Eugenio, 1996; Strube and Hahn, 1996; Kehler, 1997a, inter alia). A collection of later centering papers appears in Walker et al. (1998), and see Poesio et al. (2004) for more recent work. We have focused in this chapter on Centering and anaphora resolution; Sse Karamanis (2003, 2007), Barzilay and Lapata (2007) and related papers discussed in Ch. 23 for the application of Centering to entity-based coherence.

There is a long history in linguistics of studies of *information status* (Chafe, 1976; Prince, 1981; Ariel, 1990; Prince, 1992; Gundel et al., 1993; Lambrecht, 1994, inter alia).

Beginning with Hobbs's (1978) tree-search algorithm, researchers have pursued syntax-based methods for identifying reference robustly in naturally occurring text. An early system for a weighted combination of different syntactic and other features was Lappin and Leass (1994), which we described in detail in our first edition. Kennedy and Boguraev (1996) describe a similar system that does not rely on a full syntactic parser, but merely a mechanism for identifying noun phrases and labeling their grammatical roles. Both approaches use Alshawi's (1987) framework for integrating salience factors. An algorithm that uses this framework for resolving references in a multimodal (i.e., speech and gesture) human-computer interface is described in Huls et al. (1995). A discussion of a variety of approaches to reference in operational systems can be found in Mitkov and Boguraev (1997).

Methods for reference resolution based on supervised learning were proposed quite early (Connolly et al., 1994; Aone and Bennett, 1995; McCarthy and Lehnert, 1995; Kehler, 1997b; Ge et al., 1998, inter alia). More recently both supervised and unsupervised approaches have received a lot of research attention, focused both on anaphora resolution Kehler et al. (2004), Bergsma and Lin (2006) and full NP coreference (Cardie and Wagstaff, 1999; Ng and Cardie, 2002b; Ng, 2005). For definite NP reference, there are general algorithms (Poesio and Vieira, 1998; Vieira and Poesio, 2000), as well as specific algorithms that focus on deciding if a particular definite NP is anaphoric or not (Bean and Riloff, 1999, 2004; Ng and Cardie, 2002a; Ng, 2004).

Mitkov (2002) is an excellent comprehensive overview of anaphora resolution.

The idea of using cohesion for linear discourse segmentation was implicit in the groundbreaking work of (Halliday and Hasan, 1976), but was first explicitly implemented by Morris and Hirst (1991), and quickly picked up by many other researchers, including (Kozima, 1993; Reynar, 1994; Hearst, 1994, 1997; Reynar, 1999; Kan et al., 1998; Choi, 2000; Choi et al., 2001; Brants et al., 2002; Bestgen, 2006). Power et al. (2003) studies discourse structure, while Filippova and Strube (2006), Sporleder and Lapata (2004, 2006) focus on paragraph segmentation.

The use of cue phrases in segmentation has been widely studied, including work on many textual genres as well as speech (Passonneau and Litman, 1993; Hirschberg and Litman, 1993; Manning, 1998; Kawahara et al., 2004)

Many researchers have posited sets of coherence relations that can hold between utterances in a discourse (Halliday and Hasan, 1976; Hobbs, 1979; Longacre, 1983; Mann and Thompson, 1987; Polanyi, 1988; Hobbs, 1990; Sanders et al., 1992; Carlson et al., 2001, 2002; Asher and Lascarides, 2003; Baldridge et al., 2007, inter alia). A compendium of over 350 relations that have been proposed in the literature can be found in Hovy (1990).

There are a wide variety of approaches to coherence extraction. The cue-phrase based model described in Sec. 21.2.2 is due to Daniel Marcu and colleagues (Marcu, 2000b, 2000a; Carlson et al., 2001, 2002). The Linguistic Discourse Model (Polanyi, 1988; Scha and Polanyi, 1988; Polanyi et al., 2004a, 2004b) is a framework in which discourse syntax is more heavily emphasized; in this approach, a discourse parse tree is built on a clause-by-clause basis in direct analogy with how a sentence parse tree is built on a constituent-by-constituent basis. Corston-Oliver (1998) also explores explores syntactic and parser-based features. A more recent line of work has applied a version of the tree-adjoining grammar formalism to discourse parsing (Webber et al., 1999; Webber, 2004). This model has also been used to annotate the Penn Discourse Treebank (Miltsakaki et al., 2004b, 2004a). See Asher and Lascarides (2003) and Baldridge et al. (2007) on Segmented Discourse Representation Structure (**SDRT**). Wolf and Gibson (2005) argue that coherence structure includes crossed bracketings which make it impossible to represent as a tree, and propose a graph representation instead.

SDRT

In addition to determining discourse structure and meaning, theories of discourse coherence have been used in algorithms for interpreting discourse-level linguistic phenomena, including pronoun resolution (Hobbs, 1979; Kehler, 2000), verb phrase ellipsis and gapping (Prüst, 1992; Asher, 1993; Kehler, 1993, 1994a), and tense interpretation (Lascarides and Asher, 1993; Kehler, 1994b, 2000). An extensive investigation into the relationship between coherence relations and discourse connectives can be found in Knott and Dale (1994).

Exercises

21.1 Early work in syntactic theory attempted to characterize rules for pronominalization through purely syntactic means. A rule was proposed in which a pronoun was interpreted by deleting it from the syntactic structure of the sentence that contains it, and replacing it with the syntactic representation of the antecedent noun phrase.

Explain why the following sentences (called "Bach-Peters" sentences) are problematic for such an analysis:

(21.92) The man who deserves it gets the prize he wants.

(21.93) The pilot who shot at it hit the MIG that chased him.

What other types of reference discussed on pages 18–21 are problematic for this type of analysis?

21.2 Draw syntactic trees for example (21.66) on page 27 and apply Hobbs's tree search algorithm to it, showing each step in the search.

21.3 Hobbs (1977) cites the following examples from his corpus as being problematic for his tree-search algorithm:

(21.94) The positions of pillars in one hall were marked by river boulders and a shaped convex cushion of bronze that had served as their footings.

(21.95) They were at once assigned an important place among the scanty remains which record the physical developments of the human race from the time of its first appearance in Asia.

(21.96) Sites at which the coarse grey pottery of the Shang period has been discovered do not extend far beyond the southernmost reach of the Yellow river, or westward beyond its junction with the Wei.

(21.97) The thin, hard, black-burnished pottery, made in shapes of angular profile, which archaeologists consider as the clearest hallmark of the Lung Shan culture, developed in the east. The site from which it takes its name is in Shantung. It is traced to the north-east as far as Liao-ning province.

(21.98) He had the duty of performing the national sacrifices to heaven and earth: his role as source of honours and material rewards for services rendered by feudal lords and ministers is commemorated in thousands of inscriptions made by the recipients on bronze vessels which were eventually deposited in their graves.

In each case, identify the correct referent of the underlined pronoun and the one that the algorithm will identify incorrectly. Discuss any factors that come into play in determining the correct referent in each case, and what types of information might be necessary to account for them.

21.4 Implement the Hobbs algorithm. Test it on a sample of the Penn TreeBank. You will need to modify the algorithm to deal with differences between the Hobbs and TreeBank grammars.

21.5 Consider the following passage, from Brennan et al. (1987):

(21.99) Brennan drives an Alfa Romeo.
She drives too fast.
Friedman races her on weekends.
She goes to Laguna Seca.

Identify the referent that the BFP algorithm finds for the pronoun in the final sentence. Do you agree with this choice, or do you find the example ambiguous? Discuss why introducing a new noun phrase in subject position, with a pronominalized reference in object position, might lead to an ambiguity for a subject pronoun in the next sentence. What preferences are competing here?

21.6 Consider passages (21.100a-b), adapted from Winograd (1972).

(21.100) The city council denied the demonstrators a permit because

a. they feared violence.
b. they advocated violence.

What are the correct interpretations for the pronouns in each case? Sketch out an analysis of each in the interpretation as abduction framework, in which these reference assignments are made as a by-product of establishing the Explanation relation.

21.7 Select an editorial column from your favorite newspaper, and determine the discourse structure for a 10-20 sentence portion. What problems did you encounter? Were you helped by superficial cues the speaker included (e.g., discourse connectives) in any places?

Alshawi, H. (1987). *Memory and Context for Language Interpretation*. Cambridge University Press.

Aone, C. and Bennett, S. W. (1995). Evaluating automated and manual acquisition of anaphora resolution strategies. In *ACL-95*, Cambridge, MA, pp. 122–129.

Ariel, M. (2001). Accessibility theory: An overview. In Sanders, T., Schilperoord, J., and Spooren, W. (Eds.), *Text Representation: Linguistic and Psycholinguistic Aspects*, pp. 29–87. Benjamins.

Ariel, M. (1990). *Accessing Noun Phrase Antecedents*. Routledge.

Asher, N. (1993). *Reference to Abstract Objects in Discourse*. SLAP 50, Dordrecht, Kluwer.

Asher, N. and Lascarides, A. (2003). *Logics of Conversation*. Cambridge University Press.

Baldridge, J., Asher, N., and Hunter, J. (2007). Annotation for and robust parsing of discourse structure on unrestricted texts. *Zeitschrift für Sprachwissenschaft*. In press.

Barzilay, R. and Lapata, M. (2007). Modeling local coherence: an entity-based approach. *Computational Linguistics*. To appear.

Bean, D. and Riloff, E. (1999). Corpus-based identification of non-anaphoric noun phrases. In *ACL-99*, pp. 373–380.

Bean, D. and Riloff, E. (2004). Unsupervised learning of contextual role knowledge for coreference resolution. In *HLT-NAACL-04*.

Beeferman, D., Berger, A., and Lafferty, J. D. (1999). Statistical Models for Text Segmentation. *Machine Learning*, *34*(1), 177–210.

Bergsma, S. and Lin, D. (2006). Bootstrapping path-based pronoun resolution. In *COLING/ACL 2006*, Sydney, Australia, pp. 33–40.

Bestgen, Y. (2006). Improving Text Segmentation Using Latent Semantic Analysis: A Reanalysis of Choi, Wiemer-Hastings, and Moore (2001). *Computational Linguistics*, *32*(1), 5–12.

Brants, T., Chen, F., and Tsochantaridis, I. (2002). Topic-based document segmentation with probabilistic latent semantic analysis. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pp. 211–218.

Brennan, S. E. (1995). Centering attention in discourse. *Language and Cognitive Processes*, *10*, 137–167.

Brennan, S. E., Friedman, M. W., and Pollard, C. (1987). A centering approach to pronouns. In *ACL-87*, Stanford, CA, pp. 155–162.

Caramazza, A., Grober, E., Garvey, C., and Yates, J. (1977). Comprehension of anaphoric pronouns. *Journal of Verbal Learning and Verbal Behaviour*, *16*, 601–609.

Cardie, C. and Wagstaff, K. (1999). Noun phrase coreference as clustering. In *EMNLP/VLC-99*, College Park, MD.

Carlson, L. and Marcu, D. (2001). Discourse tagging manual. Tech. rep. ISI-TR-545, ISI.

Carlson, L., Marcu, D., and Okurowski, M. E. (2001). Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *Proceedings of SIGDIAL*.

Carlson, L., Marcu, D., and Okurowski, M. E. (2002). Building a discourse-tagged corpus in the framework of rhetorical structure theory. In van Kuppevelt, J. and Smith, R. (Eds.), *Current Directions in Discourse and Dialogue*. Kluwer.

Chafe, W. L. (1976). Givenness, contrastiveness, definiteness, subjects, topics, and point of view. In Li, C. N. (Ed.), *Subject and Topic*, pp. 25–55. Academic Press.

Charniak, E. and Shimony, S. E. (1990). Probabilistic semantics for cost based abduction. In Dietterich, T. S. W. (Ed.), *AAAI-90*, pp. 106–111. MIT Press.

Charniak, E. and Goldman, R. (1988). A logic for semantic interpretation. In *Proceedings of the 26th ACL*, Buffalo, NY.

Charniak, E. and McDermott, D. (1985). *Introduction to Artificial Intelligence*. Addison Wesley.

Choi, F. Y. Y. (2000). Advances in domain independent linear text segmentation. In *NAACL 2000*, pp. 26–33.

Choi, F. Y. Y., Wiemer-Hastings, P., and Moore, J. (2001). Latent semantic analysis for text segmentation. In *EMNLP 2001*, pp. 109–117.

Chomsky, N. (1981). *Lectures on Government and Binding*. Foris, Dordrecht.

Clark, H. H. and Sengal, C. J. (1979). In search of referents for nouns and pronouns. *Memory and Cognition*, *7*, 35–41.

Connolly, D., Burger, J. D., and Day, D. S. (1994). A machine learning approach to anaphoric reference. In *Proceedings of the International Conference on New Methods in Language Processing (NeMLaP)*.

Corston-Oliver, S. H. (1998). Identifying the linguistic correlates of rhetorical relations. In *Workshop on Discourse Relations and Discourse Markers*, pp. 8–14.

Crawley, R. A., Stevenson, R. J., and Kleinman, D. (1990). The use of heuristic strategies in the interpretation of pronouns. *Journal of Psycholinguistic Research*, *19*, 245–264.

Di Eugenio, B. (1990). Centering theory and the Italian pronominal system. In *COLING-90*, Helsinki, pp. 270–275.

Di Eugenio, B. (1996). The discourse functions of Italian subjects: A centering approach. In *COLING-96*, Copenhagen, pp. 352–357.

Filippova, K. and Strube, M. (2006). Using linguistically motivated features for paragraph boundary identification. In *EMNLP 2006*.

Ge, N., Hale, J., and Charniak, E. (1998). A statistical approach to anaphora resolution. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pp. 161–171.

Gordon, P. C., Grosz, B. J., and Gilliom, L. A. (1993). Pronouns, names, and the centering of attention in discourse. *Cognitive Science*, *17*(3), 311–347.

Grosz, B. J. (1977). The representation and use of focus in a system for understanding dialogs. In *IJCAI-77*, pp. 67–76. Morgan Kaufmann. Reprinted in Grosz et al. (1986).

Grosz, B. J., Joshi, A. K., and Weinstein, S. (1983). Providing a unified account of definite noun phrases in English. In *ACL-83*, pp. 44–50.

Grosz, B. J., Joshi, A. K., and Weinstein, S. (1995a). Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, *21*(2), 203–225.

Grosz, B. J., Joshi, A. K., and Weinstein, S. (1995b). Centering: A framework for modelling the local coherence of discourse. *Computational Linguistics*, *21*(2).

Gundel, J. K., Hedberg, N., and Zacharski, R. (1993). Cognitive status and the form of referring expressions in discourse. *Language*, *69*(2), 274–307.

Halliday, M. A. K. and Hasan, R. (1976). *Cohesion in English*. Longman, London. English Language Series, Title No. 9.

Haviland, S. E. and Clark, H. H. (1974). What's new? Acquiring new information as a process in comprehension. *Journal of Verbal Learning and Verbal Behaviour*, *13*, 512–521.

Hearst, M. A. (1994). Multi-paragraph segmentation of expository text. In *Proceedings of the 32nd ACL*, pp. 9–16.

Hearst, M. A. (1997). Texttiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, *23*, 33–64.

Hirschberg, J. and Litman, D. J. (1993). Empirical Studies on the Disambiguation of Cue Phrases. *Computational Linguistics*, *19*(3), 501–530.

Hobbs, J. R. (1977). 38 examples of elusive antecedents from published texts. Tech. rep. 77-2, Department of Computer Science, City University of New York.

Hobbs, J. R. (1978). Resolving pronoun references. *Lingua*, *44*, 311–338. Reprinted in Grosz et al. (1986).

Hobbs, J. R. (1979). Coherence and coreference. *Cognitive Science*, *3*, 67–90.

Hobbs, J. R. (1990). *Literature and Cognition*. CSLI Lecture Notes 21.

Hobbs, J. R., Stickel, M. E., Appelt, D. E., and Martin, P. (1993). Interpretation as abduction. *Artificial Intelligence*, *63*, 69–142.

Hovy, E. H. (1990). Parsimonious and profligate approaches to the question of discourse structure relations. In *Proceedings of the Fifth International Workshop on Natural Language Generation*, Dawson, PA, pp. 128–136.

Huls, C., Bos, E., and Classen, W. (1995). Automatic referent resolution of deictic and anaphoric expressions. *Computational Linguistics*, *21*(1), 59–79.

Joshi, A. K. and Kuhn, S. (1979). Centered logic: The role of entity centered sentence representation in natural language inferencing. In *IJCAI-79*, pp. 435–439.

Joshi, A. K. and Weinstein, S. (1981). Control of inference: Role of some aspects of discourse structure – centering. In *IJCAI-81*, pp. 385–387.

Kameyama, M. (1986). A property-sharing constraint in centering. In *ACL-86*, New York, pp. 200–206.

Kan, M. Y., Klavans, J. L., and McKeown, K. R. (1998). Linear segmentation and segment significance. In *Proc. 6th Workshop on Very Large Corpora (WVLC-98)*, Montreal, Canada, pp. 197–205.

Karamanis, N. (2003). *Entity Coherence for Descriptive Text Structuring*. Ph.D. thesis, University of Edinburgh.

Karamanis, N. (2007). Supplementing entity coherence with local rhetorical relations for information ordering. *Journal of Logic, Language and Information*. To appear.

Kawahara, T., Hasegawa, M., Shitaoka, K., Kitade, T., and Nanjo, H. (2004). Automatic indexing of lecture presentations using unsupervised learning of presumed discourse markers. *Speech and Audio Processing, IEEE Transactions on*, *12*(4), 409–419.

Kehler, A. (1993). The effect of establishing coherence in ellipsis and anaphora resolution. In *Proceedings of the 31st ACL*, Columbus, Ohio, pp. 62–69.

Kehler, A. (1994a). Common topics and coherent situations: Interpreting ellipsis in the context of discourse inference. In *Proceedings of the 32nd ACL*, Las Cruces, New Mexico, pp. 50–57.

Kehler, A. (1994b). Temporal relations: Reference or discourse coherence?. In *Proceedings of the 32nd ACL*, Las Cruces, New Mexico, pp. 319–321.

Kehler, A. (1997a). Current theories of centering for pronoun interpretation: A critical evaluation. *Computational Linguistics*, *23*(3), 467–475.

Kehler, A. (1997b). Probabilistic coreference in information extraction. In *EMNLP 1997*, Providence, RI, pp. 163–173.

Kehler, A. (2000). *Coherence, Reference, and the Theory of Grammar*. CSLI Publications.

Kehler, A., Appelt, D. E., Taylor, L., and Simma, A. (2004). The (non)utility of predicate-argument frequencies for pronoun interpretation. In *HLT-NAACL-04*.

Kennedy, C. and Boguraev, B. (1996). Anaphora for everyone: Pronominal anaphora resolution without a parser. In *COLING-96*, Copenhagen, pp. 113–118.

Knott, A. and Dale, R. (1994). Using linguistic phenomena to motivate a set of coherence relations. *Discourse Processes*, *18*(1), 35–62.

Kozima, H. (1993). Text segmentation based on similarity between words. In *Proceedings of the 31st ACL*, pp. 286–288.

Lambrecht, K. (1994). *Information Structure and Sentence Form*. Cambridge University Press.

Lappin, S. and Leass, H. (1994). An algorithm for pronominal anaphora resolution. *Computational Linguistics*, *20*(4), 535–561.

Lascarides, A. and Asher, N. (1993). Temporal interpretation, discourse relations, and common sense entailment. *Linguistics and Philosophy*, *16*(5), 437–493.

Longacre, R. E. (1983). *The Grammar of Discourse*. Plenum Press.

Mann, W. C. and Thompson, S. A. (1987). Rhetorical structure theory: A theory of text organization. Tech. rep. RS-87-190, Information Sciences Institute.

Manning, C. D. (1998). Rethinking text segmentation models: An information extraction case study. Tech. rep. SULTRY-98-07-01, University of Sydney.

Marcu, D. (2000a). The rhetorical parsing of unrestricted texts: A surface-based approach. *Computational Linguistics*, *26*(3), 395–448.

Marcu, D. (Ed.). (2000b). *The Theory and Practice of Discourse Parsing and Summarization*. MIT Press.

Marcu, D. and Echihabi, A. (2002). An unsupervised approach to recognizing discourse relations. In *ACL-02*, pp. 368–375.

Matthews, A. and Chodorow, M. S. (1988). Pronoun resolution in two-clause sentences: Effects of ambiguity, antecedent location, and depth of embedding. *Journal of Memory and Language*, *27*, 245–260.

McCarthy, J. F. and Lehnert, W. G. (1995). Using decision trees for coreference resolution. In *IJCAI-95*, Montreal, Canada, pp. 1050–1055.

Miltsakaki, E., Prasad, R., Joshi, A. K., and Webber, B. L. (2004a). Annotating discourse connectives and their arguments. In *Proceedings of the NAACL/HLT Workshop: Frontiers in Corpus Annotation*.

Miltsakaki, E., Prasad, R., Joshi, A. K., and Webber, B. L. (2004b). The Penn Discourse Treebank. In *LREC-04*.

Mitkov, R. (2002). *Anaphora Resolution*. Longman.

Mitkov, R. and Boguraev, B. (Eds.). (1997). *Proceedings of the ACL-97 Workshop on Operational Factors in Practical, Robust Anaphora Resolution for Unrestricted Texts*, Madrid, Spain.

Morris, J. and Hirst, G. (1991). Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational Linguistics*, *17*(1), 21–48.

Ng, V. (2004). Learning noun phrase anaphoricity to improve coreference resolution: Issues in representation and optimization. In *ACL-04*.

Ng, V. (2005). Machine learning for coreference resolution: From local classification to global ranking. In *ACL-05*.

Ng, V. and Cardie, C. (2002a). Identifying anaphoric and non-anaphoric noun phrases to improve coreference resolution. In *COLING-02*.

Ng, V. and Cardie, C. (2002b). Improving machine learning approaches to coreference resolution. In *ACL-02*.

Nissim, M., Dingare, S., Carletta, J., and Steedman, M. (2004). An annotation scheme for information status in dialogue. In *LREC-04*, Lisbon.

Passonneau, R. and Litman, D. J. (1993). Intention-based segmentation: Human reliability and correlation with linguistic cues. In *Proceedings of the 31st ACL*, Columbus, Ohio, pp. 148–155.

Peirce, C. S. (1955). Abduction and induction. In Buchler, J. (Ed.), *Philosophical Writings of Peirce*, pp. 150–156. Dover Books, New York.

Pevzner, L. and Hearst, M. A. (2002). A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, *28*(1), 19–36.

Poesio, M., Stevenson, R., Di Eugenio, B., and Hitzeman, J. (2004). Centering: A parametric theory and its instantiations. *Computational Linguistics*, *30*(3), 309–363.

Poesio, M. and Vieira, R. (1998). A corpus-based investigation of definite description use. *Computational Linguistics*, *24*(2), 183–216.

Polanyi, L., Culy, C., van den Berg, M., Thione, G. L., and Ahn, D. (2004a). A Rule Based Approach to Discourse Parsing. In *Proceedings of SIGDIAL*.

Polanyi, L., Culy, C., van den Berg, M., Thione, G. L., and Ahn, D. (2004b). Sentential Structure and Discourse Parsing. In *Discourse Annotation Workshop, ACL04*.

Polanyi, L. (1988). A formal model of the structure of discourse. *Journal of Pragmatics*, *12*.

Power, R., Scott, D., and Bouayad-Agha, N. (2003). Document structure. *Computational Linguistics*, *29*(2), 211–260.

Prince, E. (1981). Toward a taxonomy of given-new information. In Cole, P. (Ed.), *Radical Pragmatics*, pp. 223–255. Academic Press.

Prince, E. (1992). The ZPG letter: Subjects, definiteness, and information-status. In Thompson, S. and Mann, W. (Eds.), *Discourse Description: Diverse Analyses of a Fundraising Text*, pp. 295–325. John Benjamins, Philadelphia/Amsterdam.

Prüst, H. (1992). *On Discourse Structuring, VP Anaphora, and Gapping*. Ph.D. thesis, University of Amsterdam.

Reynar, J. C. (1994). An automatic method of finding topic boundaries. In *Proceedings of the 32nd ACL*, pp. 27–30.

Reynar, J. C. (1999). Statistical models for topic segmentation. In *ACL/EACL-97*, pp. 357–364.

Sanders, T. J. M., Spooren, W. P. M., and Noordman, L. G. M. (1992). Toward a taxonomy of coherence relations. *Discourse Processes*, *15*, 1–35.

Scha, R. and Polanyi, L. (1988). An augmented context free grammar for discourse. In *COLING-88*, Budapest, pp. 573–577.

Sidner, C. L. (1979). Towards a computational theory of definite anaphora comprehension in English discourse. Tech. rep. 537, MIT Artificial Intelligence Laboratory, Cambridge, MA.

Sidner, C. L. (1983). Focusing in the comprehension of definite anaphora. In Brady, M. and Berwick, R. C. (Eds.), *Computational Models of Discourse*, pp. 267–330. MIT Press.

Smyth, R. (1994). Grammatical determinants of ambiguous pronoun resolution. *Journal of Psycholinguistic Research*, *23*, 197–229.

Soon, W. M., Ng, H. T., and Lim, D. C. Y. (2001). A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, *27*(4), 521–544.

Sporleder, C. and Lapata, M. (2004). Automatic paragraph identification: A study across languages and domains. In *EMNLP 2004*.

Sporleder, C. and Lapata, M. (2006). Automatic Paragraph Identification: A Study across Languages and Domains. *ACM Transactions on Speech and Language Processing (TSLP)*, *3*(2).

Sporleder, C. and Lascarides, A. (2005). Exploiting Linguistic Cues to Classify Rhetorical Relations. In *Proceedings of the Recent Advances in Natural Language Processing (RANLP-05), Borovets, Bulgaria*.

Strube, M. and Hahn, U. (1996). Functional centering. In *ACL-96*, Santa Cruz, CA, pp. 270–277.

Sundheim, B. (1995). Overview of results of the MUC-6 evaluation. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, MD, pp. 13–31.

van Deemter, K. and Kibble, R. (2000). On coreferring: coreference in muc and related annotation schemes. *Computational Linguistics*, *26*(4), 629–637.

Vieira, R. and Poesio, M. (2000). An empirically based system for processing definite descriptions. *Computational Linguistics*, *26*(4), 539–593.

Vilain, M., Burger, J. D., Aberdeen, J., Connolly, D., and Hirschman, L. (1995). A model-theoretic coreference scoring scheme. In *MUC6 '95: Proceedings of the 6th Conference on Message Understanding*. ACL.

Walker, M. A., Iida, M., and Cote, S. (1994). Japanese discourse and the process of centering. *Computational Linguistics*, *20*(2).

Walker, M. A., Joshi, A. K., and Prince, E. (Eds.). (1998). *Centering in Discourse*. Oxford University Press.

Webber, B. L. (2004). D-LTAG: extending lexicalized TAG to discourse. *Cognitive Science*, *28*(5), 751–79.

Webber, B. L., Knott, A., Stone, M., and Joshi, A. (1999). Discourse relations: A structural and presuppositional account using lexicalised TAG. In *ACL-99*, College Park, MD, pp. 41–48.

Webber, B. L. (1978). *A Formal Approach to Discourse Anaphora*. Ph.D. thesis, Harvard University.

Webber, B. L. (1983). So what can we talk about now?. In Brady, M. and Berwick, R. C. (Eds.), *Computational Models of Discourse*, pp. 331–371. The MIT Press. Reprinted in Grosz et al. (1986).

Webber, B. L. (1991). Structure and ostension in the interpretation of discourse deixis. *Language and Cognitive Processes*, *6*(2), 107–135.

Winograd, T. (1972). *Understanding Natural Language*. Academic Press.

Wolf, F. and Gibson, E. (2005). Representing discourse coherence: A corpus-based analysis. *Computational Linguistics*, *31*(2), 249–287.

Woods, W. A. (1978). Semantics and quantification in natural language question answering. In Yovits, M. (Ed.), *Advances in Computers*, Vol. 17, pp. 2–87. Academic Press.

Woods, W. A., Kaplan, R. M., and Nash-Webber, B. L. (1972). The Lunar Sciences Natural Language Information System: Final report. Tech. rep. 2378, Bolt, Beranek, and Newman, Inc., Cambridge, MA.

22 INFORMATION EXTRACTION

I am the very model of a modern Major-General,
I've information vegetable, animal, and mineral,
I know the kings of England, and I quote the fights historical
From Marathon to Waterloo, in order categorical...

Gilbert and Sullivan, *Pirates of Penzance*

Imagine that you are an analyst with an investment firm that tracks airline stocks. You're given the task of determining the relationship (if any) between airline announcements of fare hikes and the behavior of their stocks on the following day. Historical data about stock prices is easy to come by, but what about the information about airline announcements? To do a reasonable job on this task, you would need to know at least the name of the airline, the nature of the proposed fare hike, the dates of the announcement and possibly the response of other airlines. Fortunately, this information resides in archives of news articles reporting on airline's actions, as in the following recent example.

> Citing high fuel prices, United Airlines said Friday it has increased fares by \$6 per round trip on flights to some cities also served by lower-cost carriers. American Airlines, a unit of AMR Corp., immediately matched the move, spokesman Tim Wagner said. United, a unit of UAL Corp., said the increase took effect Thursday and applies to most routes where it competes against discount carriers, such as Chicago to Dallas and Denver to San Francisco.

Of course, distilling information like names, dates and amounts from naturally occurring text is a non-trivial task. This chapter presents a series of techniques that can be used to extract limited kinds of semantic content from text. This process of **information extraction** (IE) turns the unstructured information embedded in texts into structured data. More concretely, information extraction is an effective way to to populate the contents of a relational database. Once the information is encoded formally, we can apply all the capabilities provided by database systems,

INFORMATION EXTRACTION

statistical analysis packages and other forms of decision support systems to address the problems we're trying to solve.

As we proceed through this chapter, we'll see that robust solutions to IE problems are actually clever combinations of techniques we've seen earlier in the book. In particular, the finite-state methods described in Chs. 2 and 3, the probabilistic models introduced in Chs. 4 through 6 and the syntactic chunking methods from Ch. 13 form the core of most current approaches to information extraction. Before diving into the details of how these techniques are applied, let's quickly introduce the major problems in IE and how they can be approached.

NAMED ENTITY RECOGNITION

The first step in most IE tasks is to detect and classify all the proper names mentioned in a text — a task generally referred to as **named entity recognition** (NER). Not surprisingly, what constitutes a proper name and the particular scheme used to classify them is application-specific. Generic NER systems tend to focus on finding the names of people, places and organizations that are mentioned in ordinary news texts; practical applications have also been built to detect everything from the names of genes and proteins (Settles, 2005) to the names of college courses (McCallum, 2005).

NAMED ENTITY MENTIONS

Our introductory example contains 13 instances of proper names, which we'll refer to as **named entity mentions**, which can be classified as either organizations, people, places, times or amounts.

Having located all of the mentions of named entities in a text, it is useful to link, or cluster, these mentions into sets that correspond to the entities behind the mentions. This is the task of **reference resolution**, which we introduced in Ch. 21, and is also an important component in IE. In our sample text, we would like to know that the *United Airlines* mention in the first sentence and the *United* mention in the third sentence refer to the same real world entity. This general reference resolution problem also includes anaphora resolution as a sub-problem. In this case, determining that the two uses of *it* refer to *United Airlines* and *United* respectively.

RELATION DETECTION AND CLASSIFICATION

The task of **relation detection and classification** is to find and classify semantic relations among the entities discovered in a given text. In most practical settings, the focus of relation detection is on small fixed sets of binary relations. Generic relations that appear in standard system evaluations include family, employment, part-whole, membership, and geospatial relations. The relation detection and classification task is the one that most closely corresponds to the problem of populating a relational database. Relation detection among entities is also closely related to the problem of discovering semantic relations among words introduced in Ch. 20.

Our sample text contains 3 explicit mentions of generic relations: *United* is a part of *UAL*, *American Airlines* is a part of *AMR* and *Tim Wagner* is an employee of *American Airlines*. Domain-specific relations from the airline industry would include the fact that *United* serves *Chicago*, *Dallas*, *Denver* and *San Francisco*.

EVENT DETECTION AND CLASSIFICATION

In addition to knowing about the entities in a text and their relation to one another, we might like to find and classify the events in which the entities are participating; this is the problem of **event detection and classification**. In our sample text, the key events are the fare increase by *United* and the ensuing increase by *American*. In addition, there are several events reporting these main events as indicated by the two uses of *said* and the use of *cite*. As with entity recognition, event detection brings with it the problem of reference resolution; we need to figure out which of the many event mentions in a text refer to the same event. In our running example, the events referred to as *the move* and *the increase* in the second and third sentences are the same as the *increase* in the first sentence.

TEMPORAL EXPRESSION DETECTION

TEMPORAL ANALYSIS

The problem of figuring out when the events in a text happened and how they relate to each other in time raises the twin problems of **temporal expression detection** and **temporal analysis**. Temporal expression detection tells us that our sample text contains the temporal expressions *Friday* and *Thursday*. Temporal expressions include date expressions such as days of the week, months, holidays, etc., as well as relative expressions including phrases like *two days from now* or *next year*. They also include expressions for clock times such as *noon* or *3:30PM*.

The overall problem of **temporal analysis** is to map temporal expressions onto specific calendar dates or times of day and then to use those times to situate events in time. It includes the following subtasks.

- Fixing the temporal expressions with respect to an anchoring date or time, typically the dateline of the story in the case of news stories;
- Associating temporal expressions with the events in the text;
- Arranging the events into a complete and coherent timeline.

In our sample text, the temporal expressions *Friday* and *Thursday* should be anchored with respect to the dateline associated with the article itself. We also know that *Friday* refers to the time of United's announcement, and *Thursday* refers to the time that the fare increase went into effect (i.e. the Thursday immediately preceding the Friday). Finally, we can use this information to produce a timeline where United's announcement follows the fare increase and American's announcement follows both of those events. Temporal analysis of this kind is useful in nearly any NLP application that deals with meaning, including question answering, summarization and dialogue systems.

TEMPLATE-FILLING

Finally, many texts describe stereotypical situations that recur with some frequency in the domain of interest. The task of **template-filling** is to find documents that evoke such situations and then fill the slots in templates with appropriate material. These slot-fillers may consist of text segments extracted directly from the text, or they may consist of concepts that have been inferred from text elements via some additional processing (times, amounts, entities from an ontology, etc.).

Our airline text is an example of this kind of stereotypical situation since airlines are often attempting to raise fares and then waiting to see if competitors follow along. In this situation, we can identify *United* as a lead airline that initially raised its fares, $6 as the amount by which fares are being raised, *Thursday* as the effective date for the fare increase, and *American* as an airline that followed along. A filled template from our original airline story might look like the following.

$$\text{FARE-RAISE ATTEMPT:}\quad \begin{bmatrix} \text{LEAD AIRLINE:} & \text{UNITED AIRLINES} \\ \text{AMOUNT:} & \$6 \\ \text{EFFECTIVE DATE:} & \text{2006-10-26} \\ \text{FOLLOWER:} & \text{AMERICAN AIRLINES} \end{bmatrix}$$

The following sections will review current approaches to each of these problems in the context of generic news text. Sec. 22.5 then describes how many of these problems arise in the context of procecessing biology texts.

22.1 NAMED ENTITY RECOGNITION

NAMED ENTITY

The starting point for most information extraction applications is the detection and classification of the named entities in a text. By **named entity**, we simply mean anything that can be referred to with a proper name. This process of **named entity recognition** refers to the combined task of finding spans of text that constitute proper names and then classifying the entities being referred to according to their type.

Generic news-oriented NER systems focus on the detection of things like people, places, and organizations. Figures 22.1 and 22.2 provide lists of typical named entity types with examples of each. Specialized applications may be concerned with many other types of entities, including commercial products, weapons, works of art, or as we'll see in Sec. 22.5, proteins, genes and other biological entities. What these applications all share is a concern with proper names, the characteristic ways that such names are signaled in a given language or genre, and a fixed set of categories of entities from a domain of interest.

By the way that names are signaled, we simply mean that names are denoted in a way that sets them apart from ordinary text. For example, if we're dealing with standard English text, then two adjacent capitalized words in the middle of a text are likely to constitute a name. Further, if they are are preceded by a *Dr.* or followed by an *MD*, then it is likely that we're dealing with a person. In contrast, if they are preceded by *arrived in* or followed by *NY* then we're probably dealing

Type	Tag	Sample Categories
People	PER	Individuals, fictional characters, small groups
Organization	ORG	Companies, agencies, political parties, religious groups, sports teams
Location	LOC	Physical extents, mountains, lakes, seas
Geo-Political Entity	GPE	Countries, states, provinces, counties
Facility	FAC	Bridges, buildings, airports
Vehicles	VEH	Planes, trains and automobiles

Figure 22.1 A list of generic named entity types with the kinds of entities they refer to.

Type	Example
People	*Turing* is often considered to be the father of modern computer science.
Organization	The *IPCC* said it is likely that future tropical cyclones will become more intense.
Location	The *Mt. Sanitas* loop hike begins at the base of *Sunshine Canyon*.
Geo-Political Entity	*Palo Alto* is looking at raising the fees for parking in the University Avenue district
Facility	Drivers were advised to consider either the *Tappan Zee Bridge* or the *Lincoln Tunnel.*
Vehicles	The updated *Mini Cooper* retains its charm and agility.

Figure 22.2 Named entity types with examples.

with a location. Note that these signals include facts about the proper names as well as their surrounding contexts.

The notion of a named entity is commonly extended to include things that aren't entities per se, but nevertheless have practical importance and do have characteristic signatures that signal their presence; examples include dates, times, named events and other kinds of **temporal expressions**, as well as measurements, counts, prices and other kinds of **numerical expressions**. We'll consider some of these later in Sec. 22.3.

TEMPORAL EXPRESSIONS

NUMERICAL EXPRESSIONS

Let's revisit the sample text introduced earlier with the named entities marked (with TIME and MONEY used to to mark the temporal and monetary expressions).

> Citing high fuel prices, [$_{ORG}$ United Airlines] said [$_{TIME}$ Friday] it has increased fares by [$_{MONEY}$ \$6] per round trip on flights to some cities also served by lower-cost carriers. [$_{ORG}$ American Airlines], a unit of [$_{ORG}$ AMR Corp.], immediately matched the move, spokesman [$_{PERS}$ Tim Wagner] said. [$_{ORG}$ United], a unit of [$_{ORG}$ UAL Corp.], said the increase took effect [$_{TIME}$ Thursday] and applies to most routes where it competes against discount carriers, such as [$_{LOC}$ Chicago] to [$_{LOC}$ Dallas] and [$_{LOC}$ Denver] to [$_{LOC}$ San Francisco].

Name	Possible Categories
Washington	Person, Location, Political Entity, Organization, Facility
Downing St.	Location, Organization
IRA	Person, Organization, Monetary Instrument
Louis Vuitton	Person, Organization, Commercial Product

Figure 22.3 Common categorical ambiguities associated with various proper names.

[$_{PERS}$ Washington] was born into slavery on the farm of James Burroughs.
[$_{ORG}$ Washington] went up 2 games to 1 in the four-game series.
Blair arrived in [$_{LOC}$ Washington] for what may well be his last state visit.
In June, [$_{GPE}$ Washington] passed a primary seatbelt law.
The [$_{FAC}$ Washington] had proved to be a leaky ship, every passage I made...

Figure 22.4 Examples of type ambiguities in the use of the name *Washington*.

As shown, this text contains 13 mentions of named entities including 5 organizations, 4 locations, 2 times, 1 person, and 1 mention of money. The 5 organizational mentions correspond to 4 unique organizations, since *United* and *United Airlines* are distinct mentions that refer to the same entity.

22.1.1 Ambiguity in Named Entity Recognition

Named entity recognition systems face two types of ambiguity. The first arises from the fact the same name can refer to different entities of the same type. For example, *JFK* can refer to the former president or his son. This is basically a reference resolution problem and approaches to resolving this kind of ambiguity are discussed in Ch. 21.

The second source of ambiguity arises from the fact that identical named entity mentions can refer to entities of completely different types. For example, in addition to people, *JFK* might refer to the airport in New York, or to any number of schools, bridges and streets around the United States. Some examples of this kind of cross-type confusion are given in Figures 22.3 and 22.4.

Notice that some of the ambiguities shown in Fig. 22.3 are completely coincidental. There is no relationship between the financial and organizational uses of the name *IRA* — they simply arose coincidentally as acronyms from different sources (*Individual Retirement Account* and *International Reading Association*). On the other hand, the organizational uses of *Washington* and *Downing St.* are examples of a LOCATION-FOR-ORGANIZATION **metonymy**, as discussed in Ch. 19.

22.1.2 NER as Sequence Labeling

The standard way to approach the problem of named entity recognition is as a word-by-word sequence labeling task, where the assigned tags capture both the boundary and the type of any detected named entities. Viewed in this light, named entity recognition looks very much like the problem of syntactic base-phrase chunking. In fact, the dominant approach to NER is based on the same statistical sequence labeling techniques introduced in Ch. 5 for part of speech tagging and Ch. 13 for syntactic chunking.

In the sequence labeling approach to NER, classifiers are trained to label the tokens in a text with tags that indicate the presence of particular kinds of named entities. This approach makes use of the same style of IOB encoding employed for syntactic chunking. Recall that in this scheme an I is used to label tokens *inside* of a chunk, B is used to mark the beginning of a chunk, and O labels tokens outside any chunk of interest. Consider the following sentence from our running example.

(22.1) [$_{ORG}$ American Airlines], a unit of [$_{ORG}$ AMR Corp.], immediately matched the move, spokesman [$_{PERS}$ Tim Wagner] said.

This bracketing notation provides us with the extent and the type of the named entities in this text. Fig. 22.5 shows a standard word-by-word IOB-style tagging that captures the same information. As with syntactic chunking, the tagset for such an encoding consists of 2 tags for each entity type being recognized, plus 1 for the O tag outside any entity, or $(2 \times N) + 1$ tags.

Having encoded our training data with IOB tags, the next step is to select a set of features to associate with each input example (i.e. each of the tokens to be labeled in Fig. 22.5). These features should be plausible predictors of the class label and should be easily and reliably extractable from the source text. Recall that such features can be based not only on characteristics of the token to be classified, but also on the text in a surrounding window as well.

Fig. 22.6 gives a list of standard features employed in state-of-the-art named entity recognition systems. We've seen many of these features before in the context of part-of-speech tagging and syntactic base-phrase chunking. Several, however, are particularly important in the context of NER. The **shape** feature includes the usual upper case, lower case and capitalized forms, as well as more elaborate patterns designed to capture expressions that make use of numbers (*A9*), punctuation (*Yahoo!*) and atypical case alternations (*eBay*). It turns out that this feature by itself accounts for a considerable part of the success of NER systems for English news text. And as we'll see in Sec. 22.5, shape features are also particularly important in recognizing names of proteins and genes in biological texts. Fig. 22.7 describes some commonly employed shape feature values. SHAPE

The **presence in a named entity list** feature can be very predictive. Extensive

Words	Label
American	B_{ORG}
Airlines	I_{ORG}
,	O
a	O
unit	O
of	O
AMR	B_{ORG}
Corp.	I_{ORG}
,	O
immediately	O
matched	O
the	O
move	O
,	O
spokesman	O
Tim	B_{PERS}
Wagner	I_{PERS}
said	O
.	O

Figure 22.5 IOB encoding for a sample sentence.

GAZETTEERS

lists of names for all manner of things are available from both publicly available and commercial sources. Lists of place names, called **gazetteers**, contain millions of entries for all manner of locations along with detailed geographical, geologic and political information.[1] The United States Census Bureau provides extensive lists of first names and surnames derived from its decadal census in the U.S.[2] Similar lists of corporations, commercial products, and all manner of things biological and mineral are also available from a variety of sources.

This feature is typically implemented as a binary vector with a bit for each available kind of name list. Unfortunately, such lists can be difficult to create and maintain, and their usefulness varies considerably based on the named entity class. It appears that gazetteers can be quite effective, while extensive lists of persons and organizations are not nearly as beneficial (Mikheev et al., 1999).

Finally, features based on the presence of **predictive words and N-grams** in the context window can also be very informative. When they are present, preceding

[1] www.geonames.org

[2] www.census.gov

Feature	Explanation
Lexical items	The token to be labeled
Stemmed lexical items	Stemmed version of the target token
Shape	The orthographic pattern of the target word
Character affixes	Character level affixes of the target and surrounding words
Part of speech	Part of speech of the word
Syntactic chunk labels	Base phrase chunk label
Gazetteer or name list	Presence of the word in one or more named entity lists
Predictive token(s)	Presence of predictive words in surrounding text
Bag of words/Bag of N-grams	Words and/or N-grams occurring in the surrounding context.

Figure 22.6 Features commonly used in training named entity recognition systems.

Shape	Example
Lower	cummings
Capitalized	Washington
All caps	IRA
Mixed case	eBay
Capitalized initial with period	H.
Ends in digit	A9
Contains hyphen	H-P

Figure 22.7 Selected shape features.

and following titles, honorifics, and other markers such as *Rev.*, *MD* and *Inc.* can accurately indicate the class of an entity. Unlike name lists and gazetteers, these lists are relatively short and stable over time and are therefore easy to develop and maintain.

The relative usefulness of any of these features, or combination of features, depends to a great extent on the application, genre, media, language and text encoding. For example, shape features, which are critical for English newswire texts, are of little use with materials transcribed from spoken text via automatic speech recognition, materials gleaned from informally edited sources such as blogs and discussion forums, and for character-based languages like Chinese where case information isn't available. The set of features given in Fig. 22.6 should therefore be thought of as only a starting point for any given application.

Once an adequate set of features has been developed, they are extracted from a representative training set and encoded in a form appropriate to train a machine learning-based sequence classifier. A standard way of encoding these features is to simply augment our earlier IOB scheme with more columns. Fig. 22.8 illustrates the result of adding part-of-speech tags, syntactic base-phrase chunk tags, and shape

Features				Label
American	NNP	B$_{NP}$	cap	B$_{ORG}$
Airlines	NNPS	I$_{NP}$	cap	I$_{ORG}$
,	PUNC	O	punc	O
a	DT	B$_{NP}$	lower	O
unit	NN	I$_{NP}$	lower	O
of	IN	B$_{PP}$	lower	O
AMR	NNP	B$_{NP}$	upper	B$_{ORG}$
Corp.	NNP	I$_{NP}$	cap_punc	I$_{ORG}$
,	PUNC	O	punc	O
immediately	RB	B$_{ADVP}$	lower	O
matched	VBD	B$_{VP}$	lower	O
the	DT	B$_{NP}$	lower	O
move	NN	I$_{NP}$	lower	O
,	PUNC	O	punc	O
spokesman	NN	B$_{NP}$	lower	O
Tim	NNP	I$_{NP}$	cap	B$_{PER}$
Wagner	NNP	I$_{NP}$	cap	I$_{PER}$
said	VBD	B$_{VP}$	lower	O
.	PUNC	O	punc	O

Figure 22.8 Simple word-by-word feature encoding for NER.

information to our earlier example.

Given such a training set, a sequential classifier can be trained to label new sentences. As with part-of-speech tagging and syntactic chunking, this problem can be cast either as Markov-style optimization using HMMs or MEMMs as described in Ch. 6, or as a multi-way classification task deployed as a sliding-window labeler as described in Ch. 13. Figure Fig. 22.9 illustrates the operation of such a sequence labeler at the point where the token *Corp.* is next to be labeled. If we assume a context window that includes the 2 preceding and following words, then the features available to the classifier are those shown in the boxed area. Fig. 22.10 summarizes the overall sequence labeling approach to creating a NER system.

22.1.3 Evaluating Named Entity Recognition

The familiar metrics of **recall**, **precision** and F_1 **measure** introduced in Ch. 13 are used to evaluate NER systems. Recall that recall is the ratio of the number of correctly labeled responses to the total that should have been labeled; precision is the ratio of the number of correctly labeled responses to the total labeled. The

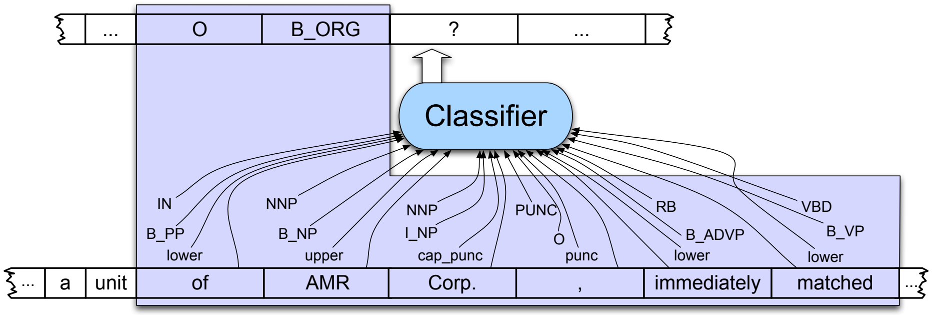


Figure 22.9 Named entity recognition as sequence labeling. The features available to the classifier during training and classification are those in the boxed area.

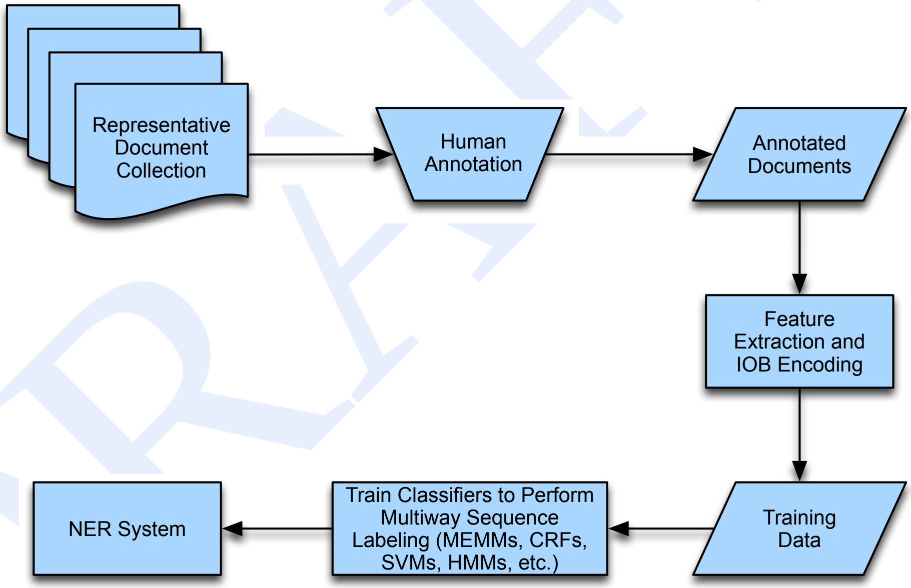

Representative Document Collection
Human Annotation
Annotated Documents
Feature Extraction and IOB Encoding
Training Data
Train Classifiers to Perform Multiway Sequence Labeling (MEMMs, CRFs, SVMs, HMMs, etc.)
NER System

Figure 22.10 Basic steps in the statistical sequence labeling approach to creating a named entity recognition system.

F-measure (van Rijsbergen, 1975) provides a way to combine these two measures into a single metric. The F-measure is defined as:

$$F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P+R} \tag{22.2}$$

The β parameter is used to differentially weight the importance of recall and precision, based perhaps on the needs of an application. Values of $\beta > 1$ favor recall, while values of $\beta < 1$ favor precision. When $\beta = 1$, precision and recall are equally balanced; this is sometimes called $F_{\beta=1}$ or just F_1:

$$F_1 = \frac{2PR}{P+R} \tag{22.3}$$

As with syntactic chunking, it is important to distinguish the metrics used to measure performance at the application level from those used during training. At the application level, recall and precision are measured with respect to the actual named entities detected. On the other hand, with an IOB encoding scheme the learning algorithms are attempting to optimize performance at the tag level. Performance at these two levels can be quite different; since the vast majority of tags in any given text are outside any entity, simply emitting an O tag for every token gives fairly high tag-level performance.

High-performing systems at recent standardized evaluations have entity level F-measures around .92 for PERSONS and LOCATIONS, and around .84 for ORGANIZATIONS (Sang and De Meulder, 2003).

22.1.4 Practical NER Architectures

Commercial approaches to NER are often based on pragmatic combinations of lists, rules and supervised machine learning(Jackson and Moulinier, 2002). One common approach is to make repeated passes over a text allowing the results of one pass to influence the next. The stages typically first involve the use of rules that have extremely high precision but low recall. Subsequent stages employ more error-prone statistical methods that take the output of the first pass into account.

1. First use high-precision rules to tag unambiguous entity mentions;
2. Then search for sub-string matches of the previously detected names using probabilistic string matching metrics (as described in Ch. 19).
3. Consult application-specific name lists to identify likely name entity mentions from the given domain.
4. Finally, apply probabilistic sequence labeling techniques that make use of the tags from previous stages as additional features.

The intuition behind this staged approach is two-fold. First, some of the entity mentions in a text will be more clearly indicative of a given entity's class than others. Second, once an unambiguous entity mention is introduced into a text, it is likely that subsequent shortened versions will refer to the same entity (and thus the same type of entity).

Relations		Examples	Types
Affiliations			
	Personal	*married to, mother of*	PER → PER
	Organizational	*spokesman for, president of*	PER → ORG
	Artifactual	*owns, invented, produces*	(PER | ORG) → ART
Geospatial			
	Proximity	*near, on outskirts*	LOC → LOC
	Directional	*southeast of*	LOC → LOC
Part-Of			
	Organizational	*a unit of, parent of*	ORG → ORG
	Political	*annexed, acquired*	GPE → GPE

Figure 22.11 Typical semantic relations with examples and the named entity types they involve.

22.2 Relation Detection and Classification

Next on our list of tasks is the ability to discern the relationships that exist among the entities detected in a text. To see what this means, let's return to our sample airline text with all the entities marked.

> Citing high fuel prices, [$_{ORG}$ United Airlines] said [$_{TIME}$ Friday] it has increased fares by [$_{MONEY}$ \$6] per round trip on flights to some cities also served by lower-cost carriers. [$_{ORG}$ American Airlines], a unit of [$_{ORG}$ AMR Corp.], immediately matched the move, spokesman [$_{PERS}$ Tim Wagner] said. [$_{ORG}$ United], a unit of [$_{ORG}$ UAL Corp.], said the increase took effect [$_{TIME}$ Thursday] and applies to most routes where it competes against discount carriers, such as [$_{LOC}$ Chicago] to [$_{LOC}$ Dallas] and [$_{LOC}$ Denver] to [$_{LOC}$ San Francisco].

This text stipulates a set of relations among the named entities mentioned within it. We know, for example, that *Tim Wagner* is a spokesman for *American Airlines*, that *United* is a unit of *UAL Corp.*, and that *American* is a unit of *AMR*. These are all binary relations that can be seen as instances of more generic relations such as **part-of** or **employs** that occur with fairly high frequency in news-style texts. Fig. 22.11 shows a list of generic relations of the kind used in recent standardized evaluations.[3] More domain-specific relations that might be extracted include the notion of an airline route. For example, from this text we can conclude that United has routes to Chicago, Dallas, Denver and San Francisco.

These relations correspond nicely to the model-theoretic notions we introduced in Ch. 17 to ground the meanings of the logical forms. That is, a relation

[3] http://www.nist.gov/speech/tests/ace/

Domain	$\mathscr{D} = \{a,b,c,d,e,f,g,h,i\}$
United, UAL, American Airlines, AMR	a,b,c,d
Tim Wagner	e
Chicago, Dallas, Denver, and San Francisco	f,g,h,i
Classes	
United, UAL, American and AMR are organizations	$Org = \{a,b,c,d\}$
Tim Wagner is a person	$Pers = \{e\}$
Chicago, Dallas, Denver and San Francisco are places	$Loc = \{f,g,h,i\}$
Relations	
United is a unit of UAL	$PartOf = \{\langle a,b\rangle, \langle c,d\rangle\}$
American is a unit of AMR	
Tim Wagner works for American Airlines	$OrgAff = \{\langle c,e\rangle\}$
United serves Chicago, Dallas, Denver and San Francisco	$Serves = \{\langle a,f\rangle, \langle a,g\rangle, \langle a,h\rangle, \langle a,i\rangle\}$

Figure 22.12 A model-based view of the relations and entities in our sample text.

consists of set of ordered tuples over elements of a domain. In most standard information extraction applications, the domain elements correspond either to the named entities that occur in the text, to the underlying entities that result from co-reference resolution, or to entities selected from a domain ontology. Fig. 22.12 shows a model-based view of the set of entities and relations that can be extracted from our running example. Notice how this model-theoretic view subsumes the NER task as well; named entity recognition corresponds to the identification of a class of unary relations.

22.2.1 Supervised Learning Approaches to Relation Analysis

Supervised machine learning approaches to relation detection and classification follow a scheme that should be familiar by now. Texts are annotated with relations chosen from a small fixed set by human analysts. These annotated texts are then used to train systems to reproduce similar annotations on unseen texts. Such annotations indicate the text spans of the two arguments, the roles played by each argument and the type of the relation involved.

The most straightforward approach breaks the problem down into two subtasks: detecting when a relation is present between two entities and then classifying any detected relations. In the first stage, a classifier is trained to make a binary decision as to whether or not a given pair of named entities participate in a relation. Positive examples are extracted directly from the annotated corpus, while negative examples are generated from within-sentence entity pairs that are not annotated with a relation.

```
function FINDRELATIONS(words) returns relations

    relations ← nil
    entities ← FINDENTITIES(words)
    forall entity pairs $\langle e1, e2\rangle$ in entities do
        if RELATED?($e1, e2$)
            relations ← relations+CLASSIFYRELATION($e1, e2$)
```

Figure 22.13 Finding and classifying the relations among entities in a text.

In the second phase, a classifier is trained to label the relations that exist between candidate entity pairs. As discussed in Ch. 6, techniques such as decision trees, naive Bayes or MaxEnt handle multiclass labeling directly. Binary approaches based on discovering separating hyperplanes such as SVMs solve multiclass problems by employing a one-versus-all training paradigm. In this approach, a sets of classifiers are trained where each classifier is trained on one label as the positive class and all the other labels as the negative class. Final classification is performed by passing each instance to be labeled to all of the classifiers and then choosing the label from the classifier with the most confidence, or returning a rank ordering over the positively responding classifiers. Fig. 22.13 illustrates the basic approach for finding and classifying relations among the named entities within a discourse unit.

As with named entity recognition, the most important step in this process is to identify surface features that will be useful for relation classification (Zhou et al., 2005). The first source of information to consider are **features of the named entities** themselves.

- Named entity types of the two candidate arguments
- Concatenation of the two entity types
- Head words of the arguments
- Bag of words from each of the arguments

The next set of features are derived from **the words in the text** being examined. It is useful to think of these features as being extracted from three locations: the text between the two candidate arguments, a fixed window before the first argument, and a fixed window after the second argument. Given these locations, the following word-based features have proven to be useful.

- The bag of words and bag of bigrams between the entities
- Stemmed versions of the same
- Words and stems immediately preceding and following the entities

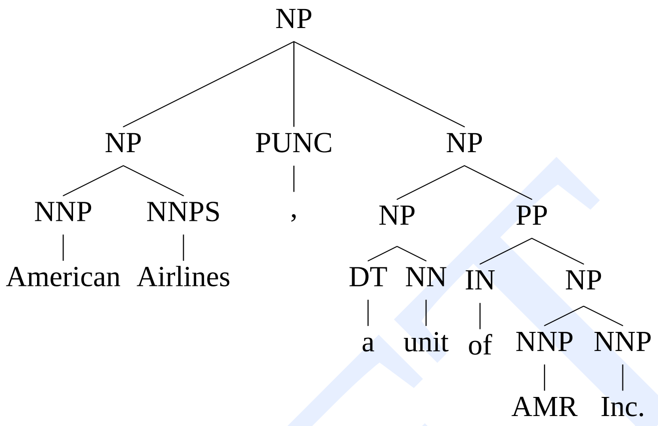


Figure 22.14 An appositive construction expressing an **a-part-of** relation.

- Distance in words between the arguments
- Number of entities between the arguments

Finally, **the syntactic structure** of a sentence can signal many of the relationships among any entities contained within it. The following features can be derived from various levels of syntactic analysis including base-phrase chunking, dependency parsing and full constituent parsing.

- Presence of particular constructions in a constituent structure
- Chunk base-phrase paths
- Bags of chunk heads
- Dependency-tree paths
- Constituent-tree paths
- Tree distance between the arguments

One method of exploiting parse trees is to create detectors that signal the presence of particular syntactic constructions and then associate binary features with those detectors. As an example of this, consider the sub-tree shown in Fig. 22.14 that dominates the named entities *American* and *AMR Inc.* The *NP* construction that dominates these two entities is called an appositive construction and is often associated with both **part-of** and **a-kind-of** relations in English. A binary feature indicating the presence of this construction can be useful in detecting these relations.

This method of feature extraction relies on a certain amount of a priori linguistic analysis to identify those syntactic constructions that may be useful predictors of certain classes. An alternative method is to automatically encode certain

Entity-based features	
Entity$_1$ type	ORG
Entity$_1$ head	*airlines*
Entity$_2$ type	PERS
Entity$_2$ head	*Wagner*
Concatenated types	ORGPERS
Word-based features	
Between-entity bag of words	{ *a, unit, of, AMR, Inc., immediately, matched, the, move, spokesman* }
Word(s) before Entity$_1$	NONE
Word(s) after Entity$_2$	*said*
Syntactic features	
Constituent path	$NP \uparrow NP \uparrow S \uparrow S \downarrow NP$
Base syntactic chunk path	$NP \rightarrow NP \rightarrow PP \rightarrow NP \rightarrow VP \rightarrow NP \rightarrow NP$
Typed-dependency path	$Airlines \leftarrow_{subj} matched \leftarrow_{comp} said \rightarrow_{subj} Wagner$

Figure 22.15 Sample of features extracted while classifying the <American Airlines, Tim Wagner> tuple.

aspects of tree structures as feature values and allow the machine learning algorithms to determine which values are informative for which classes. One simple and effective way to do this this involves the use of **syntactic paths** through trees. Consider again the tree discussed earlier that dominates *American Airlines* and *AMR Inc.* The syntactic relationship between these arguments can be characterized by the path traversed through the tree in getting from one to the other:

$$NP \uparrow NP \downarrow NP \downarrow PP \downarrow NP$$

Similar path features defined over syntactic dependency trees as well as flat base-phrase chunk structures have been shown to be useful for relation detection and classification (Culotta and Sorensen, 2004; Bunescu and Mooney, 2005). Recall that syntactic path features featured prominently in Ch. 20 in the context of semantic role labeling.

Fig. 22.15 illustrates some of the features that would be extracted while trying to classify the relationship between *American Airlines* and *Tim Wagner* from our example text.

22.2.2 Lightly Supervised Approaches to Relation Analysis

The supervised machine learning approach just described assumes that we have ready access to a large collection of previously annotated material with which to train classifiers. Unfortunately, this assumption is impractical in many real-world settings. A simple approach to extracting relational information without large amounts of annotated material is to use regular expression patterns to match text segments that are likely to contain expressions of the relations in which we are interested.

Consider the problem of building a table containing all the hub cities that various airlines utilize. Assuming we have access a search engine that permits some form of phrasal search with wildcards, we might try something like the following as a query:

```
/ * has a hub at * /
```

Given access to a reasonable amount of material of the right kind, such a search will yield a fair number of correct answers. A recent Google search using this pattern yields the following relevant sentences among the return set.

(22.4) Milwaukee-based Midwest has a hub at KCI.

(22.5) Delta has a hub at LaGuardia.

(22.6) Bulgaria Air has a hub at Sofia Airport, as does Hemus Air.

(22.7) American Airlines has a hub at the San Juan airport.

Of course, patterns such as this can fail in the two ways we discussed all the way back in Ch. 2: by finding some things they shouldn't, and by failing to find things they should. As an example of the first kind of error, consider the following sentences that were also included the earlier return set.

(22.8) airline j has a hub at airport k

(22.9) The catheter has a hub at the proximal end

(22.10) A star topology often has a hub at its center.

We can address these errors by making our proposed pattern more specific. In this case, replacing the unrestricted wildcard operator with a named entity class restriction would rule these examples out:

```
/[ORG] has a hub at [LOC]/
```

The second problem is that we can't know if we've found all the hubs for all airlines, since we've limited ourselves to this one rather specific pattern. Consider the following close calls missed by our first pattern.

(22.11) No frills rival easyJet, which has established a hub at Liverpool...

(22.12) Ryanair also has a continental hub at Charleroi airport (Belgium).

These examples are missed because they contain minor variations that cause the original pattern to fail. There are two ways to address this problem. The first is to generalize our pattern to capture expressions like these that contain the information we are seeking. This can be accomplished by relaxing the pattern to allow matches that skip parts of the candidate text. Of course, this approach is likely introduce more of the false positives that we tried to eliminate by making our pattern more specific in the first place.

The second, more promising solution, is to expand our set of specific high-precision patterns. Given a large and diverse document collection, an expanded set of patterns should be able to capture more of the information we're looking for. One way to acquire these additional patterns is to simply have human analysts familiar with the domain come up with more patterns and hope to get better coverage. A more interesting automatic alternative is to induce new patterns by **bootstrapping** from the initial search results from a small set of **seed patterns**.

BOOTSTRAPPING

SEED PATTERNS

To see how this works, let's assume that we've discovered that Ryanair has a hub at Charleroi. We can use this fact to discover new patterns by finding other mentions of this relation in our corpus. The simplest way to do this is to search for the terms *Ryanair*, *Charleroi* and *hub* in some proximity. The following are among the results from a recent search in Google News.

(22.13) Budget airline Ryanair, which uses Charleroi as a hub, scrapped all weekend flights out of the airport.

(22.14) All flights in and out of Ryanair's Belgian hub at Charleroi airport were grounded on Friday...

(22.15) A spokesman at Charleroi, a main hub for Ryanair, estimated that 8000 passengers had already been affected.

From these results, patterns such as the following can be extracted that look for relevant named entities of various types in the right places.

```
/ [ORG], which uses [LOC] as a hub /
/ [ORG]'s hub at [LOC] /
/ [LOC] a main hub for [ORG] /
```

These new patterns can then be used to search for additional tuples.

Fig. 22.16 illustrates the overall bootstrapping approach. This figure shows that the dual nature of patterns and seeds permits the process to start with either a small set of **seed tuples** or a set of **seed patterns**. This style of bootstrapping and pattern-based relation extraction is closely related to the techniques discussed in Ch. 20 for extracting hyponym and meronym-based lexical relations.

There are, of course, a fair number of technical details to be worked out to actually implement such an approach. The following are among some of the key problems.

- Representing the search patterns
- Assessing the accuracy and coverage of discovered patterns
- And assessing the reliability of the discovered tuples

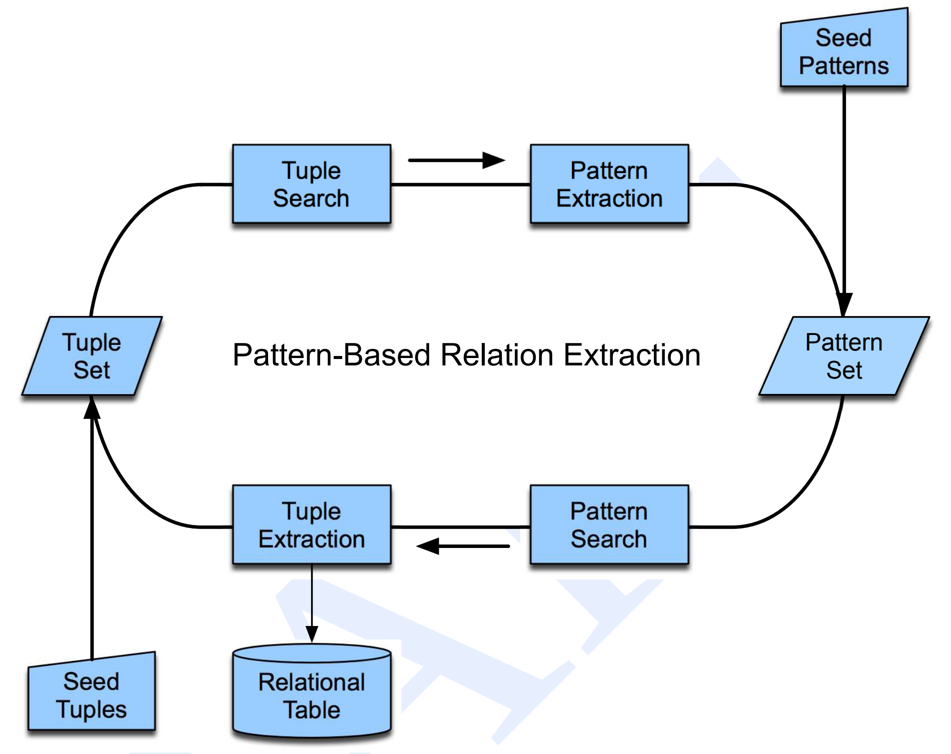


Figure 22.16 Pattern and bootstrapping-based relation extraction.

Patterns are typically represented in a way that captures the following four factors.

- Context prior to the first entity mention
- Context between the entity mentions
- Context following the second mention
- The order of the arguments in the pattern

Contexts are either captured as regular expression patterns or as vectors of features similar to those described earlier for machine learning-based approaches. In either case, they can be defined over character strings, word-level tokens, or syntactic and semantic structures. In general, regular expression approaches tend to be very specific, yielding high precision results; feature-based approaches, on the other hand, are more capable of ignoring potentially inconsequential elements of contexts.

Our next problem is how to assess the reliability of newly discovered patterns and tuples. Recall that we don't, in general, have access to annotated materials giving us the right answers. We therefore have to rely on the accuracy of the initial seed sets of patterns and/or tuples for gold-standard evaluation, and we have to ensure that we don't permit any significant **semantic drift** to occur as we're learning new patterns and tuples. Semantic drift occurs when an erroneous pattern leads to the introduction of erroneous tuples, which can then, turn, lead to the creation of problematic patterns.

SEMANTIC DRIFT

To see this consider the following example.

(22.16) Sydney has a ferry hub at Circular Quay.

If accepted as a positive example, this expression could lead to the introduction of the tuple $\langle Sydney, CircularQuay \rangle$. Patterns based on this tuple could propagate further errors into the database.

There are two factors that need to be balanced in assessing a proposed new pattern: the pattern's performance with respect to the current set of tuples, and the pattern's productivity in terms of the number of matches it produces in the document collection. More formally, given a document collection $\mathscr{D}$, a current set of tuples T, and a proposed pattern p, there are three factors that we need to track.

- *hits*: the set of tuples in T that p matches while looking in $\mathscr{D}$;
- *misses*: The set of tuples in T that p misses while looking at $\mathscr{D}$;
- $finds$: The total set of tuples that p finds in $\mathscr{D}$.

The following equation balances these considerations (Riloff and Jones, 1999).

(22.17)
$$Conf_{RlogF}(p) = \frac{hits_p}{hits_p + misses_p} \times log(finds_p)$$

It is useful to be able to treat this metric as a probability, so we'll need to normalize it. A simple way to do this is to track the range of confidences in a development set and divide by some previously observed maximum confidence (Agichtein and Gravano, 2000).

We can assess the confidence in a proposed new tuple by combining the evidence supporting it from all the patterns P' that match that tuple in $\mathscr{D}$ (Agichtein and Gravano, 2000). One way to combine such evidence is the **noisy-or** technique. Assume that a given tuple is supported by a subset of the patterns in P, each with its own confidence assessed as above. In the noisy-or model, we make two basic assumptions. First, that for a proposed tuple to be false, *all* of its supporting patterns must have been in error, and second that the sources of their individual failures are all independent. If we loosely treat our confidence measures as probabilities, then the probability of any individual pattern p failing is $1 - Conf(p)$; the probability of all of the supporting patterns for a tuple being wrong is the product of their individual failure probabilities, leaving us with the following equation for our confidence in a new tuple.

NOISY-OR

(22.18)
$$Conf(t) = 1 - \prod_{p \in P'} 1 - Conf(p)$$

The independence assumptions underlying the noisy-or model are very strong indeed. If the failure mode of the patterns are not independent, then the method will overestimate the confidence for the tuple. This overestimate is typically compensated for by setting a very high threshold for the acceptance of new tuples.

Given these measures, we can dynamically assess our confidence in both new tuples and patterns as the bootstrapping process iterates. Setting conservative thresholds for the acceptance of new patterns and tuples should help prevent the system from drifting from the targeted relation.

Although there have been no standardized evaluations for this style of relation extraction on publicly available sources, the technique has gained wide acceptance as a practical way to quickly populate relational tables from open source materials (most commonly from the Web) (Etzioni et al., 2005).

22.2.3 Evaluating Relation Analysis Systems

There are two separate methods for evaluating relation detection systems. In the first approach, the focus is on how well systems can find and classify all the relation mentions in a given text. In this approach, labeled and unlabeled recall, precision and F-measures are used to evaluate systems against a test collection with human annotated gold-standard relations. Labeled precision and recall requires the system to classify the relation correctly, while unlabeled methods simply measure a system's ability to detect entities that are related.

The second approach focuses on the tuples to be extracted from a body of text, rather than on the relation mentions. In this method, systems need not detect every mention of a relation to be scored correctly. Instead, the final evaluation is based on the set of tuples occupying the database when the system is finished. That is, we want to know if the system can discover that RyanAir has a hub at Charleroi; we don't really care how many times it discovers it.

This method has typically used to evaluate unsupervised methods of the kind discussed in the last section. In these evaluations human analysts simply examine the set of tuples produced by the system. Precision is simply the fraction of correct tuples out of all the tuples produced as judged by the human experts.

Recall remains a problem in this approach. It is obviously too costly to search by hand for all the relations that could have been extracted from a potentially large collection such as the Web. One solution is to compute recall at various levels of precision as described in Ch. 25 (Etzioni et al., 2005). Of course, this isn't true recall, since we're measuring against the number of correct tuples discovered rather than the number of tuples that are theoretically extractable from the text.

Another possibility is to evaluate recall on problems where large resources containing comprehensive lists of correct answers are available. Examples of include gazetteers for facts about locations, the Internet Movie Database (IMDB) for facts about movies or Amazon for facts about books. The problem with this approach is that it measures recall against a database that may be far more comprehensive than the text collections used by relation extraction system.

22.3 Temporal and Event Processing

Our focus thus far has been on extracting information about entities and their relations to one another. However, in most texts, entities are introduced in the course of describing the events in which they take part. Finding and analyzing the events in a text, and how they relate to each other in time, is crucial to extracting a more complete picture of the contents of a text. Such temporal information is particularly important in applications such as question answering and summarization.

In question answering, whether or not a system detects a correct answer may depend on temporal relations extracted from both the question and the potential answer text. As an example of this, consider the following sample question and potential answer text.

When did airlines as a group last raise fares?

> Last week, Delta boosted thousands of fares by $10 per round trip, and most big network rivals immediately matched the increase. (Dateline 7/2/2007).

This snippet does provide an answer to the question, but extracting it requires temporal reasoning to anchor the phrase *last week*, to link that time to the *boosting* event, and finally to link the time of the *matching* event to that.

The following sections introduce approaches to recognizing temporal expressions, figuring out the times that those expressions refer to, detecting events and associating times with those events.

22.3.1 Temporal Expression Recognition

Temporal expressions are those that refer to absolute points in time, relative times, durations and sets of these. **Absolute temporal expressions** are those that can be mapped directly to calendar dates, times of day, or both. **Relative temporal expressions** map to particular times via some other reference point (as in *a week from last Tuesday*.) Finally, **durations** denote spans of time at varying levels of granularity (seconds, minutes, days, weeks, centuries etc.) Fig. 22.17 provides some sample temporal expressions in each of these categories.

ABSOLUTE TEMPORAL EXPRESSIONS

RELATIVE TEMPORAL EXPRESSIONS

DURATIONS

Syntactically, temporal expressions are syntactic constructions that have temporal **lexical triggers** as their heads. In the annotation scheme in widest use, lex-

LEXICAL TRIGGERS

Absolute	Relative	Durations
April 24, 1916	yesterday	four hours
The summer of ’77	next semester	three weeks
10:15 AM	two weeks from yesterday	six days
The 3rd quarter of 2006	last quarter	the last three quarters

Figure 22.17 Examples of absolute, relation and durational temporal expressions.

Category	Examples
Noun	*morning, noon, night, winter, dusk, dawn*
Proper Noun	*January, Monday, Ides, Easter, Rosh Hashana, Ramadan, Tet*
Adjective	*recent, past, annual, former*
Adverb	*hourly, daily, monthly, yearly*

Figure 22.18 Examples of temporal lexical triggers.

ical triggers can be nouns, proper nouns, adjectives, and adverbs; full temporal expression consist of their phrasal projections: noun phrases, adjective phrases and adverbial phrases. Figure 22.18 provides examples of lexical triggers from these categories.

The annotation scheme in widest use is derived from the TIDES standard(Ferro et al., 2005). The approach presented here is based on the TimeML effort (Pustejovsky et al., 2005). TimeML provides an XML tag, TIMEX3, along with various attributes to that tag, for annotating temporal expressions. The following example illustrates the basic use of this scheme (ignoring the additional attributes, which we’ll discuss as needed later in Sec. 22.3.2).

> A fare increase initiated <TIMEX3>last week </TIMEX3> by UAL Corp’s United Airlines was matched by competitors over <TIMEX3>the weekend </TIMEX3>, marking the second successful fare increase in <TIMEX3>two weeks</TIMEX2>.

TEMPORAL EXPRESSION RECOGNITION

The **temporal expression recognition** task consists of finding the start and end of all of the text spans that correspond to such temporal expressions. Although there are myriad ways to compose time expressions in English, the set of temporal trigger terms is, for all practical purposes, static and the set of constructions used to generate temporal phrases is quite conventionalized. These facts suggest that any of the major approaches to finding and classifying text spans that we’ve already studied should be successful. The following three approaches have all been successfully employed in recent evaluations.

- Rule-based systems based on partial parsing or chunking
- Statistical sequence classifiers based on standard token-by-token IOB encod-

ing

- Constituent-based classification as used in semantic role labeling

Rule-based approaches to temporal expression recognition use cascades of automata to recognize patterns at increasing levels of complexity. Since temporal expressions are limited to a fixed set of standard syntactic categories, most of these systems make use of pattern-based methods for recognizing syntactic chunks. That is, tokens are first part-of-speech tagged and then larger and larger chunks are recognized using the results from previous stages. The only difference from the usual partial parsing approaches is the fact that temporal expressions must contain temporal lexical triggers. Patterns must, therefore, contain either specific trigger words (e.g. *February*), or patterns representing classes (e.g. *MONTH*). Fig. 22.19 illustrates this approach with a small representative fragment from a rule-based system written in Perl.

Sequence labeling approaches follow exactly the same scheme introduced in Ch. 13 for syntactic chunking. The three tags I, O and B are used to mark tokens that are either inside, outside or begin a temporal expression, as delimited by TIMEX3 tags. Example 22.3.1 would be labeled as follows in this scheme.

A fare increase initiated last week by UAL Corp's...
O O O O B I O O O

As expected, features are extracted from the context surrounding a token to be tagged and a statistical sequence labeler is trained using those features. As with syntactic chunking and named entity recognition, any of the usual statistical sequence methods can be applied. Fig. 22.20 lists the standard features used in the machine learning-based approach to temporal tagging.

Constituent-based methods combine aspects of both chunking and token-by-token labeling. In this approach, a complete constituent parse is produced by automatic means. The nodes in the resulting tree are then classified, one by one, as to whether they contain a temporal expression or not. This task is accomplished by training a binary classifier with annotated training data, using many of the same features employed in IOB-style training. This approach separates the classification problem from the segmentation problem by assigning the segmentation problem to the syntactic parser. The motivation for this choice was mentioned earlier; in currently available training materials, temporal expressions are limited to syntactic constituents from one of a fixed set of syntactic categories. Therefore, it makes sense to allow a syntactic parser to solve the segmentation part of the problem.

In standard evaluations, temporal expression recognizers are evaluated using the usual recall, precision and F-measures. In recent evaluations, both rule-based and statistical systems achieve about the same level of performance, with the best systems reaching an F-measure of around .87 on a strict exact match criteria. On a

```
# yesterday/today/tomorrow
$string =˜ s/(($OT+(early|earlier|later?)$CT+\s+)?(($OT+the$CT+\s+)?$OT+day$CT+\s+
$OT+(before|after)$CT+\s+)?$OT+$TERelDayExpr$CT+(\s+$OT+(morning|afternoon|evening|night)
$CT+)?)/<TIMEX2 TYPE=\"DATE\">$1<\/TIMEX2>/gio;

$string =˜ s/($OT+\w+$CT+\s+)
<TIMEX2 TYPE=\"DATE\"[ˆ>]*>($OT+(Today|Tonight)$CT+)<\/TIMEX2>/$1$2/gso;

# this/that (morning/afternoon/evening/night)
$string =˜ s/(($OT+(early|earlier|later?)$CT+\s+)?$OT+(this|that|every|the$CT+\s+
$OT+(next|previous|following))$CT+\s*$OT+(morning|afternoon|evening|night)
$CT+(\s+$OT+thereafter$CT+)?)/<TIMEX2 TYPE=\"DATE\">$1<\/TIMEX2>/gosi;
```

Figure 22.19 Fragment of Perl code from MITRE's TempEx temporal tagging system.

Feature	Explanation
Token	The target token to be labeled
Tokens in window	Bag of tokens in the window around a target
Shape	Character shape features
POS	Parts of speech of target and window words
Chunk tags	Base-phrase chunk tag for target and words in a window
Lexical triggers	Presence in a list of temporal terms

Figure 22.20 Typical features used to train IOB style temporal expression taggers.

looser criterion based on overlap with gold standard temporal expressions, the best systems reach an F-measure of .94.[4]

The major difficulties for all of these approaches are achieving reasonable coverage, correctly identifying the extent of temporal expressions and dealing with expressions that trigger false positives. The problem of false positives arises from the use of temporal trigger words as parts of proper names. For example, all of the following examples are likely to cause false positives for either rule-based or statistical taggers.

(22.19) *1984* tells the story of Winston Smith and his degradation by the totalitarian state in which he lives.

(22.20) Edge is set to join Bono onstage to perform U2's classic *Sunday* Bloody *Sunday*.

(22.21) Black *September* tried to detonate three car bombs in New York City in March 1973.

[4] http://www.nist.gov/speech/tests/ace/

```
<TIMEX3 id=t1 type="DATE" value="2007-07-02" functionInDocument="CREATION_TIME">
July 2, 2007 </TIMEX3>  A fare increase initiated <TIMEX3 id="t2" type="DATE"
value="2007-W26"  anchorTimeID="t1">last week</TIMEX3> by UAL Corp’s United  Airlines was
matched by competitors over <TIMEX3 id="t3" type="DURATION" value="P1WE"
anchorTimeID="t1"> the weekend </TIMEX3>, marking the second successful fare increase in
<TIMEX3 id="t4" type="DURATION"  value="P2W" anchorTimeID="t1"> two weeks </TIMEX3>.
```

Figure 22.21 TimeML markup including normalized values for temporal expressions.

22.3.2 Temporal Normalization

TEMPORAL NORMALIZATION

The task of recognizing temporal expressions is typically followed by the task of normalization. **Temporal normalization** refers to the process of mapping a temporal expression to either a specific point in time, or to a duration. Points in time correspond either to calendar dates or to times of day (or both). Durations primarily consist of lengths of time, but may also include information concerning the start and end points of a duration when that information is available.

Normalized representations of temporal expressions are captured using the VALUE attribute from the ISO 8601 standard for encoding temporal values(ISO8601, 2004). To illustrate some aspects of this scheme, let’s return to our earlier example, reproduced in Fig. 22.21 with the value attributes added in.

FULLY QUALIFIED DATE

The dateline, or document date, for this text was *July 2, 2007*. The ISO representation for this kind of **fully qualified date** expression is YYYY-MM-DD, or in this case, 2007-07-02. The encodings for the temporal expressions in our sample text all follow from this date, and are shown here as values for the VALUE attribute. Let’s consider each of these temporal expressions in turn.

The first temporal expression in the text proper refers to a particular week of the year. In the ISO standard, weeks are numbered from 01 to 53, with the first week of the year being the one that has the first Thursday of the year. These weeks are represented using the template YYYY-Wnn. The ISO week for our document date is week 27, thus the value for *last week* is represented as “2007-W26”.

The next temporal expression is *the weekend*. ISO weeks begin on Monday, thus, weekends occur at the end of a week and are fully contained within a single week. Weekends are treated as durations, so the value of the VALUE attribute has to be a length. Durations are represented using the pattern P*nx*, where *n* is an integer denoting the length and *x* represents the unit, as in P3Y for *three years* or P2D for *two days*. In this example, one weekend is captured as P1WE. In this case, there is also sufficient information to anchor this particular weekend as part of a particular week. Such information is encoded in the ANCHORTIMEID attribute. Finally, the phrase *two weeks* also denotes a duration captured as P2W.

Unit	Pattern	Sample Value
Fully Specified Dates	YYYY-MM-DD	1991-09-28
Weeks	YYYY-nnW	2007-27W
Weekends	PnWE	P1WE
24 hour clock times	HH:MM:SS	11:13:45
Dates and Times	YYYY-MM-DDTHH:MM:SS	1991-09-28T11:00:00
Financial quarters	Qn	1999-3Q

Figure 22.22 Sample ISO patterns for representing various times and durations.

There is a lot more to both the ISO 8601 standard and the various temporal annotation standards — far too much to cover here. Fig. 22.22 describes some of the basic ways that other times and durations are represented. Consult (ISO8601, 2004; Ferro et al., 2005; Pustejovsky et al., 2005) for more details.

Most current approaches to temporal normalization employ rule-based methods that associate semantic analysis procedures with patterns matching particular temporal expressions. This is a domain-specific instantiation of the compositional rule-to-rule approach introduced in Ch. 18. In this approach, the meaning of a constituent is computed from the meaning of its parts, and the method used to perform this computation is specific to the constituent being created. The only difference here is that the semantic composition rules involve simple temporal arithmetic rather than λ-calculus attachments.

To normalize temporal expressions, we'll need rules for four kinds of expressions.

- Fully qualified temporal expressions
- Absolute temporal expressions
- Relative temporal expressions
- Durations

Fully qualified temporal expressions contain a year, month and day in some conventional form. The units in the expression must be detected and then placed in the correct place in the corresponding ISO pattern. The following pattern normalizes the fully-qualified temporal expression used in expressions like *April 24, 1916*.

$$FQTE \rightarrow Month\ Date\ ,\ Year \qquad \{Year.val\ -\ Month.val\ -\ Date.val\}$$

In this rule, the non-terminals *Month*, *Date*, and *Year* represent constituents that have already been recognized and assigned semantic values, accessed via the **.val* notation. The value of this FQE constituent can, in turn, be accessed as *FQTE.val* during further processing.

TEMPORAL ANCHOR

Fully qualified temporal expressions are fairly rare in real texts. Most temporal expressions in news articles are incomplete and are only implicitly anchored, often with respect to the dateline of the article, which we'll refer to as the document's **temporal anchor**. The values of relatively simple temporal expressions such as *today*, *yesterday*, or *tomorrow* can all be computed with respect to this temporal anchor. The semantic procedure for *today* simply assigns the anchor, while the attachments for *tomorrow* and *yesterday* add a day and subtract a day from the anchor, respectively. Of course, given the circular nature of our representations for months, weeks, days and times of day, our temporal arithmetic procedures must use modulo arithmetic appropriate to the time unit being used.

Unfortunately, even simple expressions such as *the weekend* or *Wednesday* introduce a fair amount of complexity. In our current example, *the weekend* clearly refers to the weekend of the week that immediately precedes the document date. But this won't always be the case, as is illustrated in the following example.

(22.22) Random security checks that began yesterday at Sky Harbor will continue at least through the weekend.

In this case, the expression *the weekend* refers to the weekend of the week that the anchoring date is part of (i.e. the coming weekend). The information that signals this comes from the tense of *continue*, the verb governing *the weekend*.

Relative temporal expressions are handled with temporal arithmetic similar to that used for *today* and *yesterday*. To illustrate this, consider the expression *last week* from our example. From the document date, we can determine that the ISO week for the article is week 27, so *last week* is simply 1 minus the current week.

Again, even simple constructions such as this can be ambiguous in English. The resolution of expressions involving *next* and *last* must take into account the distance from the anchoring date to the nearest unit in question. For example, a phrase such as *next Friday* can refer to either the immediately next Friday, or to the Friday following that. The determining factor has to do with the proximity to the reference time. The closer the document date is to a Friday, the more likely it is that the phrase *next Friday* will skip the nearest one. Such ambiguities are handled by encoding language and domain specific heuristics into the temporal attachments.

The need to associate highly idiosyncratic temporal procedures with particular temporal constructions accounts for the widespread use of of rule-based methods in temporal expression recognition. Even when high performance statistical methods are used for temporal recognition, rule-based patterns are still required for normalization. Although the construction of these patterns can be tedious and filled with exceptions, it appears that sets of patterns that provide good coverage in newswire domains can be created fairly quickly (Ahn et al., 2005).

Finally, many temporal expressions are anchored to events mentioned in a text and not directly to other temporal expressions. Consider the following example.

(22.23) One week after the storm, JetBlue issued its customer bill of rights.

To determine when JetBlue issued its customer bill of rights we need to determine the time of *the storm* event, and then that time needs to be modified by the temporal expression *one week after*. We'll return to this issue when we take up event detection in the next section.

22.3.3 Event Detection and Analysis

EVENT DETECTION AND CLASSIFICATION

The task of **event detection and classification** is to identify mentions of events in texts and then assign those events to a variety of classes. For the purposes of this task, an event mention is any expression denoting an event or state that can be assigned to a particular point, or interval, in time. The following markup of Example 22.3.1 shows all the events in this text.

> [$_{EVENT}$ Citing] high fuel prices, United Airlines [$_{EVENT}$ said] Friday it has [$_{EVENT}$ increased] fares by $6 per round trip on flights to some cities also served by lower-cost carriers. American Airlines, a unit of AMR Corp., immediately [$_{EVENT}$ matched] [$_{EVENT}$ the move], spokesman Tim Wagner [$_{EVENT}$ said]. United, a unit of UAL Corp., [$_{EVENT}$ said] [$_{EVENT}$ the increase] took effect Thursday and [$_{EVENT}$ applies] to most routes where it [$_{EVENT}$ competes] against discount carriers, such as Chicago to Dallas and Denver to San Francisco.

In English, most event mentions correspond to verbs, and most verbs introduce events. However, as we can see from our example this is not always the case. Events can be introduced by noun phrases, as in *the move* and *the increase*, and some verbs fail to introduce events, as in the phrasal verb *took effect*, which refers to when the event began rather than to the event itself. Similarly, light verbs such as *make*, *take*, and *have* often fail to denote events. In these cases, the verb is simply providing a syntactic structure for the arguments to an event expressed by the direct object as in *took a flight*.

Both rule-based and statistical machine learning approaches have been applied to the problem of event detection. Both approaches make use of surface information such as parts of speech information, presence of particular lexical items, and verb tense information. Fig. 22.23 illustrates the key features used in current event detection and classification systems.

Having detected both the events and the temporal expressions in a text, the next logical task is to use this information to fit the events into a complete timeline. Such a timeline would be useful for applications such as question answering

Feature	Explanation
Character affixes	Character-level prefixes and suffixes of target word
Nominalization suffix	Character level suffixes for nominalizations (eg. *-tion*)
Part of speech	Part of speech of the target word
Light verb	Binary feature indicating that the target is governed by a light verb
Subject syntactic category	Syntactic category of the subject of the sentence
Morphological stem	Stemmed version of the target word
Verb root	Root form of the verb basis for a nominalization
Wordnet hypernyms	Hypernym set for the target

Figure 22.23 Features commonly used in both rule-based and statistical approaches to event detection.

and summarization. This ambitious task is is the subject of considerable current research but is beyond the capabilities of current systems.

A somewhat simpler, but still useful, task is to impose a partial ordering on the events and temporal expressions mentioned in a text. Such an ordering can provide many of the same benefits as a true timeline. An example of such a partial ordering would be to determine that the fare increase by *American Airlines* came *after* the fare increase by *United* in our sample text. Determining such an ordering can be viewed as a binary relation detection and classification task similar to those described earlier in Sec. 22.2.

Current approaches to this problem attempt to identify a subset of Allen's 13 temporal relations discussed earlier in Ch. 17, and shown here in Fig. 22.24. Recent evaluation efforts have focused on detecting the *before*, *after* and *during* relations among the temporal expressions, document date and event mentions in a text (Verhagen et al., 2007). Most of the top-performing systems employ statistical classifiers, of the kind discussed earlier in Sec. 22.2, trained on the TimeBank corpus (Pustejovsky et al., 2003b).

22.3.4 TimeBank

TIMEBANK

As we've seen with other tasks, it's tremendously useful to have access to text annotated with the types and relations in which we're interested. Such resources facilitate both corpus-based linguistic research as well as the training of systems to perform automatic tagging. The **TimeBank** corpus consists of text annotated with much of the information we've been discussing throughout this section (Pustejovsky et al., 2003b). The current release (TimeBank 1.2) of the corpus consists of 183 news articles selected from a variety of sources, including the Penn TreeBank and PropBank collections.

Each article in the TimeBank corpus has had the temporal expressions and

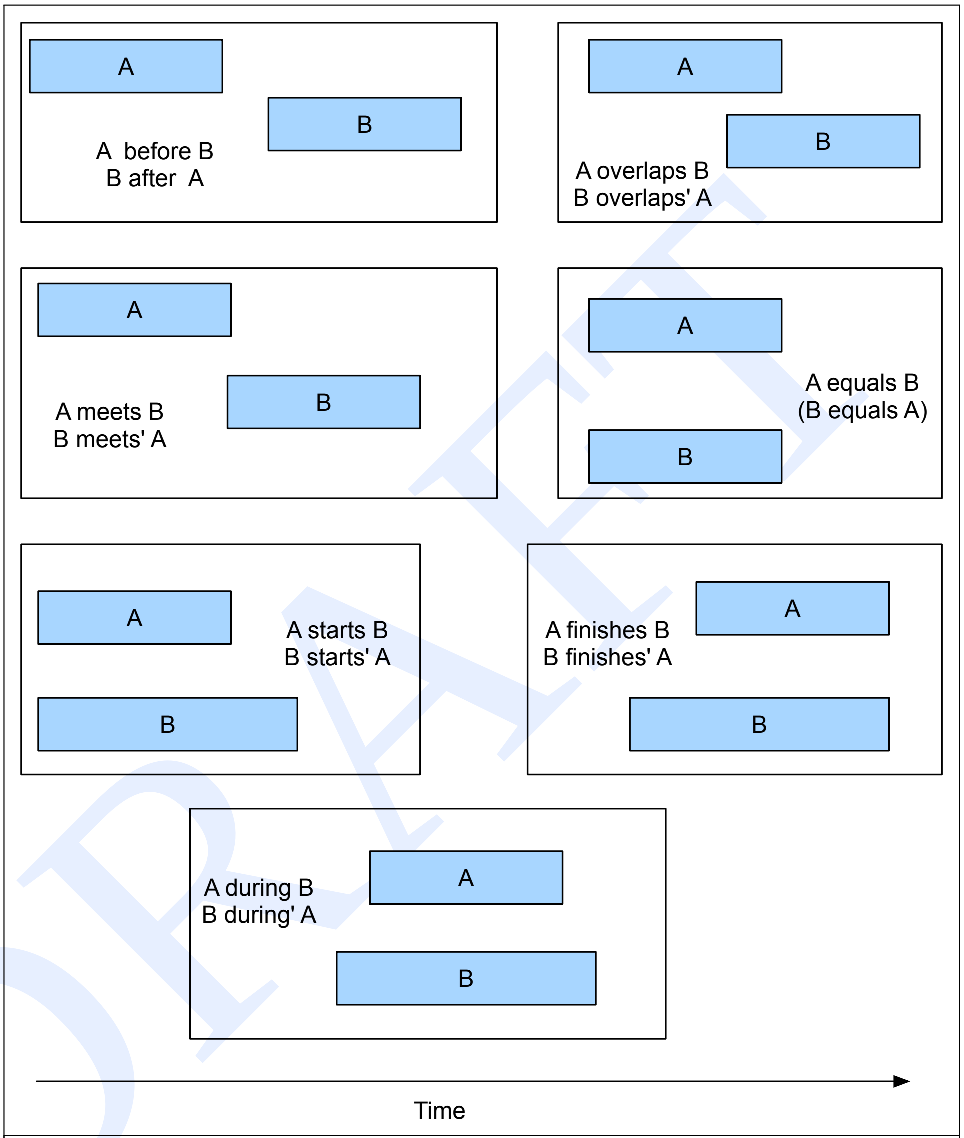


Figure 22.24 Allen's 13 possible temporal relations.

event mentions in them explicitly annotated in the TimeML annotation (Pustejovsky et al., 2003a). In addition to temporal expressions and events, the TimeML annotation provides temporal links between events and temporal expressions that specify the nature of the relation between them. Consider the following sample sentence and its corresponding markup shown in Fig. 22.25 selected from one of the TimeBank documents.

```
<TIMEX3 tid=''t57'' type="DATE'' value="1989-10-26"
functionInDocument="CREATION_TIME"> 10/26/89  </TIMEX3>

Delta Air Lines earnings <EVENT eid="e1" class="OCCURRENCE"> soared </EVENT>
33\% to a record in  <TIMEX3 tid="t58" type="DATE" value="1989-Q1" anchorTimeID="t57">
the fiscal first quarter </TIMEX3>, <EVENT eid="e3"  class="OCCURRENCE">bucking</EVENT>
the industry trend toward <EVENT eid="e4" class="OCCURRENCE">declining</EVENT> profits.
```

Figure 22.25 Example from the TimeBank corpus.

(22.24) Delta Air Lines soared 33% to a record in the fiscal first quarter, bucking the industry trend toward declining profits.

As annotated, this text includes three events and two temporal expressions. The events are all in the occurrence class and are given unique identifiers for use in further annotations. The temporal expressions include the creation time of the article, which serves as the document time, and a single temporal expression within the text.

In addition to these annotations, TimeBank provides 4 links that capture the temporal relations between the events and times in the text. The following are the within sentence temporal relations annotated for this example.

- Soaring$_{e1}$ is **included** in the fiscal first quarter$_{t58}$
- Soaring$_{e1}$ is **before** 1989-10-26$_{t57}$
- Soaring$_{e1}$ is **simultaneous** with the bucking$_{e3}$
- Declining$_{e4}$ **includes** soaring$_{e1}$

The set of 13 temporal relations used in TimeBank are based on Allen's (Allen, 1984) relations introduced earlier in Fig. 22.24.

22.4 Template-Filling

SCRIPTS

Many texts contain reports of events, and possibly sequences of events, that often correspond to fairly common, stereotypical situations in the world. These abstract situations can be characterized as **scripts**, in that they consist of prototypical sequences of sub-events, participants, roles and props (Schank and Abelson, 1977). The use of explicit representations of such scripts in language processing can assist in many of the IE tasks we've been discussing. In particular, the strong expectations provided by these scripts can facilitate the proper classification of entities, the assignment of entities into roles and relations, and most critically, the drawing of inferences that fill in things that have been left unsaid.

TEMPLATES

In their simplest form, such scripts can be represented as **templates** consisting of fixed sets of **slots** which take as values **slot-fillers** belonging to particular classes. The task of **template-filling** is to find documents that invoke particular scripts and then fill the slots in the associated templates with fillers extracted from the text. These slot-fillers may consist of text segments extracted directly from the text, or they may consist of concepts that have been inferred from text elements via some additional processing (times, amounts, entities from an ontology, etc.)

A filled template from our original airline story might look like the following.

$$\textsc{Fare-Raise Attempt:} \left[\begin{array}{ll} \textsc{Lead Airline:} & \textsc{United Airlines} \\ \textsc{Amount:} & \$6 \\ \textsc{Effective Date:} & \text{2006-10-26} \\ \textsc{Follower:} & \textsc{American Airlines} \end{array}\right]$$

Note that as is often the case, the slot-fillers in this example all correspond to detectable named entities of various kinds (organizations, amounts and times). This suggests that template-filling applications should rely on tags provided by named entity recognition, temporal expression and co-reference algorithms to identify candidate slot-fillers.

The next section describes a straightforward approach to filling slots using sequence labeling techniques. Sec. 22.4.2 then describes a system designed to address a considerably more complex template-filling task, based on the use of cascades of finite-state transducers.

22.4.1 Statistical Approaches to Template-Filling

A surprisingly effective approach to template-filling simply casts it as a statistical sequence labeling problem. In this approach, systems are trained to label sequences of tokens as potential fillers for particular slots. There are two basic ways to instantiate this approach: the first is to train separate sequence classifiers for each slot to be filled and then send the entire text through each labeler, the other is to train one large classifier (usually an HMM) that assigns labels for each of the slots to be recognized. We'll focus on the former approach here; we'll take up the single large classifier approach in Ch. 23.

Under the one classifier per slot approach, slots are filled with the text segments identified by each slot's corresponding classifier. As with the other IE tasks described earlier in this chapter, all manner of statistical sequence classifiers have been applied to this problem, all using the usual set of features: tokens, shapes of tokens, part-of-speech tags, syntactic chunk tags, and named entity tags.

There is the possibility in this approach that multiple non-identical text segments will be labeled with the same slot label. This situation can arise in two ways:

from competing segments that refer to the same entity using different referring expressions, or from competing segments that represent truly distinct hypotheses. In our sample text, we might expect the segments *United*, *United Airlines* to be labeled as the LEAD AIRLINE. These are not incompatible choices and the reference resolution techniques introduced in Ch. 21 can provide a path to a solution.

Truly competing hypotheses arise when a text contains multiple entities of the expected type for a given slot. In our example, *United Airlines* and *American Airlines* are both airlines and it is possible for both to be tagged as LEAD AIRLINE based on their similarity to exemplars in the training data. In general, most systems simply choose the hypothesis with the highest confidence. Of course, the implementation of this confidence heuristic is dependent on the style of sequence classifier being employed. Markov-based approaches simply select the segment with the highest probability labeling (Freitag and McCallum, 1999).

A variety of annotated collections have been used to evaluate this style of approach to template-filling, including sets of job announcements, conference calls for papers, restaurant guides and biological texts. A frequently employed collection is the CMU Seminar Announcement Corpus[5], a collection of 485 seminar announcements retrieved from the Web with slots annotated for the SPEAKER, LOCATION, START TIME and END TIME. State-of-the-art F-measures on this dataset range from around .98 for the start and end time slots, to as high as .77 for the speaker slot (Roth and tau Yih, 2001; Peshkin and Pfefer, 2003).

As impressive as these results are, they are due as much to the constrained nature of the task as to the techniques they have been employed. Three strong task constraints have contributed to this success. First, in most evaluations all the documents in the collection are all relevant and homogeneous, that is they are known to contain the slots of interest. Second, the documents are all relatively small, providing little room for distractor segments that might incorrectly fill slots. And finally, the target output consists solely of a small set of slots which are to be filled with snippets from the text itself.

22.4.2 Finite-State Template-Filling Systems

The tasks introduced in the *Message Understanding Conferences* (MUC) (Sundheim, 1993), a series of U.S. Government-organized information extraction evaluations, represent a considerably more complex template-filling problem. Consider the following sentences selected from the MUC-5 materials from Grishman and Sundheim (1995).

[5] http://www.isi.edu/info-agents/RISE/

TIE-UP-1:	
RELATIONSHIP:	TIE-UP
ENTITIES:	"Bridgestone Sports Co." "a local concern" "a Japanese trading house"
JOINTVENTURECOMPANY	"Bridgestone Sports Taiwan Co."
ACTIVITY	ACTIVITY-1
AMOUNT	NT$20000000
ACTIVITY-1:	
COMPANY	"Bridgestone Sports Taiwan Co."
PRODUCT	"iron and "metal wood" clubs"
STARTDATE	DURING: January 1990

Figure 22.26 The templates produced by the FASTUS (Hobbs et al., 1997) information extraction engine given the input text on page 35.

> Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.
>
> The joint venture, Bridgestone Sports Taiwan Co., capitalized at 20 million new Taiwan dollars, will start production in January 1990 with production of 20,000 iron and "metal wood" clubs a month.

The MUC-5 evaluation task required systems to produce hierarchically linked templates describing the participants in the joint venture, the resulting company, and its intended activity, ownership and capitalization. Fig. 22.26 shows the resulting structure produced by the FASTUS system (Hobbs et al., 1997). Note how the filler of the ACTIVITY slot of the TIE-UP template is itself a template with slots to be filled.

The FASTUS system produces the template given above, based on a cascade of transducers in which each level of linguistic processing extracts some information from the text, which is passed on to the next higher level, as shown in Figure 22.27

Most systems base most of these levels on finite-automata, although in practice most complete systems are not technically finite-state, either because the individual automata are augmented with feature registers (as in FASTUS), or because they are used only as preprocessing steps for full parsers (e.g., Gaizauskas et al., 1995; Weischedel, 1995) , or are combined with other components based on statistical methods (Fisher et al., 1995).

Let's sketch the FASTUS implementation of each of these levels, following Hobbs et al. (1997) and Appelt et al. (1995). After tokenization, the second level recognizes multiwords like *set up*, and *joint venture*, and names like *Bridgestone*

No.	Step	Description
1	Tokens:	Transfer an input stream of characters into a token sequence.
2	Complex Words:	Recognize multi-word phrases, numbers, and proper names.
3	Basic phrases:	Segment sentences into noun groups, verb groups, and particles.
4	Complex phrases:	Identify complex noun groups and complex verb groups.
5	Semantic Patterns:	Identify semantic entities and events and insert into templates.
6	Merging:	Merge references to the same entity or event from different parts of the text.

Figure 22.27 Levels of processing in FASTUS (Hobbs et al., 1997). Each level extracts a specific type of information which is then passed on to the next higher level.

Sports Co.. The named entity recognizer is a transducer, composed of a large set of specific mappings designed to handle the usual set of named entities.

The following are typical rules for modeling names of performing organizations like *San Francisco Symphony Orchestra* and *Canadian Opera Company*. While the rules are written using a context-free syntax, there is no recursion and therefore they can be automatically compiled into finite-state transducers.

Performer-Org	→	(pre-location) Performer-Noun+ Perf-Org-Suffix
pre-location	→	locname | nationality
locname	→	city | region
Perf-Org-Suffix	→	orchestra, company
Performer-Noun	→	symphony, opera
nationality	→	Canadian, American, Mexican
city	→	San Francisco, London

The second stage also might transduce sequences like *forty two* into the appropriate numeric value (recall the discussion of this problem in Ch. 8).

The third FASTUS stage implements chunking and produces a sequence of basic syntactic chunks, such as noun groups, verb groups, and so on, using finite-state rules of the sort discussed in Ch. 13.

The output of the FASTUS basic phrase identifier is shown in Figure 22.28; note the use of some domain-specific basic phrases like *Company* and *Location*.

Recall that Ch. 13 described how these basic phrases can be combined into more complex noun groups and verb groups. This is accomplished in Stage 4 of

Company	Bridgestone Sports Co.
Verb Group	said
Noun Group	Friday
Noun Group	it
Verb Group	had set up
Noun Group	a joint venture
Preposition	in
Location	Taiwan
Preposition	with
Noun Group	a local concern
Conjunction	and
Noun Group	a Japanese trading house
Verb Group	to produce
Noun Group	golf clubs
Verb Group	to be shipped
Preposition	to
Location	Japan

Figure 22.28 The output of Stage 2 of the FASTUS basic-phrase extractor, which uses finite-state rules of the sort described by Appelt and Israel (1997) and shown on page **??**.

(1)	RELATIONSHIP:	TIE-UP
	ENTITIES:	“Bridgestone Sports Co.” “a local concern” “a Japanese trading house”
(2)	ACTIVITY:	PRODUCTION
	PRODUCT	“golf clubs”
(3)	RELATIONSHIP:	TIE-UP
	JOINTVENTURECOMPANY:	“Bridgestone Sports Taiwan Co.”
	AMOUNT:	NT$20000000
(4)	ACTIVITY:	PRODUCTION
	COMPANY:	“Bridgestone Sports Taiwan Co.”
	STARTDATE	DURING: January 1990
(5)	ACTIVITY	PRODUCTION
	PRODUCT	“iron and “metal wood” clubs”

Figure 22.29 The five partial templates produced by Stage 5 of the FASTUS system. These templates will be merged by the Stage 6 merging algorithm to produce the final template shown in Fig. 22.26 on page 36.

FASTUS, by dealing with conjunction and with the attachment of measure phrases as in the following.

20,000 iron and “metal wood” clubs a month, and prepositional phrases:

production of 20,000 iron and “metal wood” clubs a month,

The output of Stage 4 is a list of complex noun groups and verb groups. Stage 5 takes this list, ignoring all input that has not been chunked into a complex group, recognizes entities and events in the complex groups, and inserts the recognized objects into the appropriate slots in templates. The recognition of entities and events is done by hand-coded finite-state automata whose transitions are based on particular complex-phrase types annotated by particular head words or particular features like *company*, *currency*, or *date*.

As an example, the first sentence of the news story above realizes the semantic patterns based on the following two regular expressions (where NG indicates Noun-Group and VG Verb-Group).

- NG(Company/ies) VG(Set-up) NG(Joint-Venture) with NG(Company/ies)
- VG(Produce) NG(Product)

The second sentence realizes the second pattern above as well as the following two patterns:

- NG(Company) VG-Passive(Capitalized) at NG(Currency)
- NG(Company) VG(Start) NG(Activity) in/on NG(Date)

The result of processing these two sentences is the set of five draft templates shown in Fig. 22.29. These five templates must then be merged into the single hierarchical structure shown in Fig. 22.26. The merging algorithm decides whether two activity or relationship structures are sufficiently consistent that they might be describing the same events, and merges them if so. The merging algorithm must also perform reference resolution as described in Ch. 21.

22.5 Advanced: Biomedical Information Extraction [*]

Information extraction from biomedical journal articles has become an important application area in recent years. The motivation for this work comes primarily from biologists, who find themselves faced with an enormous increase in the number of publications in their field since the advent of modern genomics — so many that keeping up with the relevant literature is nearly impossible for many scientists. Fig. 22.30 amply demonstrates the severity of the problem faced by these scientists. Clearly, applications that can automate the extraction and aggregation of useful information from such sources would be a boon to researchers.

[*] This section was written by K. Bretonnel Cohen

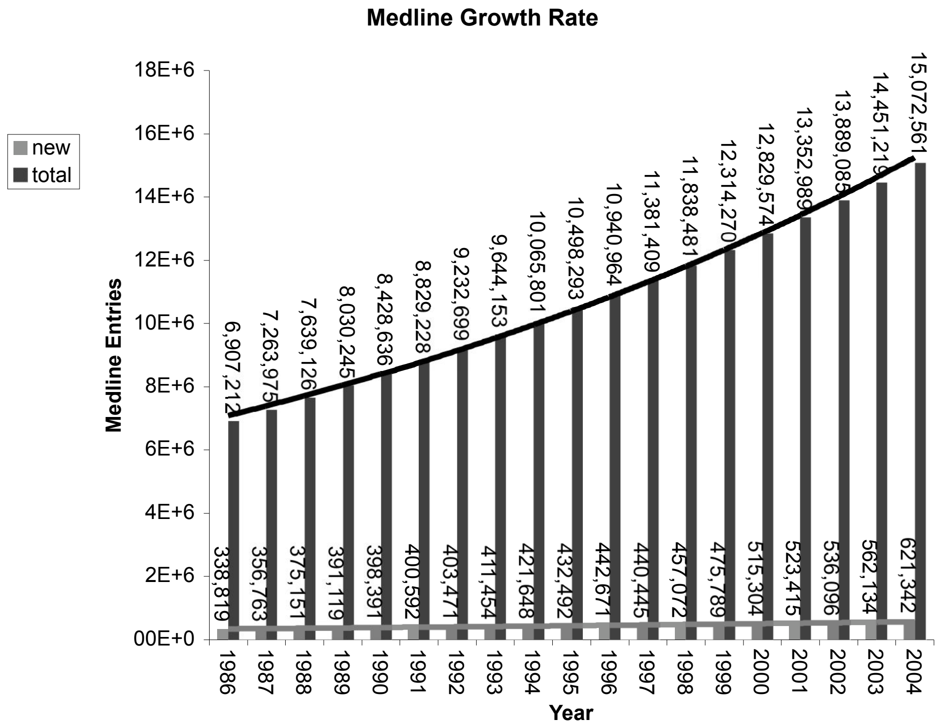


Figure 22.30 Exponential growth in number of articles available in the PubMed database from 1986 to 2004 (after (Cohen and Hunter, 2004)).

A growing application area for information extraction in the biomedical domain is as an aid to the construction of large databases of genomic and related information. Without the availability of information extraction-based curator assistance tools, many manual database construction efforts will not be complete for decades — a time-span much too long to be useful (Jr. et al., 2007).

A good example of this kind of application is the MuteXt system. This system targets two named entity types — mutations in proteins and two very specific types of proteins called *G-coupled protein receptors and nuclear hormone receptors*. MuteXt was used to build a database that drew information from 2,008 documents; building it by hand would have taken an enormously time-consuming and expensive undertaking. Mutations in G-protein coupled receptors are associated with a range of diseases that includes diabetes, ocular albinism, and retinitis pigmentosa, so even this simple text mining system has a clear application to the relief of human suffering.

Biologists and bioinformaticians have recently come up with even more innovative uses for text mining systems, in which the output is never intended for

Semantic class	Examples
Cell lines	*T98G, HeLa cell, Chinese hamster ovary cells, CHO cells*
Cell types	*primary T lymphocytes, natural killer cells, NK cells*
Chemicals	*citric acid, 1,2-diiodopentane, C*
Drugs	*cyclosporin A, CDDP*
Genes/proteins	*white, HSP60, protein kinase C, L23A*
Malignancies	*carcinoma, breast neoplasms*
Medical/clinical concepts	*amyotrophic lateral sclerosis*
Mouse strains	*LAFT, AKR*
Mutations	*C10T, Ala64 → Gly*
Populations	*judo group*

Figure 22.31 A sample of the semantic classes of named entities that have been recognized in biomedical NLP. Note the surface similarities between many of the examples.

viewing by humans, but rather is used as part of the analysis of high-throughput assays—experimental methods which produce masses of data points that would have been unimaginable just twenty years ago—and as part of techniques for using data in genomic data repositories. Ng (2006) provides a review and an insightful analysis of work in this vein.

22.5.1 Biological Named Entity Recognition

Information extraction tasks in the biological realm are characterized by a much wider range of relevant types of entities than the PERSON, ORGANIZATION, and LOCATION semantic classes that characterize work that is focused on news-style texts. Fig. 22.31 and the following example illustrate just a small subset of the variety of semantic classes of named entities that have been the target of NER systems in the biomedical domain.

> [$_{TISSUE}$ Plasma] [$_{GP}$ BNP] concentrations were higher in both the [$_{POPULATION}$ judo] and [$_{POPULATION}$ marathon groups] than in [$_{POPULATION}$ controls], and positively correlated with [$_{ANAT}$ LV] mass as well as with deceleration time.

Nearly all of the techniques described in Sec. 22.1 have been applied to the biomedical NER problem, with a particular focus on the problem of recognizing gene/protein names. This task is particularly difficult due to the wide range of forms that gene names can take: *white, insulin, BRCA1, ether a go-go,* and *breast cancer associated 1* are all the names of genes. The choice of algorithm for gene name recognition seems to be less important than the choice of features; typical feature sets include word-shape and contextual features, as discussed earlier; additionally, knowledge-based features, such as using the count of Google hits for a sequence like *BRCA1 gene* to decide whether or not a token of the string *BRCA1* is a reference to a gene or not, are sometimes incorporated into statistical systems.

Surprisingly, the use of huge publicly available lists of gene names has not generally contributed to the performance of a gene/protein NER system (Yeh et al., 2005), and in fact may actually degrade it (Jr. et al., 2006). It is not uncommon for gene names to be many tokens long (e.g. *breast cancer associated 1*). Gene name length has a demonstrable effect on NER system performance (Kinoshita et al., 2005; Yeh et al., 2005), and any technique for correctly finding the boundaries of multi-token names seems to increase performance. Use of the abbreviation-definition-detection algorithm (Schwartz and Hearst, 2003) is common for this purpose, since many such names appear as abbreviation or symbol definitions at some point in a publication. Base noun group chunkers can also be useful in this regard, as can a surprisingly small number of heuristic rules (Kinoshita et al., 2005).

22.5.2 Gene Normalization

GENE NORMALIZATION

Having identified all the mentions of biological entities in a text, the next step is to map them to unique identifiers in databases or ontologies. This task has been most heavily studied for genes, where it is known as **gene normalization**. Some of the complexities of the problem come from high degrees of variability in the realization of the names of specific entities in naturally-occurring text; the nature of the problem was first delineated by Cohen et al. (2002). In that work a standard discovery procedure from descriptive linguistics was used to determine what sorts of variability in gene names can be ignored, and what sorts must not be ignored. More recently, Morgan et al. (2007) have shown how linguistic characteristics of community-specific gene-naming conventions affect the complexity of this task when the normalization of genes from varying species is attempted. Gene normalization can be considered a type of word sense disambiguation task, midway between a targetted WSD task and an all-words WSD task.

An important thread of work on this problem involves mapping named entities to biomedical ontologies, especially the Gene Ontology (Ashburner et al., 2000). This has proven considerably more challenging; terms in the Gene Ontology tend to be long, to have many possible lexical and syntactic forms, and to sometimes require significant amounts of inference. ? (?) introduce this ontology from the perspective of computational lexical semantics and review much of the named entity recognition work that has involved it.

22.5.3 Biological Roles and Relations

Finding and normalizing all the mentions of biological entities in a text is a preliminary step to determining the roles played by entities in the text. Two ways to do this that have been the focus of recent research are to discover and classify the expressed binary relations between the entities in a text, and to identify and classify the roles played by entities with respect to the central events in the text. These two tasks correspond roughly to the tasks of classifying the relationship between pairs of entities as described in Sec. 22.2, and to the semantic role labeling task introduced in Ch. 20.

Consider the following example texts that express binary relations between entities.

(22.25) These results suggest that con A-induced [$_{DISEASE}$ hepatitis] was ameliorated by pretreatment with [$_{TREATMENT}$ TJ-135].

(22.26) [$_{DISEASE}$ Malignant mesodermal mixed tumor of the uterus] following [$_{TREATMENT}$ irradiation]

Each of these examples asserts a relationship between a *disease* and a *treatment*. In the first example, the relationship can be classified as that of *curing*. In the second example, the disease is a *result* of the mentioned treatment. Rosario and Hearst (2004) present a system for the classification of 7 kinds disease-treatment relations. In this work, a series of HMM-based generative models as well as a discriminative neural network model were successfully applied.

More generally, a wide-range of rule-based and statistical approaches have been applied to binary relation recognition problems such as this. Examples of other widely studied biomedical relation recognition problems include genes and their biological functions (Blaschke et al., 2005), genes and drugs (Rindflesch et al., 2000), genes and mutations (Rebholz-Schuhmann et al., 2004), and protein-protein interactions (Rosario and Hearst, 2005).

Now consider the following example that corresponds to a semantic role labeling style of problem.

(22.27) [$_{THEME}$ Full-length cPLA2] was [$_{TARGET}$ phosphorylated] stoichiometrically by [$_{AGENT}$ p42 mitogen-activated protein (MAP) kinase] in vitro... and the major site of phosphorylation was identified by amino acid sequencing as [$_{SITE}$ Ser505]

The *phosphorylation* event that lies at the core of this text has three semantic roles associated with it: the causal AGENT of the event, the THEME or entity being phosphorylated and finally the location, or SITE of the event. The problem is to identify the constituents in the input that play these roles and assign them the correct role labels. Note that this example, contains a further complication in that the second event mention *phosphorylation* must be identified as coreferring with the first *phosphorylated* in order to capture the SITE role correctly.

Much of the difficulty with semantic role labeling in the biomedical domain stems from the preponderance of nominalizations in these texts. Nominalizations like *phosphorylation* typically offer fewer syntactic cues to signal their arguments than their verbal equivalents, making the identification task more difficult. A further complication is that different semantic roles arguments often occur as parts of the same, or dominating nominal constituents. To see this consider the following examples.

(22.28) Serum stimulation of fibroblasts in floating matrices does not result in [$_{TARGET}$ [$_{ARG1}$ ERK] translocation] to the [$_{ARG3}$ nucleus] and there was decreased serum activation of upstream members of the ERK signaling pathway, MEK and Raf,

(22.29) The translocation of RelA/p65 was investigated using Western blotting and immunocytochemistry. the COX-2 inhibitor SC236 worked directly through suppressing [$_{TARGET}$ [$_{ARG3}$ nuclear] translocation] of [$_{ARG1}$ RelA/p65].

(22.30) Following UV treatment, Mcl-1 protein synthesis is blocked, the existing pool of Mcl-1 protein is rapidly degraded by the proteasome, and [$_{ARG1}$ [$_{ARG2}$ cytosolic] Bcl-xL] [$_{TARGET}$ translocates] to the [$_{ARG3}$ mitochondria]

Each these examples contains arguments that are bundled into constituents with other arguments or with the target predicate itself. For example, in the second example the constituent *nuclear translocation* signals both the TARGET and the ARG3 role.

Both rule-based and statistical approaches have been applied to these semantic role-like problems. As with relation-finding and NER, the choice of algorithm is less important than the choice of features, many of which are derived from accurate syntactic analyses. However, since there are no large treebanks available for biological texts, we are left with the option using off-the-shelf parsers trained on generic newswire texts. Of course, the errors introduced in this process may negate whatever power we can derive from syntactic features. Therefore, an important area of research revolves around the adaptation of generic syntactic tools to this domain (Blitzer et al., 2006).

Relational and event extraction applications in this domain often have an extremely limited foci. The motivation for this is that even systems with narrow scope can make a contribution to the productivity of working bioscientists. An extreme example of this is the RLIMS-P system discussed earlier. It tackles only the verb *phosphorylate* and the associated nominalization *phosphorylization.* Nevertheless, this system was successfully used to produce a large online database that is in widespread use by the research community.

As the targets of biomedical information extraction applications have become more ambitious, the range of BioNLP application types has become correspondingly more broad. Computational lexical semantics and semantic role labelling (Verspoor et al., 2003; Wattarujeekrit et al., 2004; Ogren et al., 2004; Kogan et al., 2005; Cohen and Hunter, 2006), summarization (Lu et al., 2006), and question-answering are all active research topics in the biomedical domain. Shared tasks like BioCreative continue to be a source of large data sets for named entity recognition, question-answering, relation extraction, and document classification (Hirschman and Blaschke, 2006), as well as a venue for head-to-head assessment of the benefits of various approaches to information extraction tasks.

22.6 Summary

This chapter has explored a series of techniques for extracting limited forms of semantic content from texts. Most techniques can be characterized as problems in detection followed by classification.

- **Named entities** can be recognized and classified by **statistical sequence labeling** techniques.
- **Relations among entities** can be detected and classified using supervised learning methods when annotated training data is available; lightly supervised **bootstrapping** methods can be used when small numbers of **seed tuples** or **seed patterns** are available.
- Reasoning about time can be facilitated by detecting and normalizing **temporal expressions** through a combination of statistical learning and rule-based methods.
- Rule-based and statistical methods can be used to detect, classify and order **events** in time. The **TimeBank corpus** can facilitate the training and evaluation of temporal analysis systems.
- **Template-filling** applications can recognize stereotypical situations in texts and assign elements from the text to roles represented as **fixed sets of slots**.
- Information extraction techniques have proven to be particularly effective in processing texts from the **biological domain**.
- Scripts, plans and goals...

Bibliographical and Historical Notes

The earliest work on information extraction addressed the template-filling task and was performed in the context of the Frump system (DeJong, 1982). Later work was stimulated by the U.S. government sponsored MUC conferences (Sundheim, 1991, 1992, 1993, 1995). Chinchor et al. (1993) describes the evaluation techniques used in the MUC-3 and MUC-4 conferences. Hobbs (1997) partially credits the inspiration for FASTUS to the success of the University of Massachusetts CIRCUS system (Lehnert et al., 1991) in MUC-3. The SCISOR system is another system based loosely on cascades and semantic expectations that did well in MUC-3 (Jacobs and Rau, 1990).

Due to the difficulty of reusing or porting systems from one domain to another, attention shifted to the problem of automatic knowledge acquisition for these systems. The earliest supervised learning approaches to IE are described in Cardie (1993), Cardie (1994), Riloff (1993), Soderland et al. (1995), Huffman (1996), and Freitag (1998).

These early learning efforts focused on automating the knowledge acquisition process for mostly finite-state rule-based systems. Their success, and the earlier success of HMM-based methods for automatic speech recognition, led to the development of statistical systems based on sequence labeling. Early efforts applying HMMs to IE problems include the work of Bikel et al. (1997, 1999) and Freitag and McCallum (1999). Subsequent efforts demonstrated the effectiveness of a range of statistical methods including MEMMs (McCallum et al., 2000), CRFs (Lafferty et al., 2001) and SVMs (Sassano and Utsuro, 2000; McNamee and Mayfield, 2002).

Progress in this area continues to be stimulated by formal evaluations with shared benchmark datasets. The MUC evaluations of the mid-1990s were succeeded by the Automatic Content Extraction (ACE) program evaluations held periodically from 2000 to 2007.[6] These evaluations focused on the named entity recognition, relation detection, and temporal expression detection and normalization tasks. Other IE evaluations include the 2002 and 2003 CoNLL shared tasks on language-independent named entity recognition (Sang, 2002; Sang and De Meulder, 2003), and the 2007 SemEval tasks on temporal analysis (Verhagen et al., 2007) and people search (Artiles et al., 2007).

The scope of information extraction continues to expand to meet the ever-increasing needs of applications for novel kinds of information. Some of the emerging IE tasks that we haven't discussed include the classification of gender

[6] www.nist.gov/speech/tests/ace/

USER GENERATED CONTENT

SOCIAL MEDIA

(Koppel et al., 2002), moods (Mishne and de Rijke, 2006), sentiment, affect and opinions (Qu et al., 2004). Much of this work involves **user generated content** in the context of **social media** such as blogs, discussion forums, newsgroups and the like. Research results in this domain have been the focus of a number of recent workshops and conferences (Nicolov et al., 2006; Nicolov and Glance, 2007).

EXERCISES

22.1 Develop a set of regular expressions to recognize the character shape features described in Fig. 22.7.

22.2 Using a statistical sequence modeling toolkit of your choosing, develop and evaluate an NER system.

22.3 The IOB labeling scheme given in this chapter isn't the only possible one. For example, an E tag might be added to mark the end of entities, or the B tag can be reserved only for those situations where an ambiguity exists between adjacent entities. Propose a new set of IOB tags for use with your NER system. Perform experiments and compare its performance against the scheme presented in this chapter.

22.4 Names of works of art (books, movies, video games, etc.) are quite different from the kinds of named entities we've discussed in this chapter. Collect a list of names of works of art from a particular category from a web-based source (eg. gutenberg.org, amazon.com, imdb.com, etc.). Analyze your list and give examples of ways that the names in it are likely to be problematic for the techniques described in this chapter.

22.5 Develop an NER system specific to the category of names that you collected in the last exercise. Evaluate your system on a collection of text likely to contain instances of these named entities.

22.6 Acronym expansion, the process of associating a phrase with a particular acronym, can be accomplished by a simple form of relational analysis. Develop a system based on the relation analysis approaches described in this chapter to populate a database of acronym expansions. If you focus on English **Three Letter Acronyms** (TLAs) you can evaluate your system's performance by comparing it to Wikipedia's TLA page (`en.wikipedia.org/wiki/Category:Lists_of_TLAs`).

22.7 Collect a corpus of biographical Wikipedia entries of prominent people from some coherent area of interest (sports, business, computer science, linguistics, etc.).

Develop a system that can extract an occupational timeline for the subjects of these articles. For example, the Wikipedia entry for Peter Norvig might result in the ordering: Sun, Harlequin, Junglee, NASA, Google; the entry for David Beckham would be: Manchester United, Real Madrid, Los Angeles Galaxy.

22.8 A useful functionality in newer email and calendar applications is the ability to associate temporal expressions associated with events in emails (doctor's appointments, meeting planning, party invitations, etc.) with specific calendar entries. Collect a corpus of emails containing temporal expressions related to event planning. How do these expressions compare to the kind of expressions commonly found in news text that we've been discussing in this chapter?

22.9 Develop and evaluate a recognition system capable of recognizing temporal expressions of the kind appearing in your email corpus.

22.10 Design a system capable of normalizing these expressions to the degree required to insert them into a standard calendaring application.

22.11 Acquire the CMU seminar announcement corpus and develop a template-filling system using any of the techniques mentioned in Sec. 22.4. Analyze how well your system performs as compared to state-of-the-art results on this corpus.

22.12 Develop a new template that covers a situation commonly reported on by standard news sources. Carefully characterize your slots in terms of the kinds of entities that appear as slot-fillers. Your first step in this exercise should be to acquire a reasonably sized corpus of stories that instantiate your template.

22.13 Given your corpus, develop an approach to annotating the relevant slots in your corpus so that it can serve as a training corpus. Your approach should involve some hand-annotation, but should not be based solely on it.

22.14 Retrain your system and analyze how well it functions on your new domain.

22.15 Species identification is a critical issue for biomedical information extraction applications such as document routing and classification. But it is especially crucial for realistic versions of the gene normalization problem.

Build a species identification system that works on the document level, using the machine learning or rule-based method of your choice. As gold standard data, use the BioCreative gene normalization data (`biocreative.sourceforge.net`).

22.16 Build, or borrow, a named entity recognition system that targets mentions of genes and gene products in texts. As development data, use the BioCreative gene mention corpus (`biocreative.sourceforge.net`).

22.17 Build a gene normalization system that maps the output of your gene mention recognition system to the appropriate database entry. Use the BioCreative gene normalization data as your development and test data, *be sure you don't give your system access to the species identification in the metadata.*

Agichtein, E. and Gravano, L. (2000). Snowball: Extracting relations from large plain-text collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*.

Ahn, D., Adafre, S. F., and de Rijke, M. (2005). Extracting temporal information from open domain text: A comparative exploration. In *Proceedings of the 5th Dutch-Belgian Information Retrieval Workshop (DIR'05)*.

Allen, J. (1984). Towards a general theory of action and time. *Artificial Intelligence*, *23*(2), 123–154.

Appelt, D. E., Hobbs, J. R., Bear, J., Israel, D., Kameyama, M., Kehler, A., Martin, D., Myers, K., and Tyson, M. (1995). SRI International FASTUS system MUC-6 test results and analysis. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, San Francisco, pp. 237–248. Morgan Kaufmann.

Appelt, D. E. and Israel, D. (1997). ANLP-97 tutorial: Building information extraction systems. Available as `www.ai.sri.com/~appelt/ie-tutorial/`.

Artiles, J., Gonzalo, J., and Sekine, S. (2007). The semeval-2007 weps evaluation: Establishing a benchmark for the web people search task. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, Prague, Czech Republic, pp. 64–69. Association for Computational Linguistics.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, M. A., Hill, D. P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J. E., Ringwald, M., Rubin, G. M., and Sherlock, G. (2000). Gene ontology: tool for the unification of biology. *Nature Genetics*, *25*(1), 25–29.

Bikel, D. M., Miller, S., Schwartz, R., and Weischedel, R. (1997). Nymble: a high-performance learning name-finder. In *Proceedings of ANLP-97*, pp. 194–201.

Bikel, D. M., Schwartz, R., and Weischedel, R. (1999). An algorithm that learns what's in a name. *Machine Learning*, *34*, 211–231.

Blaschke, C., Leon, E. A., Krallinger, M., and Valencia, A. (2005). Evaluation of BioCreative assessment of task 2. *BMC Bioinformatics*, *6*(2).

Blitzer, J., McDonald, R., and Pereira, F. (2006). Domain adaptation with structural correspondence learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia.

Bunescu, R. C. and Mooney, R. J. (2005). A shortest path dependency kernel for relation extraction. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pp. 724–731.

Cardie, C. (1993). A case-based approach to knowledge acquisition for domain specific sentence analysis. In *AAAI-93*, pp. 798–803. AAAI Press.

Cardie, C. (1994). *Domain-Specific Knowledge Acquisition for Conceptual Sentence Analysis*. Ph.D. thesis, University of Massachusetts, Amherst, MA. Available as CMPSCI Technical Report 94-74.

Chinchor, N., Hirschman, L., and Lewis, D. L. (1993). Evaluating Message Understanding systems: An analysis of the third Message Understanding Conference. *Computational Linguistics*, *19*(3), 409–449.

Cohen, K. B. and Hunter, L. (2006). A critical review of PASBio's argument structures for biomedical verbs. *BMC Bioinformatics*, *7*(Suppl 3).

Cohen, K. B., Dolbey, A., Mensah, A. G., and Hunter, L. (2002). Contrast and variability in gene names. In *Proceedings of the ACL Workshop on Natural Language Processing in the Biomedical Domain*, pp. 14–20.

Cohen, K. B. and Hunter, L. (2004). Natural language processing and systems biology. In Dubitzky, W. and Azuaje, F. (Eds.), *Artificial Intelligence Methods and Tools for Systems Biology*, pp. 147–174. Springer.

Culotta, A. and Sorensen, J. (2004). Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*.

DeJong, G. F. (1982). An overview of the FRUMP system. In Lehnert, W. G. and Ringle, M. H. (Eds.), *Strategies for Natural Language Processing*, pp. 149–176. Lawrence Erlbaum.

Etzioni, O., Cafarella, M., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D. S., and Yates, A. (2005). Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, *165*(1), 91–134.

Ferro, L., Gerber, L., Mani, I., Sundheim, B., and Wilson, G. (2005). Tides 2005 standard for the annotation of temporal expressions. Tech. rep., MITRE.

Fisher, D., Soderland, S., McCarthy, J., Feng, F., and Lehnert, W. G. (1995). Description of the UMass system as used for MUC-6. In *Proceedings of the Sixth Message*

Understanding Conference (MUC-6), San Francisco, pp. 127–140. Morgan Kaufmann.

Freitag, D. (1998). Multistrategy learning for information extraction. In *ICML 1998*, Madison, WI, pp. 161–169.

Freitag, D. and McCallum, A. (1999). Information extraction using hmms and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Retrieval*.

Gaizauskas, R., Wakao, T., Humphreys, K., Cunningham, H., and Wilks, Y. (1995). University of Sheffield: Description of the LaSIE system as used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, San Francisco, pp. 207–220. Morgan Kaufmann.

Grishman, R. and Sundheim, B. (1995). Design of the MUC-6 evaluation. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, San Francisco, pp. 1–11. Morgan Kaufmann.

Hirschman, L. and Blaschke, C. (2006). Evaluation of text mining in biology. In Ananiadou, S. and McNaught, J. (Eds.), *Text Mining for Biology and Biomedicine*, chap. 9, pp. 213–245. Artech House, Norwood, MA.

Hobbs, J. R., Appelt, D. E., Bear, J., Israel, D., Kameyama, M., Stickel, M. E., and Tyson, M. (1997). FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. In Roche, E. and Schabes, Y. (Eds.), *Finite-State Language Processing*, pp. 383–406. MIT Press.

Huffman, S. (1996). Learning information extraction patterns from examples. In Wertmer, S., Riloff, E., and Scheller, G. (Eds.), *Connectionist, Statistical, and Symbolic Approaches to Learning Natural Language Processing*, pp. 246–260. Springer, Berlin.

ISO8601 (2004). Data elements and interchange formats information interchange representation of dates and times. Tech. rep., International Organization for Standards (ISO).

Jackson, P. and Moulinier, I. (2002). *Natural language processing for online applications: text retrieval, extraction, and categorization*. John Benjamins Publishing Company.

Jacobs, P. and Rau, L. (1990). SCISOR: A system for extracting information from on-line news. *Communications of the ACM*, *33*(11), 88–97.

Jr., W. A. B., Cohen, K. B., Fox, L., Acquaah-Mensah, G. K., and Hunter, L. (2007). Manual curation is not sufficient for annotation of genomic databases. *Bioinformatics*, *23*, i41–i48.

Jr., W. A. B., Lu, Z., Johnson, H. L., Caporaso, J. G., Paquette, J., Lindemann, A., White, E. K., Medvedeva, O., Cohen, K. B., and Hunter, L. (2006). An integrated approach to concept recognition in biomedical text. In *Proceedings of BioCreative 2006*.

Kinoshita, S., Cohen, K. B., Ogren, P. V., and Hunter, L. (2005). BioCreAtIvE Task1A: entity identification with a stochastic tagger.. *BMC Bioinformatics*, *6*(1).

Kogan, Y., Collier, N., Pakhomov, S., and Krauthammer, M. (2005). Towards semantic role labeling & IE in the medical literature. In *AMIA 2005 Symposium Proceedings*, pp. 410–414.

Koppel, M., Argamon, S., and Shimoni, A. R. (2002). Automatically categorizing written texts by author gender. *Literary and Linguistic Computing*, *17*(4), 401–412.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML 2001*, Stanford, CA.

Lehnert, W. G., Cardie, C., Fisher, D., Riloff, E., and Williams, R. (1991). Description of the CIRCUS system as used for MUC-3. In Sundheim, B. (Ed.), *Proceedings of the Third Message Understanding Conference*, pp. 223–233. Morgan Kaufmann.

Lu, Z., Cohen, B. K., and Hunter, L. (2006). Finding GeneRIFs via Gene Ontology annotations.. In *PSB 2006*, pp. 52–63.

McCallum, A. (2005). Information extraction: Distilling structured data from unstructured text. *ACM Queue*, 48–57.

McCallum, A., Freitag, D., and Pereira, F. C. N. (2000). Maximum entropy Markov models for information extraction and segmentation. In *ICML 2000*, pp. 591–598.

McNamee, P. and Mayfield, J. (2002). Entity extraction without language-specific resources. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL-2002)*, Taipei, Taiwan.

Mikheev, A., Moens, M., and Grover, C. (1999). Named entity recognition without gazetteers. In *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics*, Morristown, NJ, USA, pp. 1–8. Association for Computational Linguistics.

Mishne, G. and de Rijke, M. (2006). MoodViews: Tools for blog mood analysis. In Nicolov, N., Salvetti, F., Liberman, M., and Martin, J. H. (Eds.), *Computational Approaches to Analyzing Weblogs: Papers from the 2006 Spring Symposium*, Stanford, Ca. AAAI.

Morgan, A. A., Wellner, B., Colombe, J. B., Arens, R., Colosimo, M. E., and Hirschman, L. (2007). Evaluating human gene and protein mention normalization to unique identifiers. In *Pacific Symposium on Biocomputing*, pp. 281–291.

Ng, S.-K. (2006). Integrating text mining with data mining. In Ananiadou, S. and McNaught, J. (Eds.), *Text mining for biology and biomedicine*. Artech House Publishers.

Nicolov, N. and Glance, N. (Eds.). (2007). *Proceedings of the First International Conference on Weblogs and Social Media (ICWSM)*, Boulder, CO.

Nicolov, N., Salvetti, F., Liberman, M., and Martin, J. H. (Eds.). (2006). *Computational Approaches to Analyzing Weblogs: Papers from the 2006 Spring Symposium*, Stanford, Ca. AAAI.

Ogren, P. V., Cohen, K. B., Acquaah-Mensah, G. K., Eberlein, J., and Hunter, L. (2004). The compositional structure of Gene Ontology terms. In *Pac Symp Biocomput*, pp. 214–225.

Peshkin, L. and Pfefer, A. (2003). Bayesian information extraction network. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*.

Pustejovsky, J., Castao, J., Ingria, R., Saur, R., Gaizauskas, R., Setzer, A., and Katz, G. (2003a). TimeML: robust specification of event and temporal expressions in text. In *Proceedings of the Fifth International Workshop on Computational Semantics (IWCS-5)*.

Pustejovsky, J., Hanks, P., Saur, R., See, A., Gaizauskas, R., Setzer, A., Radev, D., Sundheim, B., Day, D., Ferro, L., and Lazo, M. (2003b). The TIMEBANK corpus. In *Proceedings of Corpus Linguistics 2003 Conference*, pp. 647–656.

Pustejovsky, J., Ingria, R., Sauri, R., Castano, J., Littman, J., Gaizauskas, R., Setzer, A., Katz, G., and Mani, I. (2005). *The Specification Language TimeML*, chap. 27. Oxford, Oxford, England.

Qu, Y., Shanahan, J., and Wiebe, J. (Eds.). (2004). *Exploring Attitude and Affect in Text: Papers from the 2004 Spring Symposium*, Stanford, Ca. AAAI.

Rebholz-Schuhmann, D., Marcel, S., Albert, S., Tolle, R., Casari, G., and Kirsch, H. (2004). Automatic extraction of mutations from medline and cross-validation with omim. *Nucleic Acids Research*, *32*(1), 135–142.

Riloff, E. (1993). Automatically constructing a dictionary for information extraction tasks. In *AAAI-93*, Washington, D.C., pp. 811–816.

Riloff, E. and Jones, R. (1999). Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI)*, pp. 474–479.

Rindflesch, T. C., Tanabe, L., Weinstein, J. N., and Hunter, L. (2000). EDGAR: Extraction of drugs, genes and relations from the biomedical literature. In *Pacific Symposium on Biocomputing*, pp. 515–524.

Rosario, B. and Hearst, M. A. (2004). Classifying semantic relations in bioscience texts. In *Proceedings of ACL 2004*, pp. 430–437.

Rosario, B. and Hearst, M. A. (2005). Multi-way Relation Classification: Application to Protein-Protein Interactions. In *Proceedings of the 2005 HLT-NAACL*.

Roth, D. and tau Yih, W. (2001). Relational learning via propositional algorithms: An information extraction case study. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1257–1263.

Sang, E. F. T. K. (2002). Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pp. 155–158. Taipei, Taiwan.

Sang, E. F. T. K. and De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. In Daelemans, W. and Osborne, M. (Eds.), *Proceedings of CoNLL-2003*, pp. 142–147. Edmonton, Canada.

Sassano, M. and Utsuro, T. (2000). Named entity chunking techniques in supervised learning for japanese named entity recognition. In *COLING-00*, Saarbrcken, Germany, pp. 705–711.

Schank, R. C. and Abelson, R. P. (1977). *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum.

Schwartz, A. S. and Hearst, M. A. (2003). A simple algorithm for identifying abbreviation definitions in biomedical text. In *Pacific Symposium on Biocomputing*, Vol. 8, pp. 451–462.

Settles, B. (2005). ABNER: an open source tool for automatically tagging genes, proteins and other entity names in text. *Bioinformatics*, *21*(14), 3191–3192.

Soderland, S., Fisher, D., Aseltine, J., and Lehnert, W. G. (1995). CRYSTAL: Inducing a conceptual dictionary. In *IJCAI-95*, Montreal, pp. 1134–1142.

Sundheim, B. (Ed.). (1991). *Proceedings of the Third Message Understanding Conference*. Morgan Kaufmann.

Sundheim, B. (Ed.). (1992). *Proceedings of the Fourth Message Understanding Conference*. Morgan Kaufmann.

Sundheim, B. (Ed.). (1993). *Proceedings, Fifth Message Understanding Conference (MUC-5), Baltimore, MD*. Morgan Kaufmann.

Sundheim, B. (Ed.). (1995). *Proceedings of the Sixth Message Understanding Conference*. Morgan Kaufmann.

van Rijsbergen, C. J. (1975). *Information Retrieval*. Butterworths, London.

Verhagen, M., Gaizauskas, R., Schilder, F., Hepple, M., Katz, G., and Pustejovsky, J. (2007). Semeval-2007 task 15: Tempeval temporal relation identification. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, Prague, Czech Republic, pp. 75–80. Association for Computational Linguistics.

Verspoor, C. M., Joslyn, C., and Papcun, G. J. (2003). The gene ontology as a source of lexical semantic knowledge for a biological natural language processing application. In *Proceedings of the SIGIR'03 Workshop on Text Analysis and Search for Bioinformatics*, Toronto, CA.

Wattarujeekrit, T., Shah, P. K., and Collier, N. (2004). PASBio: predicate-argument structures for event extraction in molecular biology. *BMC Bioinformatics*, *5*(155).

Weischedel, R. (1995). BBN: Description of the PLUM system as used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, San Francisco, pp. 55–70. Morgan Kaufmann.

Yeh, A., Morgan, A., Colosimo, M., and Hirschman, L. (2005). BioCreatve task 1A: gene mention finding evaluation. *BMC Bioinformatics*, *6*(1).

Zhou, G., Su, J., Zhang, J., and Zhang, M. (2005). Exploring various knowledge in relation extraction. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, Ann Arbor, Michigan, pp. 427–434. Association for Computational Linguistics.

23 QUESTION ANSWERING AND SUMMARIZATION

‘Alright’, said Deep Thought. ‘The Answer to the Great Question...’
‘Yes!’
‘Of Life The Universe and Everything...’ said Deep Thought.
‘Yes!’
‘Is...’
‘Yes...!!!...?’
‘Forty-two’, said Deep Thought, with infinite majesty and calm...
Douglas Adams, *The Hitchhiker’s Guide to the Galaxy*

I read *War and Peace*. . . It’s about Russia. . .
Woody Allen, *Without Feathers*

Because so much text information is available generally on the Web, or in specialized collections such as PubMed, or even on the hard drives of our laptops, the single most important use of language processing these days is to help us query and extract meaning from these large repositories. If we have a very structured idea of what we are looking for, we can use the information extraction algorithms of the previous chapter. But many times we have an information need that is best expressed more informally in words or sentences, and we want to find either a specific answer fact, or a specific document, or something in between.

In this chapter we introduce the tasks of **question answering** (**QA**) and **summarization**, tasks which produce specific phrases, sentences, or short passages, often in response to a user’s need for information expressed in a natural language query. In studying these topics, we will also cover highlights from the field of **information retrieval** (**IR**), the task of returning documents which are relevant to a particular natural language query. IR is a complete field in its own right, and we will only be giving a brief introduction to it here, but one that is essential for understand QA and summarization.

In this chapter we focus on a central idea behind all of these subfields, the idea of meeting a user’s information needs by **extracting** passages directly from documents or from document collections like the Web.

Information retrieval (**IR**) is an extremely broad field, encompassing a wide-range of topics pertaining to the storage, analysis, and retrieval of all manner of media, including text, photographs, audio, and video (Baeza-Yates and Ribeiro-Neto, 1999). Our concern in this chapter is solely with the storage and retrieval of text documents in response to users' word-based queries for information. In section 23.1 we present the **vector space model**, some variant of which is used in most current systems, including most web search engines.

Rather than make the user read through an entire document, we'd often prefer to give a single concise short answer. Researchers have been trying to automate this process of **question answering** since the earliest days of computational linguistics (Simmons, 1965).

The simplest form of question answering is dealing with **factoid questions** . As the name implies, the answers to factoid questions are simple facts that can be found in short text strings. The following are canonical examples of this kind of question.

(23.1) Who founded Virgin Airlines?

(23.2) What is the average age of the onset of autism?

(23.3) Where is Apple Computer based?

Each of these questions can be answered directly with a text string that contain the name of person, a temporal expression, or a location, respectively. Factoid questions, therefore, are questions whose answers can be found in short spans of text and correspond to a specific, easily characterized, category, often a named entity of the kind we discussed in Ch. 22. These answers may be found on the Web, or alternatively within some smaller text collection. For example a system might answer questions about a company's product line by searching for answers in documents on a particular corporate website or internal set of documents. Effective techniques for answering these kinds of questions are described in Sec. 23.2.

Sometimes we are seeking information whose scope is greater than a single factoid, but less than an entire document. In such cases we might need a **summary** of a document or set of documents. The goal of **text summarization** is to produce an abridged version of a text which contains the important or relevant information. For example we might want to generate an **abstract** of a scientific article, a **summary** of email threads, a **headline** for a news article, or generate the short **snippets** that web search engines like Google return to the user to describe each retrieved document. For example, Fig. 23.1 shows some sample snippets from Google summarizing the first four documents returned from the query *German Expressionism Brücke*.

SNIPPETS

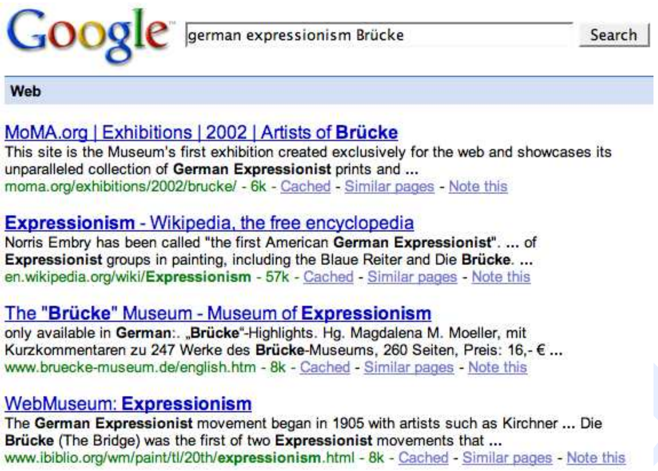
To produce these various kinds of summaries, we'll introduce algorithms for summarizing single documents, and those for producing summaries of multiple documents by combining information from different textual sources.

Finally, we turn to a field that tries to go beyond factoid question answering by borrowing techniques from summarization to try to answer more complex questions like the following:

(23.4) Who is Celia Cruz?

(23.5) What is a Hajj?

(23.6) In children with an acute febrile illness, what is the efficacy of single-medication therapy with acetaminophen or ibuprofen in reducing fever?



MoMA.org | Exhibitions | 2002 | Artists of **Brücke**
This site is the Museum's first exhibition created exclusively for the web and showcases its unparalleled collection of **German Expressionist** prints and ...
moma.org/exhibitions/2002/brucke/ - 6k - Cached - Similar pages - Note this

Expressionism - Wikipedia, the free encyclopedia
Norris Embry has been called "the first American **German Expressionist**". ... of **Expressionist** groups in painting, including the Blaue Reiter and Die **Brücke**. ...
en.wikipedia.org/wiki/**Expressionism** - 57k - Cached - Similar pages - Note this

The "**Brücke**" Museum - Museum of **Expressionism**
only available in **German**:. „**Brücke**"-Highlights. Hg. Magdalena M. Moeller, mit Kurzkommentaren zu 247 Werke des **Brücke**-Museums, 260 Seiten, Preis: 16,- € ...
www.bruecke-museum.de/english.htm - 8k - Cached - Similar pages - Note this

WebMuseum: **Expressionism**
The **German Expressionist** movement began in 1905 with artists such as Kirchner ... Die **Brücke** (The Bridge) was the first of two **Expressionist** movements that ...
www.ibiblio.org/wm/paint/tl/20th/**expressionism**.html - 8k - Cached - Similar pages - Note this

Figure 23.1 The first 4 snippets from Google for *German Expressionism Brücke*.

Answers to questions such as these do not consist of simple named entity strings. Rather they involve potentially lengthy coherent texts that knit together an array of associated facts to produce a biography, a complete definition, a summary of current events, or a comparison of clinic results on particular medical interventions. In addition to the complexity and style differences in these answers, the facts that go into such answers may be context, user, and time dependent.

COMPLEX QUESTIONS

Current methods answer these kinds of **complex questions** by piecing together relevant text segments that come from summarizing longer documents. For example we might construct an answer from text segments extracted from a a corporate report, a set of medical research journal articles, or a set of relevant news articles or web pages. This idea of summarizing text in response to a user query is called **query-based summarization** or **focused summarization**, and will be explored in Sec. 23.5.

QUERY-BASED SUMMARIZATION

Finally, we reserve for Ch. 24 all discussion of the role that questions play in extended dialogues; this chapter focuses only on responding to a single query.

23.1 INFORMATION RETRIEVAL

INFORMATION RETRIEVAL
IR

Information retrieval (**IR**) is a growing field that encompasses a wide range of topics related to the storage and retrieval of all manner of media. The focus of this section is with the storage of text documents and their subsequent retrieval in response to users' requests for information. In this section our goal is just to give a sufficient overview of information retrieval techniques to lay a foundation for the following sections on question answering and summarization. Readers with more interest specifically in information retrieval should see the references at the end of the chapter.

Most current information retrieval systems are based on a kind of extreme version of compositional semantics in which the meaning of a document resides solely in the

BAG-OF-WORDS

set of words it contains. To revisit the Mad Hatter's quote from the beginning of Ch. 19, in these systems *I see what I eat* and *I eat what I see* mean precisely the same thing. The ordering and constituency of the words that make up the sentences that make up documents play no role in determining their meaning. Because they ignore syntactic information, these approaches are often referred to as **bag-of-words** models.

DOCUMENT
COLLECTION
TERM
QUERY

Before moving on, we need to introduce some new terminology. In information retrieval, a **document** refers generically to the unit of text indexed in the system and available for retrieval. Depending on the application, a document can refer to anything from intuitive notions like newspaper articles, or encyclopedia entries, to smaller units such as paragraphs and sentences. In web-based applications, it can refer to a web page, a part of a page, or to an entire website. A **collection** refers to a set of documents being used to satisfy user requests. A **term** refers to a lexical item that occurs in a collection, but it may also include phrases. Finally, a **query** represents a user's information need expressed as a set of terms.

AD HOC RETRIEVAL

The specific information retrieval task that we will consider in detail is known as **ad hoc retrieval**. In this task, it is assumed that an unaided user poses a query to a retrieval system, which then returns a possibly ordered set of potentially useful documents. The high level architecture is shown in Fig. 23.2.

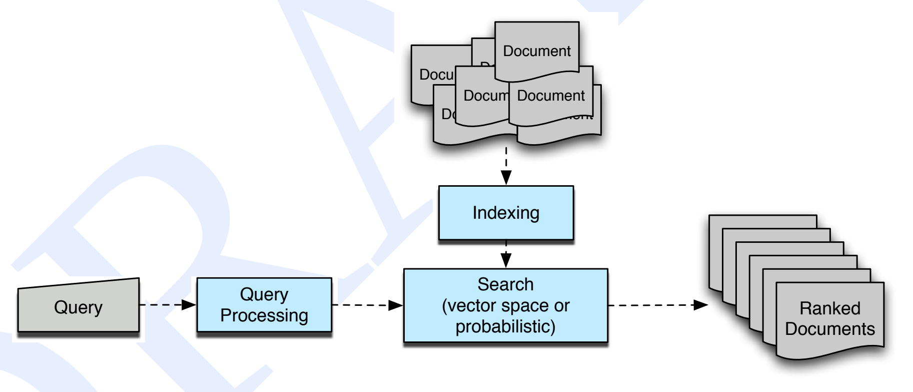


Figure 23.2 The architecture of an ad hoc IR system.

23.1.1 The Vector Space Model

VECTOR SPACE MODEL

In the **vector space model** of information retrieval, documents and queries are represented as vectors of features representing the terms (words) that occur within the collection (Salton, 1971).

TERM WEIGHT

The value of each feature is called the **term weight** and is usually a function of the term's frequency in the document, along with other factors.

For example, in a fried chicken recipe we found on the Web the four terms *chicken*, *fried*, *oil*, and *pepper* occur with term frequencies 8, 2, 7, and 4, respectively. So if we just used simple term frequency as our weights, and assuming we pretended only these

4 words occurred in the collection and we put the features are in the above order, the vector for this document (call it j) would be:

$$\vec{d_j} = (8,2,7,4)$$

More generally, we represent a vector for a document d_j as

$$\vec{d_j} = (w_{1,j}, w_{2,j}, w_{3,j}, \cdots, w_{n,j})$$

where $\vec{d_j}$ denotes a particular document, and the vector contains a weight feature for each of the N terms that occur in the collection as a whole; $w_{2,j}$ thus refers to the weight that term 2 has in document j.

We can also represent a query in the same way. For example, a query q for *fried chicken* would have the representation:

$$\vec{q} = (1,1,0,0)$$

More generally,

$$\vec{q} = (w_{1,q}, w_{2,q}, w_{3,q}, \cdots, w_{n,q})$$

Note that N, the number of dimensions in the vector, is the total number of terms in the whole collection. This can be hundreds of thousands of words, even if (as is often done) we don't consider some function words in the set of possible terms. But of course a query or even a long document can't contain very many of these hundreds of thousands of terms. Thus most of the values of the query and document vectors will be zero. Thus in practice we don't actually store all the zeros (we use hashes and other sparse representations).

Now consider a different document, a recipe for poached chicken; here the counts are:

$$\vec{d_k} = (6,0,0,0)$$

Intuitively we'd like the query q *fried chicken* to match document d_j (the fried chicken recipe) rather than document d_k (the poached chicken recipe). A brief glance at the feature suggests that this might be the case; both the query and the fried chicken recipe have the words *fried* and *chicken*, while the poached chicken recipe is missing the word *fried*.

It is useful to view the features used to represent documents and queries in this model as dimensions in a multi-dimensional space, where the feature weights serve to locate documents in that space. When a user's query is translated into a vector it denotes a point in that space. Documents that are located close to the query can then be judged as being more relevant than documents that are farther away.

Fig. 23.3 shows a graphical illustration, plotting the first two dimensions (*chicken* and *fried*) for all three vectors. Note that if we measure the similarity between vectors by the angle between the vectors, that q is more similar to d_j than to d_k, because the angle between q and d_j is smaller.

COSINE In vector-based information retrieval we standardly use the **cosine** metric that we

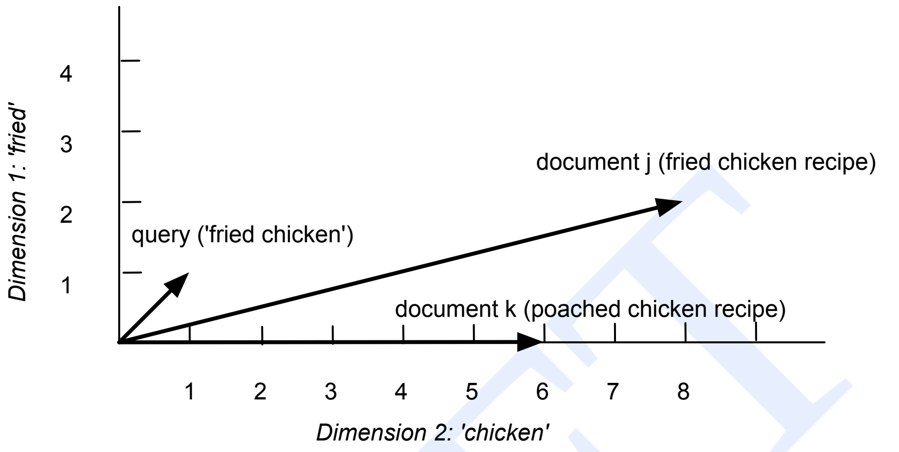


Figure 23.3 A graphical illustration of the vector model for information retrieval, showing the first two dimensions (*fried* and *chicken*) assuming that we use raw frequency in the document as the feature weights.

introduced in Ch. 20 rather than the actual angle. We measure the distance between two documents by the **cosine** of the angle between their vectors. When two documents are identical they will receive a cosine of one; when they are orthogonal (share no common terms) they will receive a cosine of zero. The equation for cosine is:

$$sim(\vec{q},\vec{d_j}) = \frac{\sum_{i=1}^{N} w_{i,q} \times w_{i,j}}{\sqrt{\sum_{i=1}^{N} w_{i,q}^2} \times \sqrt{\sum_{i=1}^{N} w_{i,j}^2}} \quad (23.7)$$

Recall from Ch. 20 that another way to think of the cosine ias as the **normalized dot product**. That is, the cosine is the dot product between the two vectors divided by the lengths of each of the two vectors. This is because the numerator of the cosine is the **dot product**:

DOT PRODUCT

$$\text{dot-product}(\vec{x},\vec{y}) = \vec{x}\cdot\vec{y} = \sum_{i=1}^{N} x_i \times u_i \quad (23.8)$$

while the denominator of the cosine contains terms for the lengths of the two vectors; recall that **vector length** is defined as:

VECTOR LENGTH

$$|\vec{x}| = \sqrt{\sum_{i=1}^{N} x_i^2} \quad (23.9)$$

This characterization of documents and queries as vectors provides all the basic parts for an ad hoc retrieval system. A document retrieval system can simply accept a user's query, create a vector representation for it, compare it against the vectors representing all known documents, and sort the results. The result is a list of documents rank ordered by their similarity to the query.

TERM-BY-DOCUMENT MATRIX

A further note on representation; the characterization of documents as vectors of term weights allows us to view the document collection as a whole as a (sparse) matrix of weights, where $w_{i,j}$ represents the weight of term i in document j. This weight matrix is typically called a **term-by-document matrix**. Under this view, the columns of the matrix represent the documents in the collection, and the rows represent the terms. The term-by-document matrix for the two recipe documents above (again using only the raw term frequency counts as the term weights) would be:

$$A = \begin{pmatrix} 8 & 6 \\ 2 & 0 \\ 7 & 0 \\ 4 & 0 \end{pmatrix}$$

23.1.2 Term Weighting

In the examples above, we assumed that the term weights were set as the simple frequency counts of the terms in the documents. This is a simplification of what we do in practice. The method used to assign terms weights in the document and query vectors has an enormous impact on the effectiveness of a retrieval system. Two factors have proven to be critical in deriving effective term weights. We have already seen the first, the term frequency, in its simplest form the raw frequency of a term within a document (Luhn, 1957). This reflects the intuition that terms that occur frequently within a document may reflect its meaning more strongly than terms that occur less frequently and should thus have higher weights.

INVERSE DOCUMENT FREQUENCY

IDF

The second factor is used to give a higher weight to words that only occur in a few documents. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection, while terms that occur frequently across the entire collection aren't as helpful. documents. The **inverse document frequency** or **IDF** term weight (Sparck Jones, 1972) is one way of assigning higher weights to these more discriminative words. IDF is defined via the fraction N/n_i, where N is the total number of documents in the collection, and n_i is the number of documents in which term i occurs, The fewer documents a term occurs in, the higher this weight. The lowest weight of 1 is assigned to terms that occur in all the documents. Due to the large number of documents in many collections, this measure is usually squashed with a log function. The resulting definition for inverse document frequency (IDF) is thus:

$$\text{idf}_i = \log\left(\frac{N}{n_i}\right) \tag{23.10}$$

TF-IDF

Combining term frequency with IDF results in a scheme known as **tf-idf** weighting:

$$w_{i,j} = \text{tf}_{i,j} \times \text{idf}_i \tag{23.11}$$

In tf-idf weighting, the weight of term i in the vector for document j is the product of its overall frequency in j with the log of its inverse document frequency in the collection (sometimes the term frequency is logged as well). Tf-idf thus prefers words which are frequent in the current document j but rare overall in the collection. Let's repeat the cosine formula for query-document comparison with tf-idf weights added. We'll modify the formula slightly, since as we noted earlier, most values for any query or document vector will be zero. This means that in practice we don't compute the cosine by iterating over all the (mostly zero) dimensions. Instead we only compute over the words that are present, as suggested by the following equation for the **tf-idf weighted cosine** between a query q and a document d:

$$sim(\vec{q},\vec{d}) = \frac{\sum\limits_{w\in q,d} tf_{w,q} tf_{w,d} (idf_w)^2}{\sqrt{\sum\limits_{q_i\in q}(tf_{q_i,q} idf_{q_i})^2} \times \sqrt{\sum\limits_{d_i\in d}(tf_{d_i,d} idf_{d_i})^2}} \tag{23.12}$$

With some minor variations, this tf-idf weighting scheme is used to assign term weights to documents in nearly all vector space retrieval models. The tf-idf scheme is also used in many other aspects of language processing; we'll see it again when we introduce **summarization** on page 31.

23.1.3 Term Selection and Creation

Thus far, we have been assuming that it is precisely the words that occur in a collection that are used to index the documents in the collection. Two common variations on this assumption involve the use of **stemming**, and a **stop list**.

STEMMING

Stemming, as we discussed in Ch. 3, is the process of collapsing the morphological variants of a word together. For example, without stemming, the terms *process*, *processing* and *processed* will be treated as distinct items with separate term frequencies in a term-by-document matrix; with stemming they will be conflated to the single term *process* with a single summed frequency count. The major advantage to using stemming is that it allows a particular query term to match documents containing any of the morphological variants of the term. The Porter stemmer (Porter, 1980) described in Ch. 3 is frequently used for retrieval from collections of English documents.

A problem with this approach is that it throws away useful distinctions. For example, consider the use of the Porter stemmer on documents and queries containing the words *stocks* and *stockings*. In this case, the Porter stemmer reduces these surface forms to the single term *stock*. Of course, the result of this is that queries concerning *stock prices* will return documents about *stockings*, and queries about *stockings* will find documents about *stocks*. Additionally we probably don't want to stem, e.g., the word *Illustrator* to *illustrate*, since the capitalized form *Illustrator* tends to refer to the software package Most modern web search engines therefore need to use more sophisticated methods for stemming.

STOP LIST

A second common technique involves the use of stop lists, which address the issue of what words should be allowed into the index. A **stop list** is simply a list of high frequency words that are eliminated from the representation of both documents and queries. Two motivations are normally given for this strategy: high frequency, closed-class terms are seen as carrying little semantic weight and are thus unlikely to help with retrieval, and eliminating them can save considerable space in the inverted index files used to map from terms to the documents that contain them. The downside of using a stop list is that it makes it difficult to search for phrases that contain words in the stop list. For example, a common stop list presented in Frakes and Baeza-Yates (1992), would reduce the phrase *to be or not to be* to the phrase *not*.

23.1.4 Evaluating Information Retrieval Systems

The basic tools used to measure the performance of ranked retrieval system are the **precision** and **recall** measures we employed in earlier settings. Here we assume that the returned items can be divided into two categories: those relevant to our purposes and those that are not. Therefore, precision is the fraction of the returned documents that are relevant, while recall is the fraction of all possible relevant documents that are contained in the return set. More formally, let's assume that we have been given a total of T ranked documents in response to a given information request, a subset of these documents, R, consists of relevant documents, and a disjoint subset, N, consists of the remaining irrelevant documents, and finally let's assume that there are U documents in the collection as a whole that are relevant to this particular request. Given all this we can define our precision and recall measures to be:

$$Precision = \frac{|R|}{|T|} \tag{23.13}$$

$$Recall = \frac{|R|}{|U|} \tag{23.14}$$

Unfortunately, these metrics are not quite sufficient to measure the performance of a system that *ranks* the documents it return. That is, if we are comparing the performance of two ranked retrieval systems, we require a metric that will prefer the one that ranks the relevant documents higher. Simple precision and recall as defined above are not dependent on rank in any way; we need to adapt them to capture how well a system does at putting relevant documents higher in the ranking. The two standard methods in information retrieval for accomplishing this are based on plotting precision/recall curves and on averaging precision measures in various ways.

Let's consider each of these methods in turn using the data given in the table in Fig. 23.4. This table provides rank-specific precision and recall values calculated as we proceed down through a set of ranked items. That is, the precision numbers are the fraction of relevant documents seen at a given rank, and recall is the fraction of relevant documents found at the same rank. The recall measures in this example are based on this query having 9 relevant documents in the collection as a whole. Note that recall is non-decreasing as we proceed, when relevant items are encountered recall increases and when non-relevant documents are found it remains unchanged. Precision on the other hand hops up and down, increasing when relevant documents are found and decreasing otherwise.

One common way to get a handle on this kind of data is to plot precision against recall on a single graph using data gathered from across a set of queries. To do this we'll need a way to average the recall and precision values across a set of queries. The standard way to do this is to plot averaged precision values at 11 fixed levels of recall (0 to 100, in steps of 10). Of course, as is illustrated by our earlier table we're not likely to have datapoints at these exact levels for all (or any) of the queries in our evaluation

Rank	Judgment	$\text{Precision}_{\mathit{Rank}}$	$\text{Recall}_{\mathit{Rank}}$
1	R	1.0	.11
2	N	.50	.11
3	R	.66	.22
4	N	.50	.22
5	R	.60	.33
6	R	.66	.44
7	N	.57	.44
8	R	.63	.55
9	N	.55	.55
10	N	.50	.55
11	R	.55	.66
12	N	.50	.66
13	N	.46	.66
14	N	.43	.66
15	R	.47	.77
16	N	.44	.77
17	N	.44	.77
18	R	.44	.88
19	N	.42	.88
20	N	.40	.88
21	N	.38	.88
22	N	.36	.88
23	N	.35	.88
24	N	.33	.88
25	R	.36	1.0

Figure 23.4 Rank-specific precision and recall values calculated as we proceed down through a set of ranked documents.

INTERPOLATED PRECISION

set. We'll therefore use **interpolated precision** values for the 11 recall values from the data points we do have. This is accomplished by choosing the maximum precision value achieved at any level of recall at or above the one we're calculating. In other words,

$$\mathit{IntPrecision}(r) = \max_{i>=r} \mathit{Precision}(i) \tag{23.15}$$

Note that this interpolation scheme not only provides us with the means to average performance over a set of queries, but it also provides a sensible way to smooth over the irregular precision values in the original data. This particular smoothing method is designed to give systems the benefit of the doubt by assigning the maximum precision value achieved at higher levels of recall from the one being measured. The interpolated data points for our earlier example are given in the following table and plotted in Fig. 23.5.

Given curves such as this we can compare two systems or approaches by comparing their curves. Clearly curves that are higher in precision across all recall values are

Interpolated Precision	Recall
1.0	0.0
1.0	.10
.66	.20
.66	.30
.66	.40
.63	.50
.55	.60
.47	.70
.44	.80
.36	.90
.36	1.0

Figure 23.5 Interpolated data points from Fig. 23.4.

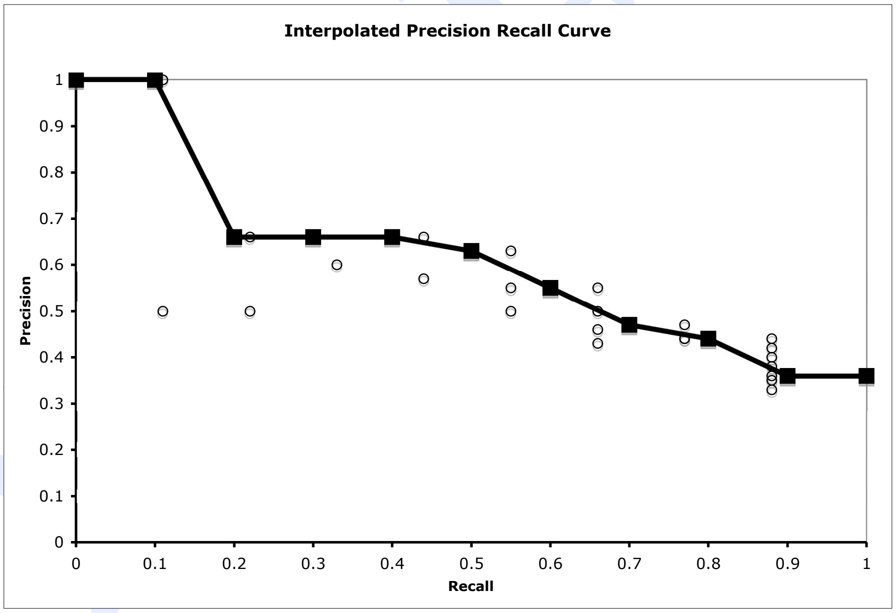


Figure 23.6 An 11 point interpolated precision-recall curve. Precision at each of the 11 standard recall levels is interpolated for each query from the maximum at any higher level of recall. The original measured precision recall points are also shown.

preferred. However, these curves can also provide insight into the overall behavior of a system. Systems that are higher in precision towards the left may favor precision over recall, while systems that are more geared towards recall will be higher at higher levels of recall (to the right).

MEAN AVERAGE PRECISION

A second popular way way to evaluate ranked retrieval systems is known as **mean average precision**(MAP). In this approach, we again descend through the ranked list of items and note the precision only at those points where a relevant item has been encountered. For a single query, we average these individual precision measurements over the return set up to some fixed cutoff. More formally, if we assume that R_r is the set of relevant documents at or above r, then the average precision for a single query is:

$$\frac{1}{|R_r|}\sum_{d\in R_r} Precision_r(d) \tag{23.16}$$

where $Precision_r(d)$ is the precision measured at the rank where document d was found. For an ensemble of queries, we then average over these averages, giving us our mean average precision measure. Applying this technique to the data in Fig. 23.5 yields a MAP measure of 0.6 for this single retrieval.

MAP has the advantage of providing a single crisp metric that can be used to compare competing systems or approaches. Note, that MAP will tend to favor systems that provide relevant documents at high ranks. Of course, this isn't really a problem since that is a big part of what we're looking for in a retrieval system. But since the measure essentially ignores recall, it can favor those systems that are tuned to return small sets of documents in which they are highly confident, at the expense of systems that attempt to be more comprehensive by trying to attain higher levels of recall.

The U.S. government-sponsored TREC (Text REtrieval Conference) evaluations, run annually since 1992, provide a rigorous testbed for the evaluation of a variety of information retrieval tasks and techniques. TREC provides large document sets for both training and testing, along with a uniform scoring system. Training materials consist of sets of documents accompanied by sets of queries (called topics in TREC) and relevance judgments. TREC subtasks over the years have included question answering, IR in Chinese and Spanish, interactive IR, retrieval from speech and video, and others. See Voorhees and Harman (2005). Details of all of the meetings can be found at the TREC page on the National Institute of Standards and Technology website.

23.1.5 Homonymy, Polysemy, and Synonymy

Since the vector space model is based solely on the use of simple terms, it is useful to consider the effect that various lexical semantic phenomena may have on the model. Consider a query containing the word *canine*, a word that has senses meaning something like *tooth* and *dog*. A query containing *canine* will be judged similar to documents making use of either of these senses. However, given that users are probably only interested in one of these senses, the documents containing the other sense will be judged non-relevant. Homonymy and polysemy, therefore, can have the effect of *reducing precision* by leading a system to return documents irrelevant to the user's information need.

Now consider a query consisting of the lexeme *dog*. This query will be judged close to documents that make frequent use of the term *dog*, but may fail to match documents that use close synonyms like *canine*, as well as documents that use hyponyms such as *Malamute*. Synonymy and hyponymy, therefore, can have the effect of *reducing recall* by causing the retrieval system to miss relevant documents.

Note that it is inaccurate to state flatly that polysemy reduces precision, and synonymy reduces recall since, as we discuss on page 11, both measures are relative to a fixed cutoff. As a result, every non-relevant document that rises above the cutoff due to polysemy takes up a slot in the fixed size return set, and may thus push a relevant document below threshold, thus reducing recall. Similarly, when a document is missed due to synonymy, a slot is opened in the return set for a non-relevant document, potentially reducing precision as well.

These issues lead naturally to the question of whether or not word sense disambiguation can help in information retrieval. The current evidence on this point is mixed, with some experiments reporting a gain using disambiguation-like techniques (Schütze and Pedersen, 1995), and others reporting either no gain, or a degradation in performance (Krovetz and Croft, 1992; Sanderson, 1994; Voorhees, 1998).

23.1.6 Improving User Queries

One of the most effective ways to improve retrieval performance is to find a way to improve user queries. The techniques presented in this section have been shown to varying degrees to be effective at this task.

RELEVANCE FEEDBACK

The single most effective way to improve retrieval performance in the vector space model is the use of **relevance feedback** (Rocchio, 1971). In this method, a user presents a query to the system and is presented with a small set of retrieved documents. The user is then asked to specify which of these documents appears relevant to their need. The user's original query is then reformulated based on the distribution of terms in the relevant and non-relevant documents that the user examined. This reformulated query is then passed to the system as a *new* query with the new results being shown to the user. Typically an enormous improvement is seen after a single iteration of this technique.

The formal basis for the implementation of this technique falls out directly from some of the basic geometric intuitions of the vector model. In particular, we would like to *push* the vector representing the user's original query toward the documents that have been found to be relevant, and away from the documents judged not relevant. This can be accomplished by adding an averaged vector representing the relevant documents to the original query, and subtracting an averaged vector representing the non-relevant documents.

More formally, let's assume that $\vec{q}_i$ represents the user's original query, R is the number of relevant documents returned from the original query, S is the number of non-relevant documents, and documents in the relevant and non-relevant sets are denoted as $\vec{r}$ and $\vec{s}$, respectively. In addition, assume that β and γ range from 0 to 1 and that $\beta + \gamma = 1$. Given these assumptions, the following represents a standard relevance feedback update formula:

$$\vec{q}_{i+1} = \vec{q}_i + \frac{\beta}{R}\sum_{j=1}^{R}\vec{r}_j - \frac{\gamma}{S}\sum_{k=1}^{S}\vec{s}_k$$

The factors β and γ in this formula represent parameters that can be adjusted experimentally. Intuitively, β represents how far the new vector should be pushed towards the relevant documents, and γ represents how far it should be pushed away from the non-relevant ones. Salton and Buckley (1990) report good results with $\beta = .75$ and $\gamma = .25$.

RESIDUAL COLLECTION

We should note that evaluating systems that use relevance feedback is rather tricky. In particular, an enormous improvement is often seen in the documents retrieved by the first reformulated query. This should not be too surprising since it includes the documents that the user told the system were relevant on the first round. The preferred way to avoid this inflation is to only compute recall and precision measures for what is called the **residual collection**, the original collection without any of the documents shown to the user on any previous round. This usually has the effect of driving the system's raw performance below that achieved with the first query, since the most highly relevant documents have now been eliminated. Nevertheless, this is an effective technique to use when comparing distinct relevance feedback mechanisms.

QUERY EXPANSION

An alternative approach to query improvement focuses on terms that comprise the query vector. In **query expansion**, the user's original query is expanded by adding terms that are synonymous with or related to the original terms. Query expansion is thus a technique for improving recall, perhaps at the expense of precision. For example the query *Steve Jobs* could be expanded by adding terms like *Apple*, *Macintosh*, and *personal computer*.

THESAURUS

THESAURUS GENERATION

TERM CLUSTERING

The terms to be added to the query are taken from a **thesaurus**. It is possible to use a hand-built resource like WordNet or UMLS as the thesaurus for query expansion, when the domain is appropriate. But often these thesauruses are not suitable for the collection, and instead, we do **thesaurus generation**, generating a thesaurus automatically from documents in the collection. We can do this by clustering the words in the collection, a method known as **term clustering**. Recall from our characterization of the term-by-document matrix that the columns in the matrix represent the documents and the rows represent the terms. Thus, in thesaurus generation, the rows can be clustered to form sets of synonyms, which can then be added to the user's original query to improve its recall. The distance metric for clustering can be simple cosine, or any of the other distributional methods for word relatedness discussed in Ch. 20.

The thesaurus can be generated once from the document collection as a whole (Crouch and Yang, 1992), or sets of synonym-like terms can be generated dynamically from the returned set for the original query (Attar and Fraenkel, 1977). Note that this second approach entails far more effort, since in effect a small thesaurus is generated for the documents returned for every query, rather than once for the entire collection.

23.2 Factoid Question Answering

There are many situations where the user wants a particular piece of information rather than an entire document or document set. We use the term **question answering** for the task of returning a particular piece of information to the user in response to a question. We call the task **factoid question answering** if the information is a simple fact, and particularly if this fact has to do with a **named entity** like a person, organization, or location.

The task of a factoid question answering system is thus to answer questions by finding, either from the Web or some other collection of documents, short text segments that are likely to contain answers to questions, reformatting them, and presenting them to the user. Fig. 23.7 shows some sample factoid questions together with their answers.

Question	Answer
Where is the Louvre Museum located?	in Paris, France
What's the abbreviation for limited partnership?	L.P.
What are the names of Odin's ravens?	Huginn and Muninn
What currency is used in China?	the yuan
What kind of nuts are used in marzipan?	almonds
What instrument does Max Roach play?	drums
What's the official language of Algeria?	Arabic
What is the telephone number for the University of Colorado, Boulder?	(303)492-1411
How many pounds are there in a stone?	14

Figure 23.7 Some sample factoid questions and their answers.

Since factoid question answering is based on information retrieval techniques to find these segments, it is subject to the same difficulties as information retrieval. That is, the fundamental problem in factoid question answering is the gap between the way that questions are posed and the way that answers are expressed in a text. Consider the following question/answer pair from the TREC question answering task:

(23.17) *User Question:* What company sells the most greeting cards?

(23.18) *Potential Document Answer:* Hallmark remains the largest maker of greeting cards.

Here the user uses the verbal phrase *sells the most* while the document segment uses a nominal *the largest maker*. The solution to the possible mismatches between question and answer form lies in the ability to robustly process *both* questions and candidate answer texts in such a way that a measure of similarity between the question and putative answers can be performed. As we'll see, this process involves many of the techniques that we have introduced in earlier chapters including limited forms of morphological analysis, part-of-speech tagging, syntactic parsing, semantic role labelling, named-entity recognition, and information retrieval.

Because it is impractical to employ these relatively expensive NLP techniques like parsing or role labeling on vast amounts of textual data, question answering systems generally use information retrieval methods to first retrieve a smallish number of potential documents. The most expensive techniques then used in a second pass on these smaller numbers of candidate relevant texts.

Fig. 23.8 shows the three phases of a modern factoid question answering system: question processing, passage retrieval and ranking, and answer processing.

23.2.1 Question Processing

The goal of the question processing phase is to extract two things from the question: a keyword **query** suitable as input to an IR system and an **answer type**, a specification of the kind of entity that would constitute a reasonable answer to the question.

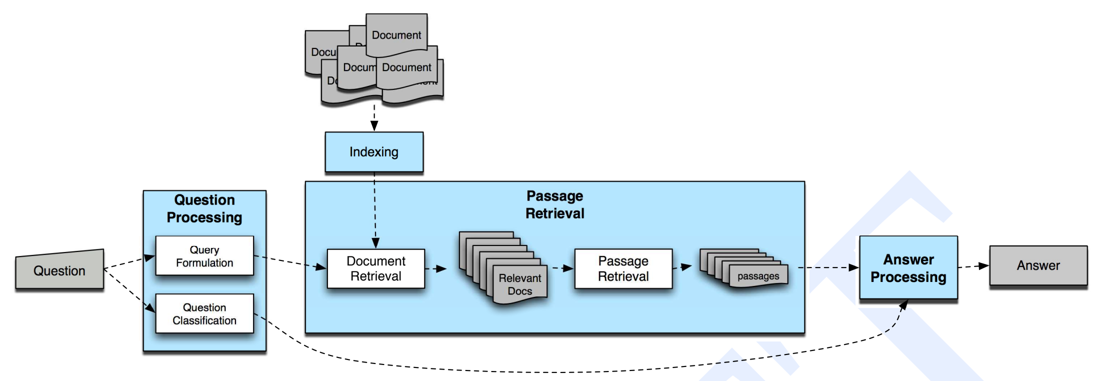


Figure 23.8 The 3 stages of a generic question answering system: question processing, passage retrieval, and answer processing..

Query Formulation

The process of **query formulation** is very similar to the processing done on other IR queries. Our goal is to create from the question a list of keywords that forms an IR query.

Exactly what query to form depends on the question answering application. If question answering is applied to the Web, we might simply create a keyword from every word in the question, letting the web search engine automatically remove any stopwords. Often we leave out the question word (*where*, *when*, etc). Alternatively, keywords can be formed from only the terms found in the noun phrases in the question, applying stopword lists to ignore function words and high-frequency, low-content verbs.

When question answering is applied to smaller sets of documents, for example to answer questions about corporate information pages, we still use an IR engine to search our documents for us. But for this smaller set of documents we generally need to apply query expansion. On the Web the answer to a question might appear in many different forms, and so if we search with words from the question we'll probably find an answer written in the same form. In smaller sets of corporate pages, by contrast, an answer might appear only once, and the exact wording might look nothing like the question. Thus query expansion methods can add query terms hoping to match the particular form of the answer as it appears.

Thus we might add to the query all morphological variants of the content words in the question, as well as applying the thesaurus-based or other query expansion algorithms discussed in the previous section to get a larger set of keywords for the query. Many systems use WordNet as a thesaurus, while others rely on special-purpose thesauruses that are specifically hand-built for question-answering.

QUERY REFORMULATION

Another query formulation approach that is sometimes used when questioning the Web is to apply a set of **query reformulation** rules to the query. The rules rephrase the question to make it look like a substring of possible declarative answers. For example the question *"when was the laser invented?"* would be reformulated as *the laser was* *invented*; the question *“where is the Valley of the Kings?”* might be reformulated as *“the Valley of the Kings is located in”*. We can apply multiple such rules to the query, and pass all the resulting reformulated queries to the web search engine. Here are some sample hand-written reformulation rules from Lin (2007):

(23.19) *wh-word* did A *verb* B → . . . A *verb*+ed B

(23.20) Where is A → A is located in

Question Classification

ANSWER TYPE

QUESTION CLASSIFICATION

ANSWER TYPE RECOGNITION

The second task in question processing is to classify the question by its expected **answer type**. For example a question like *“Who founded Virgin Airlines”* expects an answer of type PERSON. A question like *“What Canadian city has the largest population?”* expects an answer of type CITY. This task is called **question classification** or **answer type recognition**. If we know the answer type for a question, we can avoid looking at every sentence or noun phrase in the entire suite of documents for the answer, instead focusing on, e.g., just people or cities. Knowing an answer type is also important for presenting the answer. A DEFINITION question like *“What is a prism”* might use a simple answer template like *“A prism is. . . ”* while an answer to a BIOGRAPHY question like *“Who is Zhou Enlai?”* might use a biography-specific template, perhaps beginning with the persons nationality and proceeding to their dates of birth and other biographical information.

ANSWER TYPE TAXONOMY

QUESTION ONTOLOGY

As some of the above examples suggest, we might draw the set of possible answer types for a question classifier from a set of named entities like the PERSON, LOCATION, and ORGANIZATION described in Ch. 22. Usually, however, a somewhat richer set of answer types is used. These richer tagsets are often hierarchical, and so we usually call them an **answer type taxonomy** or a **question ontology**. Such taxonomies can be built semi-automatically and dynamically, for example from WordNet (Harabagiu et al., 2000; Pasca, 2003), or they can be designed by hand.

Fig. 23.9 shows one such hand-built ontology, the hierarchical Li and Roth (2005) tagset. In this tagset, each question can be labeled with a coarse-grained tag like HUMAN, or a fine-grained tag like HUMAN:DESCRIPTION, HUMAN:GROUP, HUMAN:IND, and so on. Similar tags are used in other systems; the type HUMAN:DESCRIPTION is often called a BIOGRAPHY question, because the answer requires giving a brief biography of the person, rather than just a name.

Question classifiers can be built by hand-writing rules, via supervised machine learning, or via some combination. The Webclopedia QA Typology, for example, contains 276 hand-written rules associated with the approximately 180 answer types in the typology (Hovy et al., 2002). A regular expression rule for detecting an answer type like BIOGRAPHY (which assumes the question has been named-entity tagged) might be:

(23.21) who {is | was | are | were} PERSON

Most modern question classifiers, however, are based on supervised machine learning techniques. These classifiers are trained on databases of questions that have been hand-labeled with an answer type such as the corpus of Li and Roth (2002). Typical features used for classification include the words in the questions, the part-of-speech of each word, and named entities in the questions.

Tag	Example
ABBREVIATION	
abb	What's the abbreviation for limited partnership?
exp	What does the "c" stand for in the equation E=mc2?
DESCRIPTION	
definition	What are tannins ?
description	What are the words to the Canadian National anthem?
manner	How can you get rust stains out of clothing?
reason	What caused the Titanic to sink ?
ENTITY	
animal	What are the names of Odin's ravens?
body	What part of your body contains the corpus callosum ?
color	What colors make up a rainbow ?
creative	In what book can I find the story of Aladdin?
currency	What currency is used in China?
disease/medicine	What does Salk vaccine prevent ?
event	What war involved the battle of Chapultepec?
food	What kind of nuts are used in marzipan?
instrument	What instrument does Max Roach play?
lang	What's the official language of Algeria?
letter	What letter appears on the cold-water tap in Spain?
other	What is the name of King Arthur's sword?
plant	What are some fragrant white climbing roses?
product	What is the fastest computer ?
religion	What religion has the most members ?
sport	What was the name of the ball game played by the Mayans?
substance	What fuel do airplanes use?
symbol	What is the chemical symbol for nitrogen ?
technique	What is the best way to remove wallpaper?
term	How do you say " Grandma " in Irish ?
vehicle	What was the name of Captain Bligh's ship ?
word	What's the singular of dice?
HUMAN	
description	Who was Confucius?
group	What are the major companies that are part of Dow Jones ?
ind	Who was the first Russian astronaut to do a spacewalk?
title	What was Queen Victoria's title regarding India?
LOCATION	
city	What's the oldest capital city in the Americas ?
country	What country borders the most others?
mountain	What is the highest peak in Africa?
other	What river runs through Liverpool?
state	What states do not have state income tax?
NUMERIC	
code	What is the telephone number for the University of Colorado?
count	About how many soldiers died in World War II?
date	What is the date of Boxing Day?
distance	How long was Mao's 1930s Long March?
money	How much did a McDonald's hamburger cost in 1963?
order	Where does Shanghai rank among world cities in population?
other	What is the population of Mexico?
period	What was the average life expectancy during the Stone Age?
percent	
speed	What is the speed of the Mississippi River?
temp	How fast must a spacecraft travel to escape Earth's gravity?
size	What is the size of Argentina?
weight	How many pounds are there in a stone?

Figure 23.9 Question typology from Li and Roth (2002, 2005). Example sentences are from their corpus of 5500 labeled questions. A question can be labeled either with a coarse-grained tag like HUMAN or NUMERIC, or a fine-grained tag like HUMAN:DESCRIPTION, HUMAN:GROUP, HUMAN:IND, and so on.

Often a single word in the question gives extra information about the answer type, and its identity is used as a feature. This word is sometimes called the question **headword** or the **answer type word**, and may be defined as the headword of the first NP after the question's *wh-word*; headwords are indicated in boldface in the following examples:

(23.22) Which **city** in China has the largest number of foreign financial companies.

(23.23) What is the state **flower** of California?

Finally, it often helps to use semantic information about the words in the questions. The WordNet synset id of the word can be used as a feature, as can the ids of the hypernym and hyponyms of each word in the question.

In general question classification accuracies are relatively high on easy question types like PERSON, LOCATION, and TIME questions; detecting REASON and DESCRIPTION questions can be much harder.

23.2.2 Passage Retrieval

The query that was created in the question processing phase is next used to query an information retrieval system, either a general IR engine over a proprietary set of indexed documents or a web search engine. The result of this document retrieval stage is a set of documents.

Although the set of documents is generally ranked by relevance, the top-ranked document is probably not the answer to the question. This is because documents are not an appropriate unit to rank with respect to the goals of a question answering system. A highly relevant and large document that does not prominently answer a question is not an ideal candidate for further processing.

Therefore, the next stage is to extract a set of potential answer passages from the retrieved set of documents. The definition of a passage is necessarily system dependent, but the typical units include sections, paragraphs and sentences. For example, we might run a paragraph segmentation algorithm of the type discussed in Ch. 21 on all the returned documents and treat each paragraph as a segment.

PASSAGE RETRIEVAL

We next perform **passage retrieval**. In this stage we first filter out passages in the returned documents that don't contain potential answers, and then rank the rest according to how likely they are to contain an answer to the question. The first step in this process is to run a named entity or answer-type classification on the retrieved passages. The answer type that we determined from the question tells us the possible answer types (extended named entities) we expect to see in the answer. We can therefore filter out documents that don't contain any entities of the right type.

The remaining passages are then ranked; either via hand-crafted rules or supervised training with machine learning techniques. In either case, the ranking is based on a relatively small set of features that can be easily and and efficiently extracted from a potentially large number of answer passages. Among the more common features are:

- The number of **named entities** of the right type in the passage
- The number of **question keywords** in the passage
- The longest exact sequence of question keywords that occurs in the passage

- The rank of the document from which the passage was extracted
- The **proximity** of the keywords from the original query to each other:
  For each passage identify the shortest span that covers the keywords contained in that passage. Prefer smaller spans that include more keywords (Pasca, 2003; Monz, 2004).
- The *N*-**gram overlap** between the passage and the question:
  Count the *N*-grams in the question and the *N*-grams in the answer passages. Prefer the passages with higher *N*-gram overlap with the question (Brill et al., 2002).

For question answering from the Web, instead of extracting passages from all the returned documents, we can rely on the web search to do passage extraction for us. We do this by using **snippets** produced by the web search engine as the returned passages. For example, Fig. 23.10 shows some snippets for the first 5 document returned from the Google search engine for the query *When was movable type metal printing invented in Korea?*

23.2.3 Answer Processing

The final stage of question answering is to extract a specific answer from the passage, so as to be able to present the user with an answer like *300 million* to the question *"What is the current population of the United States"*.

Two classes of algorithms have been applied to the answer extraction task, one based on **answer-type pattern extraction** and one based on *N*-**gram tiling**.

In the **pattern extraction** methods for answer processing, we use information about the expected answer type together with regular expression patterns. For example, for questions with a HUMAN answer type we run the answer type or named entity tagger on the candidate passage or sentence, and return whatever entity is labeled with type HUMAN. Thus in the following examples, the underlined named entities are extracted from the candidate answer passages as the answer to the HUMAN and DISTANCE-QUANTITY questions:

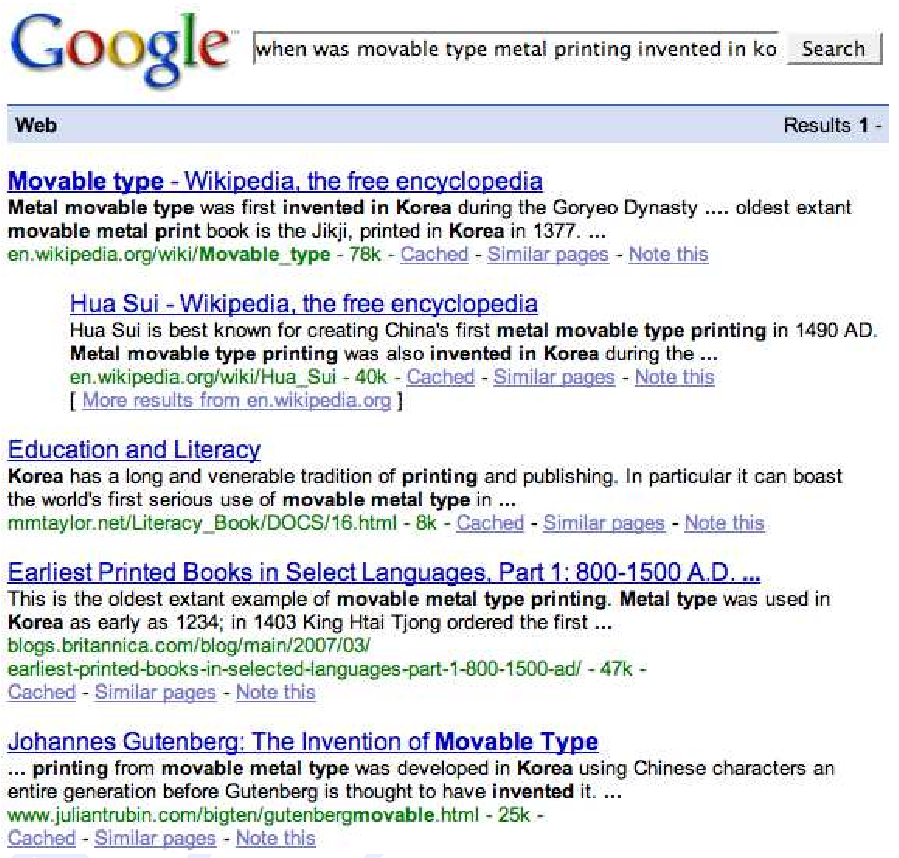
"Who is the prime minister of India"
> *Manmohan Singh, Prime Minister of India, had told left leaders that the deal would not be renegotiated.*

"How tall is Mt. Everest?
> *The official height of Mount Everest is 29035 feet*

Unfortunately, the answers to some questions, such as DEFINITION questions, don't tend to be of a particular named entity type. For some questions, then, instead of using answer types, we use handwritten regular expression patterns to help extract the answer. These patterns are also useful in cases where a passage contains multiple examples of the same named entity type. Fig. 23.11 shows some patterns from Pasca (2003) for the question phrase (QP) and answer phrase (AP) of definition questions.

The patterns are specific to each question type, and can either be written by hand or learned automatically.

The automatic pattern learning method of Ravichandran and Hovy (2002), Echihabi et al. (2005), for example, makes use of the pattern-based methods for relation extrac-



Movable type - Wikipedia, the free encyclopedia
Metal movable type was first **invented in Korea** during the Goryeo Dynasty .... oldest extant **movable metal print** book is the Jikji, printed in **Korea** in 1377. ...
en.wikipedia.org/wiki/**Movable_type** - 78k - Cached - Similar pages - Note this

Hua Sui - Wikipedia, the free encyclopedia
Hua Sui is best known for creating China's first **metal movable type printing** in 1490 AD. **Metal movable type printing** was also **invented in Korea** during the ...
en.wikipedia.org/wiki/Hua_Sui - 40k - Cached - Similar pages - Note this
[More results from en.wikipedia.org]

Education and Literacy
Korea has a long and venerable tradition of **printing** and publishing. In particular it can boast the world's first serious use of **movable metal type** in ...
mmtaylor.net/Literacy_Book/DOCS/16.html - 8k - Cached - Similar pages - Note this

Earliest Printed Books in Select Languages, Part 1: 800-1500 A.D. ...
This is the oldest extant example of **movable metal type printing**. **Metal type** was used in **Korea** as early as 1234; in 1403 King Htai Tjong ordered the first ...
blogs.britannica.com/blog/main/2007/03/earliest-printed-books-in-selected-languages-part-1-800-1500-ad/ - 47k -
Cached - Similar pages - Note this

Johannes Gutenberg: The Invention of **Movable Type**
... **printing** from **movable metal type** was developed in **Korea** using Chinese characters an entire generation before Gutenberg is thought to have **invented** it. ...
www.juliantrubin.com/bigten/gutenberg**movable**.html - 25k -
Cached - Similar pages - Note this

Figure 23.10 Five snippets from Google in response to the query *When was movable type metal printing invented in Korea?*

Pattern	Question	Answer
<AP> such as <QP>	What is autism?	", developmental disorders such as autism"
<QP> (an <AP>)	What is a caldera?	"the Long Valley caldera, a volcanic crater 19 miles long"

Figure 23.11 Some answer extraction patterns for definition questions (Pasca, 2003).

tion we introduced in Ch. 20 and Ch. 22 (Brin, 1998; Agichtein and Gravano, 2000). The goal of the pattern learning method is to learn a relation between a particular answer type such as YEAR-OF-BIRTH, and a particular aspect of the question, in this case the name of the person whose birth year we want. We are thus trying to learn patterns which are good cues for a relation between two phrases (PERSON-NAME/YEAR-OF-BIRTH, or TERM-TO-BE-DEFINED/DEFINITION, etc). This task is thus very similar to the task of learning hyponym/hyponym relations between WordNet synsets introduced in Ch. 20 , or learning ACE relations between words from Ch. 22. Here is a sketch of the algorithm as applied to question-answer relation extraction:

1. For a given relation between two terms (i.e. person-name→year-of-birth), we start with a hand-built list of correct pairs (e.g., “gandhi:1869”, “mozart:1756”, etc).
2. Now query the Web with instances of these pairs (e.g., ”gandhi” and ”1869”, etc) and examine the top X returned documents.
3. Break each document into sentences, and keep only sentences containing both terms (e.g., PERSON-NAME and BIRTH-YEAR).
4. Extract a regular expression pattern representing the words and punctuation that occur between and around the two terms.
5. Keep all patterns that are sufficiently high-precision.

In Ch. 20 and Ch. 22 we discussed various ways to measure accuracy of the patterns. A method used in question-answer pattern matching is to keep patterns which are **high-precision**. Precision is measured by performing a query with only the question terms, but not the answer terms (i.e. query with just “gandhi” or “mozart”). We then run the resulting patterns on the sentences from the document, and extract a birth-date. Since we know the correct birth-date, we can compute the percentage of times this pattern produced a correct birthdate. This percentage is the precision of the pattern.

For the YEAR-OF-BIRTH answer type, this method learns patterns like the following:

```
<NAME> (<BD>-<DD>)
<NAME> (<BD>-<DD>),
<NAME> was born on <BD>
```

These two methods, named entity detection and question-answer pattern extraction, are still not sufficient for answer extraction. Not every relation is signaled by unambiguous surrounding words or punctuation, and often multiple instances of the same named-entity type occur in the answer passages. The most successful answer-extraction method is thus to combine all these methods, using them together with other information as features in a classifier that ranks candidate answers. We extract potential answers using named entities or patterns or even just looking at every sentence returned from passage retrieval, and rank them using a classifier with features like the following:

Answer type match: True if the candidate answer contains a phrase with the correct answer type.

Pattern match: The identity of a pattern that matches the candidate answer.

Number of matched question keywords: How many question keywords are contained in the candidate answer.

Keyword distance: The distance between the candidate answer and query keywords (measured in average number of words, or as the number of keywords that occur in the same syntactic phrase as the candidate answer.

Novelty factor: True if at least one word in the candidate answer is novel, i.e. not in the query.

Apposition features: True if the candidate answer is an appositive to a phrase containing many question terms. Can be approximated by the number of question terms separated from the candidate answer through at most three words and one comma Pasca (2003).

Punctuation location: True if the candidate answer is immediately followed by a comma, period, quotation marks, semicolon, or exclamation mark.

Sequences of question terms: The length of the longest sequence of question terms that occurs in the candidate answer.

N-GRAM TILING

N-GRAM MINING

N-GRAM FILTERING

An alternative approach to answer extraction, used solely in web search, is based on *N***-gram tiling**, sometimes called the **redundancy-based approach** (Brill et al., 2002; Lin, 2007). This simplified method begins with the snippets returned from the web search engine, produced by a reformulated query. In the first step of the method, *N***-gram mining**, every unigram, bigram, and trigram occurring in the snippet is extracted and weighted. The weight is a function of the number of snippets the *N*-gram occurred in, and the weight of the query reformulation pattern that returned it. In the *N***-gram filtering** step, *N*-grams are scored by how well they match the predicted answer type. These scores are computed by hand-written filters built for each answer type. Finally, an *N***-gram tiling** algorithm concatenates overlapping *N*-gram fragments into longer answers. A standard greedy method is to start with the highest-scoring candidate and try to tile each other candidate with this candidate. The best scoring concatenation is added to the set of candidates, the lower scoring candidate is removed, and the process continues until a single answer is built.

For any of these answer extraction methods, the exact answer phrase can just be presented to the user by itself. In practice, however, users are rarely satisfied with an unadorned number or noun as an answer; they prefer to see the answer accompanied by enough passage information to substantiate the answer. Thus we often give the user an entire passage with the exact answer inside it highlighted or boldfaced.

23.2.4 Evaluation of Factoid Answers

A wide variety of techniques have been employed to evaluate question answering systems. By far the most influential evaluation framework has been provided by the TREC Q/A track first introduced in 1999.

MEAN RECIPROCAL RANK

MRR

The primary measure used in TREC is an **intrinsic** or **in vitro** evaluation metric known as **mean reciprocal rank**, or **MRR**. As with the ad hoc information retrieval task described in Sec. 23.1, MRR assumes a test set of questions that have been human-labeled with correct answers. MRR also assumes that systems are returning a short **ranked** list of answers, or passages containing answers. Each question is then scored based on the reciprocal of the **rank** of the first correct answer. For example if the system returned 5 answers but the first 3 are wrong and hence the highest-ranked correct answer is ranked 4, the reciprocal rank score for that question would be $\frac{1}{4}$. Questions with return sets that that do not contain any correct answers are assigned a zero. The score of a system is then the average of the score for each question in the set. More formally, for an evaluation of a system returning *M* ranked answers for test set consisting of *N* questions, the MRR is defined as:

$$\mathrm{MRR} = \frac{\sum_{i=1}^{N} \frac{1}{rank_i}}{N} \tag{23.24}$$

23.3 Summarization

The algorithms we have described so far in this chapter present the user an entire document (information retrieval), or a short factoid answer phrase (factoid question answering). But sometimes the user wants something that lies in between these extremes: something like a **summary** of a document or set of documents.

TEXT SUMMARIZATION

Text summarization is *the process of distilling the most important information from a text to produce an abridged version for a particular task and user* (definition adapted from Mani and Maybury (1999)). Important kinds of summaries that are the focus of current research include:

- **outlines** of any document
- **abstracts** of a scientific article
- **headlines** of a news article
- **snippets** summarizing a web page on a search engine results page
- **action items or other summaries** of a (spoken) business meeting
- **summaries** of email threads
- **compressed sentences** for producing simplified or compressed text
- **answers** to complex questions, constructed by summarizing multiple documents

These kinds of summarization goals are often characterized by their position on two dimensions:

- **single document** versus **multiple document** summarization
- **generic** summarization versus **query-focused** summarization

SINGLE DOCUMENT SUMMARIZATION

In **single document summarization** we are given a single document and produce a summary. Single document summarization is thus used in situations like producing a headline or an outline, where the final goal is to characterize the content of a single document.

MULTIPLE DOCUMENT SUMMARIZATION

In **multiple document summarization**, the input is a group of documents, and our goal is to produce a condensation of the content of the entire group. We might use multiple document summarization when we are summarizing a series of news stories on the same event, or whenever we have web content on the same topic that we'd like to synthesize and condense.

GENERIC SUMMARY

QUERY-FOCUSED SUMMARIZATION

FOCUSED SUMMARIZATION

A **generic summary** is one in which we don't consider a particular user or a particular information need; the summary simply gives the important information in the document(s). By contrast, in **query-focused summarization**, also called **focused summarization**, **topic-based summarization** and **user-focused summarization**, the summary is produced in response to a user query. We can think of query-focused summarization as a kind of longer, non-factoid answer to a user question.

In the remainder of this section we give a brief overview of the architecture of automatic text summarization systems; the following sections then give details.

EXTRACT

ABSTRACT

One crucial architectural dimension for text summarizers is whether they are producing an **abstract** or an **extract**. The simplest kind of summary, an **extract**, is formed by selecting (**extracting**) phrases or sentences from the document to be summarized and pasting them together. By contrast, an **abstract** uses different words to describe the contents of the document. We'll illustrate the difference between an extract and an abstract using the well-known Gettysburg address, a famous speech by Abraham Lincoln, shown in Fig. 23.12.[1] Fig. 23.13 shows an extractive summary from the speech followed by an abstract of the speech.

> Fourscore and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battle- field of that war. We have come to dedicate a portion of that field as a final resting-place for those who here gave their lives that this nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we cannot dedicate...we cannot consecrate...we cannot hallow... this ground. The brave men, living and dead, who struggled here, have consecrated it far above our poor power to add or detract. The world will little note nor long remember what we say here, but it can never forget what they did here. It is for us, the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us...that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion; that we here highly resolve that these dead shall not have died in vain; that this nation, under God, shall have a new birth of freedom; and that government of the people, by the people, for the people, shall not perish from the earth.

Figure 23.12 The Gettysburg Address. Abraham Lincoln, 1863.

Most current text summarizers are extractive, since extraction is much easier than abstracting; the transition to more sophisticated abstractive summarization is a key goal of recent research.

Text summarization systems and, as it turns out, **natural language generation** systems as well, are generally described by their solutions to the following three problems:

1. **Content Selection:** What information to select from the document(s) we are summarizing. We usually make the simplifying assumption that the granularity of extraction is the sentence or clause. Content selection thus mainly consists of choosing which sentences or clauses to extract into the summary.
2. **Information Ordering:** How to order and structure the extracted units.
3. **Sentence Realization:** What kind of clean up to perform on the extracted units so they are fluent in their new context.

[1] In general one probably wouldn't need a summary of such a short speech, but a short text makes it easier to see how the extract maps to the original for pedagogical purposes. For an amusing alternative application of modern technology to the Gettysburg Address, see Norvig (2005).

Extract from the Gettysburg Address:

Four score and seven years ago our fathers brought forth upon this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation can long endure. We are met on a great battlefield of that war. We have come to dedicate a portion of that field. But the brave men, living and dead, who struggled here, have consecrated it far above our poor power to add or detract. From these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion — that government of the people, by the people for the people shall not perish from the earth.

Abstract of the Gettysburg Address:

This speech by Abraham Lincoln commemorates soldiers who laid down their lives in the Battle of Gettysburg. It reminds the troops that it is the future of freedom in America that they are fighting for.

Figure 23.13 An extract versus an abstract from the Gettysburg Address (abstract from Mani (2001)).

In the next sections we'll show these components in three summarization tasks: **single document** summarization, **multiple document** summarization, and **query-focused summarization**.

23.3.1 Summarizing Single Documents

Let's first consider the task of building an extractive summary for a single document. Assuming that the units being extracted are at the level of the sentence, the three summarization stages for this task are:

1. **Content Selection:** Choose sentences to extract from the document
2. **Information Ordering:** Choose an order to place these sentences in the summary
3. **Sentence Realization:** Clean up the sentences, for example by removing nonessential phrases from each sentence, or fusing multiple sentences into a single sentence, or by fixing problems in coherence.

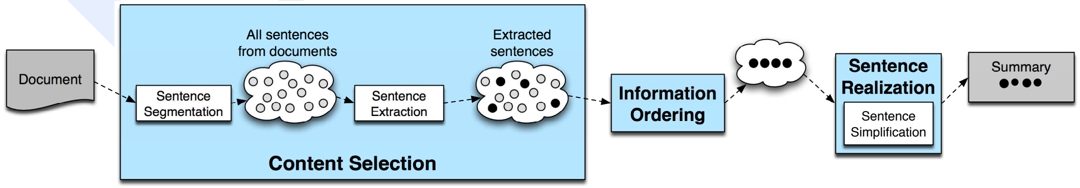


Figure 23.14 The basic architecture of a generic single document summarizer.

We'll first describe basic summarization techniques with only one of these components: *content selection*. Indeed, many single document summarizers have no information ordering component, simply ordering the extracted sentences in the order they appeared in the original document. In addition, we'll assume for now that sentences are not combined or cleaned up after they are extracted, although we'll briefly mention later how this is done.

Unsupervised Content Selection

CONTENT SELECTION

The **content selection** task of extracting sentences is often treated as a classification task. The goal of the classifier is to label each sentence in a document with a binary label: *important* versus *unimportant* (or *extract-worthy* versus *not extractworthy*). We begin with some unsupervised algorithms for sentence classification and then turn to supervised algorithms in the next section.

TOPIC SIGNATURE

SIGNATURE TERMS

The simplest unsupervised algorithm, based on an intuition that dates back to the early summarizer of (Luhn, 1958), is to select sentences that have more **salient** or **informative** words. Sentences that contain more informative words tend to be more extract-worthy. Saliency is usually defined by computing the **topic signature**, a set of **salient** or **signature terms**, each of whose saliency scores is greater than some threshold θ.

Saliency could be measured in terms of simple word frequency, but frequency has the problem that a word might have a high probability in English in general but not be particularly topical to a particular document. Therefore weighting schemes like **tf-idf** or **log-likelihood ratio** are more often used.

Recall from page 8 that the tf-idf scheme gives a high weight to words that appear frequently in the current document, but rarely in the overall document collection, suggesting that the word is particularly relevant to this document. For each term i that occurs in the sentence to be evaluated, we compute its count in the current document j $\mathrm{tf}_{i,j}$, and multiply by the inverse document frequency over the whole collection idf_i:

$$weight(w_i) = \mathrm{tf}_{i,j} \times \mathrm{idf}_i \tag{23.25}$$

LOG LIKELIHOOD RATIO

A better performing method for finding informative words is **log likelihood ratio** (LLR). The log likelihood ratio for a word, generally called $\lambda(w)$, is the ratio between the probability of observing w both in the input and in the background corpus assuming equal probabilities in both corpora, and the probability of observing w in both assuming different probabilities for w in the input and the background corpus. See Dunning (1993), Moore (2004)and Manning and Schütze (1999) for details on log likelihood and how it is calculated.

It turns out for log likelihood ratio that the quantity $-2\log(\lambda)$ is asymptotically well approximated by the χ^2 distribution, which means that a word appears in the input significantly more often than in the background corpus (at $\alpha = 0.001$) if $-2log(\lambda) > 10.8$. Lin and Hovy (2000) first suggested that this made log likelihood ratio particularly appropriate for selecting a topic signature for summarization. Thus the word weight with log likelihood ratio is generally defined as follows:

$$weight(w_i) = \begin{cases} 1 & \text{if } -2\log(\lambda(w_i)) > 10 \\ 0 & \text{otherwise.} \end{cases} \tag{23.26}$$

Equation (23.26) is used to set a weight of 1 or 0 for each word in the sentence. The score for a sentence s_i is then the average weight of its non-stop words:

$$weight(s_i) = \sum_{w \in s_i} \frac{weight(w)}{|\{w | w \in s_i\}|} \tag{23.27}$$

The summarization algorithms computes this weight for every sentence, and then ranks all sentences by their score. The extracted summary consists of the top ranked sentences.

The family of algorithms that this thresholded LLR algorithm belongs to is called **centroid-based summarization** because we can view the set of signature terms as a pseudo-sentence which is the 'centroid' of all the sentences in the document and we are looking for sentences which are as close as possible to this centroid sentence.

CENTRALITY

A common alternative to the log likelihood ratio/centroid method is to use a different model of sentence **centrality**. These other centrality based methods resemble the centroid method described above, in that their goal is to rank the input sentences in terms of how central they are in representing the information present in the document. But rather than just ranking sentences by whether they contain salient words, centrality based methods compute distances between each candidate sentence and each other sentence and choose sentences that are on average closer to other sentences. To compute centrality, we can represent each sentence as a bag-of-words vector of length N as described in Ch. 20. For each pair of sentences x and y, we compute the tf-idf weighted cosine as described in Equation (23.12) above.

Each of the k sentences in the input is then assigned a centrality score which is its average cosine with all other sentences:

$$centrality(x) = \frac{1}{K} \sum_{y} \textit{tf-idf-cosine}(x,y) \tag{23.28}$$

Sentences are ranked by this centrality score, and the sentence which has the highest average cosine across all pairs, i.e. is most like other sentences, is chosen as the most 'representative' or 'topical' of all the sentences in the input.

It is also possible to extend this centrality score to use more complex graph-based measures of centrality like PageRank (Erkan and Radev, 2004).

Unsupervised Summarization based on Rhetorical Parsing

The sentence extraction algorithm we introduced above for content extraction relied solely on a single shallow feature, word saliency, ignoring possible higher-level cues such as discourse information. In this section we briefly summarize a way to get more sophisticated discourse knowledge into the summarization task.

The summarization algorithm we'll describe makes use of **coherence relations** such as the RST (rhetorical structure theory) relations described in Ch. 21. Recall that RST relations are often expressed in terms of a **satellite** and a **nucleus**; nucleus sentence are more likely to be appropriate for a summary. For example, consider the following two paragraphs taken from the Scientific American magazine text that we introduced in Fig. **??**:

> With its distant orbit – 50 percent farther from the sun than Earth – and slim atmospheric blanket, Mars experiences frigid weather conditions. Surface temperatures typically average about -70 degrees Fahrenheit at the equator, and can dip to -123 degrees C near the poles.
>
> Only the midday sun at tropical latitudes is warm enough to thaw ice on occasion, but any liquid water formed in this way would evaporate almost instantly because of the low atmospheric pressure. Although the atmosphere holds a small amount of water, and water-ice clouds sometimes develop, most Martian weather involves blowing dust or carbon dioxide.

The first two discourse units in this passage are related by the RST JUSTIFICATION relation, with the first discourse unit justifying the second unit, as shown in Fig. 23.15. The second unit (*"Mars experiences frigid weather conditions"*) is thus the nucleus, and captures better what this part of the document is about.

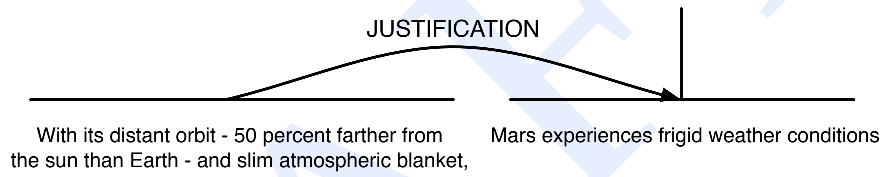

JUSTIFICATION
With its distant orbit - 50 percent farther from the sun than Earth - and slim atmospheric blanket,
Mars experiences frigid weather conditions

Figure 23.15 The justification relation between two discourse units, a satellite (on the left) and a nucleus (on the right).

We can use this intuition for summarization by first applying a discourse parser of the type discussed in Ch. 21 to compute the coherence relations between each discourse unit. Once a sentence has been parsed into a coherence relation graph or parse tree, we can use the intuition that the nuclear units are important for summarization by recursively extracting the salient units of a text.

Consider the coherence parse tree in Fig. 23.16. The salience of each node in the tree can be defined recursively as follows:

- Base case: The salient unit of a leaf node is the leaf node itself
- Recursive case: The salient units of an intermediate node are the union of the salient units of its immediate *nuclear* children

By this definition, discourse unit (2) is the most salient unit of the entire text (since the root node spanning units 1-8 has the node spanning units 1-6 as its nucleus, and unit 2 is the nucleus of the node spanning units 1-6.)

If we rank each discourse unit by the height of the nodes that it is the nucleus of, we can assign a partial ordering of salience to units; the algorithm of Marcu (1995) assigns the following partial ordering to this discourse:

$$2 > 8 > 3 > 1,4,5,7 > 6 \tag{23.29}$$

See Marcu (1995, 2000) for the details of exactly how this partial order is computed, and Teufel and Moens (2002) for another method for using rhetorical structure in summarization.

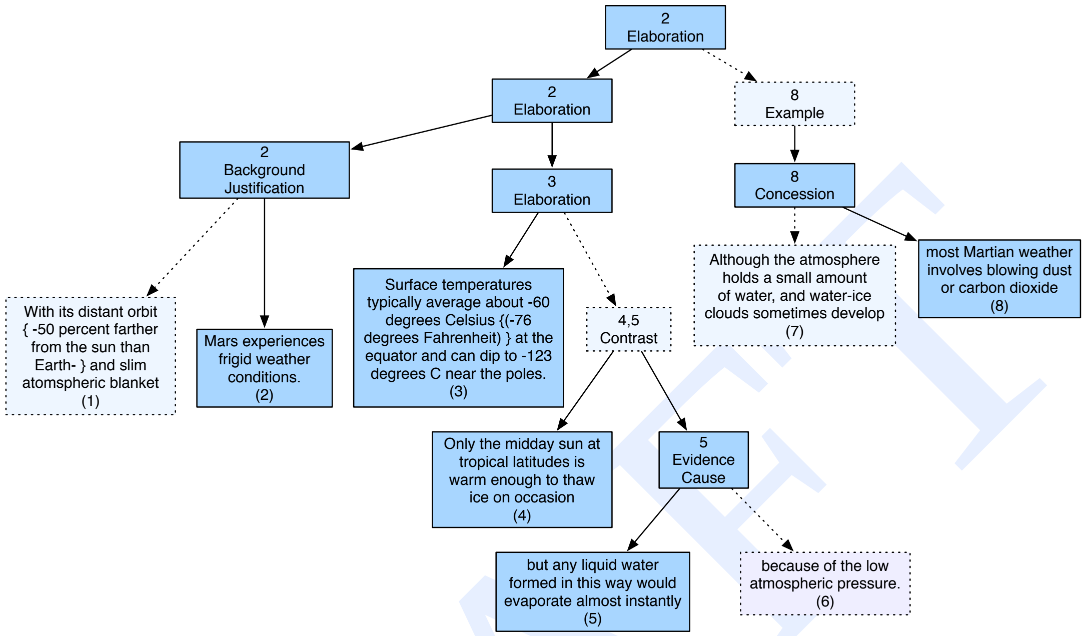

2
Elaboration
2
Elaboration
8
Example
2
Background
Justification
3
Elaboration
8
Concession
Although the atmosphere holds a small amount of water, and water-ice clouds sometimes develop
(7)
most Martian weather involves blowing dust or carbon dioxide
(8)
Surface temperatures typically average about -60 degrees Celsius {(-76 degrees Fahrenheit) } at the equator and can dip to -123 degrees C near the poles.
(3)
4,5
Contrast
With its distant orbit { -50 percent farther from the sun than Earth- } and slim atomspheric blanket
(1)
Mars experiences frigid weather conditions.
(2)
Only the midday sun at tropical latitudes is warm enough to thaw ice on occasion
(4)
5
Evidence
Cause
but any liquid water formed in this way would evaporate almost instantly
(5)
because of the low atmospheric pressure.
(6)

Figure 23.16 The discourse tree for the text on page 29. Boldface links connect nodes to their nuclei children; dotted lines to the satellite children. After Marcu (1995).

Supervised Content Selection

While the use of topic signatures for unsupervised content selection is an extremely effective method, topic signatures is only a single cue for finding extractworthy sentences. Many other cues exist, including the alternative saliency methods discussed above like centrality and PageRank mathods, as well as other cues like the position of the sentence in the document (sentences at the very beginning or end of the document tend to be more important), the length of each sentence, and so on. We'd like a method that can weigh and combine all of these cues.

The best principled method for weighing and combining evidence is supervised machine learning. For supervised machine learning, we'll need a training set of documents paired with human-created summary extracts, such as the Ziff-Davis corpus (Marcu, 1999). Since these are *extracts*, each sentence in the summary is, by definition, taken from the document. That means we can assign a label to every sentence in the document; **1** if it appears in the extract, **0** if it doesn't. To build our classifier, then, we just need to choose features to extract which are predictive of being a good sentence to appear in a summary. Some of the features commonly used in sentence classification are shown in Fig. 23.17.

Each sentence in our training document thus has a label (0 if the sentence is not in the training summary for that document, 1 if it is) and set of extracted feature values like those in Fig. 23.17. We can then train our classifier to estimate these labels for unseen data; for example a probabilistic classifier like naive Bayes or MaxEnt would

position	The position of the sentence in the document. For example Hovy and Lin (1999) found that the single most extract-worthy sentence in most newspaper articles is the title sentence. In the Ziff-Davis corpus they examined, the next most informative was the first sentence of paragraph 2 (P1S1), followed by the first sentence of paragraph 3 (P3S1); thus the list of ordinal sentence positions starting from the most informative was: T1, P2S1, P3S1, P4S1, P1S1, P2S2,... Position, like almost all summarization features, is heavily genre-dependent. In Wall Street Journal articles, they found the most important information appeared in the following sentences: T1, P1S1, P1S2,...
cue phrases	Sentences containing phrases like *in summary*, *in conclusion*, or *this paper* are more likely to be extract-worthy. These cue phrases are very dependent on the genre. For example in British House of Lords legal summaries, the phrase *it seems to me that* is a useful cue phrase. (Hachey and Grover, 2005).
word informativeness	Sentences that contain more terms from the **topic signature**, as described in the previous section, are more extractworthy.
sentence length	Very short sentences are rarely appropriate for extracting. We usually capture this fact by using a binary feature based on a cutoff (true if the sentence has more than, say, 5 words).
cohesion	Recall from Ch. 21 that a **lexical chain** is a series of related words that occurs throughout a discourse. Sentences which contain more terms from a lexical chain are often extractworthy because they are indicative of a continuing topic. (Barzilay and Elhadad, 1997). This kind of cohesion can also be computed by graph-based methods (Mani and Bloedorn, 1999). The PageRank graph-based measures of sentence centrality discussed above can also be viewed as a coherence metric (Erkan and Radev, 2004).

Figure 23.17 Some features commonly used in supervised classifiers for determining whether a document sentence should be extracted into a summary;

be computing the probability that a particular sentence s is extractworthy given a set of features $f_1 \ldots f_n$; then we can just extract any sentences for which this probability is greater than 0.5:

$$P(\text{extractworthy}(s)|f_1, f_2, f_3, \ldots, f_n) \tag{23.30}$$

There is one problem with the algorithm as we've described it: it requires that we have a training summary for each document which consists solely of extracted sentences. If we could weaken this restriction, we could apply the algorithm to a much wider variety of summary-document pairs, such as conference papers or journal articles and their abstracts. Luckily it turns out that when humans write summaries, even with the goal of writing abstractive summaries, they very often use phrases and sentences from the document to compose the summary. But they don't use *only* extracted sentences; they often combine two sentences into one, or change some of the words in the sentences, or write completely new abstractive sentences. Here is an example of an extracted sentence from a human summary that, although modified in the final human summary, was clearly a document sentence that should be labeled as extractworthy:

(23.31) **Human summary**: This paper identifies the desirable features of an ideal multisensor gas monitor and lists the different models currently available.

(23.32) **Original document sentence**: The present part lists the desirable features and the different models of portable, multisensor gas monitors currently available.

ALIGNMENT

Thus an important preliminary stage is to *align* each training document with its summary, with the goal of finding which sentences in the document were (completely or mostly) included in the summary. A simple algorithm for **alignment** is to find the source document and abstract sentences with the longest common subsequences of non-stopwords; alternatively minimum edit distance can be computed, or more sophisticated knowledge sources can be used, such as WordNet. Recent work has focused on more complex alignment algorithms such as the use of HMMs (Jing, 2002; Daumé III and Marcu, 2005, inter alia).

Given such alignment algorithms, supervised methods for content selection can make use of parallel corpora of documents and human abstractive summaries, such as academic papers with their abstracts (Teufel and Moens, 2002).

Sentence Simplification

SENTENCE COMPRESSION
SENTENCE SIMPLIFICATION

Once a set of sentences has been extracted and ordered, the final step in single-document summarization is **sentence realization**. One component of sentence realization is **sentence compression** or **sentence simplification**. The following examples, taken by Jing (2000) from a human summary, show that the human summarizer chose to eliminate some of the adjective modifiers and subordinate clauses when expressing the extracted sentence in the summary:

(23.33) **Original sentence:** ~~When it arrives sometime new year in new TV sets,~~ the V-chip will give parents a ~~new and potentially revolutionary~~ device to block out programs they don't want their children to see.

(23.34) **Simplified sentence by humans:** The V-chip will give parents a device to block out programs they don't want their children to see.

The simplest algorithms for sentence simplification use rules to select parts of the sentence to prune or keep, often by running a parser or partial parser over the sentences. Some representative rules from Zajic et al. (2007), Conroy et al. (2006), and Vanderwende et al. (2007a) remove the following:

appositives	Rajam, ~~28, an artist who was living at the time in Philadelphia~~, found the inspiration in the back of city magazines.
attribution clauses	Rebels agreed to talks with government officials, ~~international observers said Tuesday.~~
PPs without named entities	The commercial fishing restrictions in Washington will not be lifted [SBAR unless the salmon population 329 increases [PP ~~to a sustainable number~~]]
initial adverbials	"For example", "On the other hand", "As a matter of fact", "At this point"

More sophisticated models of sentence compression are based on supervised machine learning, in which a parallel corpus of documents together with their human summaries is used to compute the probability that particular words or parse nodes will be pruned. See the end of the chapter for pointers to this extensive recent literature.

23.4 Multi-Document Summarization

MULTI-DOCUMENT SUMMARIZATION

When we apply summarization techniques to groups of documents rather than a single document we call the goal **multi-document summarization**. Multi-document summarization is particularly appropriate for web-based applications, for example for building summaries of a particular event in the news by combining information from different news stories, or finding answers to complex questions by including components from extracted from multiple documents.

While multi-document summarization is far from a solved problem, even the current technology can be useful for information-finding tasks. McKeown et al. (2005), for example, gave human experimental participants documents together with a human summary, an automatically generated summary, or no summary, and had the participants perform time-restricted fact-gathering tasks. The participants had to answer three related questions about an event in the news; subjects who read the automatic summaries gave higher-quality answers to the questions.

Multi-document summarization algorithms are based on the same three steps we've seen before. In many cases we assume that we start with a cluster of documents that we'd like to summarize, and we must then perform **content selection**, **information ordering**, and **sentence realization**, as described in the next three sections and sketched in Fig. 23.18

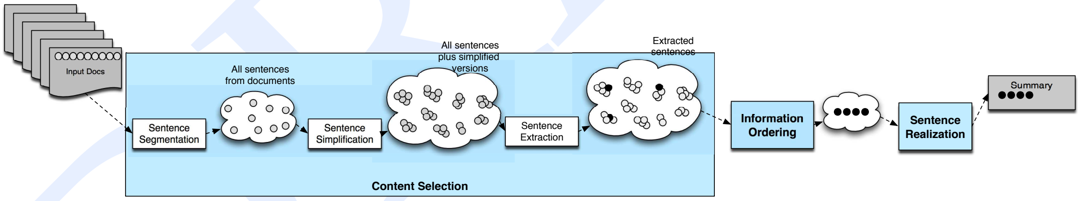


Figure 23.18 The basic architecture of a multi-document summarizer.

23.4.1 Content Selection in Multi-Document Summarization

In single document summarization we used both supervised and unsupervised methods for content selection. For multiple document summarization supervised training sets are less available, and we focus more on unsupervised methods.

The major difference between the tasks of single document and multiple document summarization is the greater amount of **redundancy** when we start with multiple documents. A group of documents can have significant overlap in words, phrases, and concepts, in addition to information that might be unique to each article. While we want each sentence in the summary to be about the topic, we don't want the summary to consist of a set of identical sentences.

For this reason, algorithms for multi-document summarization focus on ways to avoid redundancy when selected sentences for the summary. When adding a new sentence to a list of extracted sentences we need some way to make sure the sentence doesn't overlap too much with the already-extracted sentences.

MMR

MAXIMAL MARGINAL RELEVANCE

A simple method of avoiding redundancy is to explicitly include a redundancy factor in the scoring for choosing a sentence to extract. The redundancy factor is based on the similarity between a candidate sentence and the sentences that have already been extracted into the summary; a sentence is penalized if it is too similar to the summary. For example the **MMR** or **Maximal Marginal Relevance** scoring system Carbonell and Goldstein (1998), Goldstein et al. (2000) includes the following penalization term for representing the similarity between a sentence s and the set of sentences already extracted for the summary *Summary*, where λ is a weight that can be tuned and Sim is some similarity function:

$$\text{MMR penalization factor}(s) = \lambda max_{s_i \in Summary} \text{Sim}(s, s_i) \tag{23.35}$$

An alternative to MMR-based method is to instead apply a clustering algorithm to all the sentences in the documents to be summarized to produce a number of clusters of related sentences and then to select a single (centroid) sentence from each cluster into the summary.

By adding MMR or clustering methods for avoiding redundancy, we can also do sentence simplification or compression at the content selection stage rather than at the sentence realization stage. A common way to fit simplification into the architecture is to run various sentence simplification rules (Sec. 23.3.1) on each sentence in the input corpus. The result will be multiple versions of the input sentence, each version with different amounts of simplification. For example, the following sentence:

> Former Democratic National Committee finance director Richard Sullivan faced more pointed questioning from Republicans during his second day on the witness stand in the Senate's fund-raising investigation.

might produce different shortened versions:

- Richard Sullivan faced pointed questioning.
- Richard Sullivan faced pointed questioning from Republicans
- Richard Sullivan faced pointed questioning from Republicans during day on stand in Senate fundraising investigation
- Richard Sullivan faced pointed questioning from Republicans in Senate fundraising investigation

This expanded corpus is now used as the input to content extraction. Redundancy methods such as clustering or MMR will choose only the (optimally long) single version of each original sentence.

23.4.2 Information Ordering in Multi-Document Summarization

The second stage of an extractive summarizer is the ordering or structuring of information, where we must decide how to concatenate the extracted sentences into a coherent order. Recall that in single document summarization, we can just use the original article ordering for these sentences. This isn't appropriate for most multiple document applications, although we can certainly apply it if many or all of the extracted sentences happen to come from a single article.

CHRONOLOGICAL ORDERING

For sentences extracted from news stories, one technique is to use the dates associated with the story, a strategy known as **chronological ordering**. It turns out that pure chronological ordering can produce summaries which lack cohesion; this problem can be addressed by ordering slightly larger chunks of sentences rather than single sentences; see Barzilay et al. (2002).

Perhaps the most important factor for information ordering, however, is **coherence**. Recall from Ch. 21 the various devices that contribute to the coherence of a discourse. One is having sensible coherence relations between the sentences; thus we could prefer orderings in summaries that resulting in sensible coherence relations between the sentences. Another aspect of coherence has to do with cohesion and lexical chains; we could for example prefer orderings which have more local cohesion. A final aspect of coherence is coreference; a coherence discourse is one in which entities are mentioned in coherent patterns. We could prefer orderings with coherent entity mention patterns.

All of these kinds of coherence have been used for information ordering. For example we can use *lexical cohesion* as an ordering heuristic by ordering each sentence next to sentences containing similar words. This can done by defining the standard tf-idf cosine distance between each pair of sentences and choosing the overall ordering that minimizes the average distance between neighboring sentences Conroy et al. (2006), or by building models of predictable word sequences across sentences (Soricut and Marcu, 2006).

Coreference-based coherence algorithms have also made use of the intuitions of **Centering**. Recall that the Centering algorithm was based on the idea that each discourse segment has a salient entity, the *focus*. Centering theory proposed that certain syntactic realizations of the focus (i.e. as subject or object) and certain transitions between these realizations (e.g., if the same entity is the subject of adjacent sentences) created a more coherent discourse. Thus we can prefer orderings in which the transition between entity mentions is a preferred one.

ENTITY GRID

For example in the entity-based information approach of Barzilay and Lapata (2005, 2007), a training set of summaries is parsed and labeled for coreference. The resulting sequence of entity realizations can be automatically extracted and represented into an **entity grid**. Fig. 23.19 shows a simplified version of a parsed summary and the extracted grid. A probabilistic model of particular entity transitions (i.e. $\{S,O,X,-\}$ can then be trained from the entity grid. For example the transitions $\{X,O,S,S\}$ for the head word *Microsoft* exemplify the fact that new entities in a discourse are often introduced first in oblique or object position and then only later appear in subject position. See Barzilay and Lapata (2007) for details.

A general way to view all of these methods is as assigning a coherence score to a sequence of sentences via a local coherence score between pairs or sequences of sentences; a single general transition score between sentences could then combine lexical coherence and entity-based coherence. Once we have such a scoring function, choosing an ordering which optimizes all these local pairwise distances is known to be quite difficult. The task of finding the optimal ordering of a set of sentences given a set of pairwise distances between the sentences is equivalent to very hard problems like Cyclic Ordering and the Traveling Salesman Problem.[2] Sentence ordering is thus equivalent to the difficult class of problems known as **NP-complete**. While difficult to solve exactly, there are a number of good approximation methods for solving NP-complete problems that have been applied to the information ordering task. See Althaus et al. (2004), Knight (1999), Cohen et al. (1999), Brew (1992) for the relevant proofs and approximation techniques.

1 [The Justice Department]$_S$ is conducting an [anti-trust trial]$_O$ against [Microsoft Corp.]$_X$

2 [Microsoft]$_O$ is accused of trying to forcefully buy into [markets]$_X$ where [its own products]$_S$ are not competitive enough to unseat [established brands]$_O$

3 [The case]$_S$ resolves around [evidence]$_O$ of [Microsoft]$_S$ aggressively pressuring [Netscape]$_O$ into merging [browser software]$_O$

4 [Microsoft]$_S$ claims [its tactics]$_S$ are commonplace and good economically.

	Department	Trial	Microsoft	Markets	Products	Brands	Case	Netscape	Software	Tactics
1	S	O	X	-	-	-	-	-	-	-
2	-	-	O	X	S	O	-	-	-	-
3	-	-	S	O	-	-	S	O	O	-
4	-	-	S	-	-	-	-	-	-	O

Figure 23.19 A summary (showing entities in subject (S), object (O) or oblique (X) position), and the entity grid that is extracted from it. Adapted from Barzilay and Lapata (2005).

In the models described above, the information ordering task is completely separate from content extraction. An alternative approach is to learn the two tasks jointly, resulting in a model that both selects sentences and orders them. For example in the HMM model of Barzilay and Lee (2004), the hidden states correspond to document content topics and the observations to sentences. For example for newspaper articles on earthquakes, the hidden states (topics) might be *strength of earthquake*, *location*, *rescue efforts*, and *casualties*. They apply clustering and HMM induction to induce these hidden states and the transitions between them. For example, here are three sentences from the *location* cluster they induce:

(23.36) The Athens seismological institute said the temblor's epicenter was located 380 kilometers (238 miles) south of the capital.

(23.37) Seismologists in Pakistan's Northwest Frontier Province said the temblor's epicenter was about 250 kilometers (155 miles) north of the provincial capital Peshawar.

(23.38) The temblor was centered 60 kilometers (35 miles) northwest of the provincial capital of Kunming, about 2,200 kilometers (1,300 miles) southwest of Beijing, a bureau seismologist said.

The learned structure of the HMM then implicitly represent information ordering facts like *mention 'casualties' prior to 'rescue efforts'* via the HMM transition probabilities.

In summary, we've seen information ordering based on **chronological order**, based on **coherence**, and an ordering that is learned automatically from the data. In the next section on query-focused summarization we'll introduce a final method in which information ordering can be specified according to an ordering template which is predefined advance for different query types.

[2] The Traveling Salesman Problem: given a set of cities and the pairwise distances between them, find the shortest path that visits each city exactly once.

Sentence Realization

While discourse coherence can be factored in during sentence ordering, the resulting sentences may still have coherence problems. For example, as we saw in Ch. 21, when a referent appears multiple times in a coreference chain in a discourse, the longer or more descriptive noun phrases occur before shorter, reduced, or pronominal forms. But the ordering we choose for the extracted sentences may not respect this coherence preference.

For example the boldfaced names in the original summary in Fig. 23.20 appear in an incoherent order; the full name **U.S. President George W. Bush** occurs only after the shortened form **Bush** has been introduced.

One possible way to address this problem in the sentence realization stage is to apply a coreference resolution algorithm to the output, extracting names and applying some simple cleanup rewrite rules like the following:

(23.39) Use the **full name** at the first mention, and just the **last name** at subsequent mentions.

(23.40) Use a **modified** form for the first mention, but remove appositives or premodifiers from any subsequent mentions.

The rewritten summary in Fig. 23.20 shows how such rules would apply; in general such methods would depend on high-accuracy coreference resolution.

Original summary:

Presidential advisers do not blame **O'Neill**, but they've long recognized that a shakeup of the economic team would help indicate **Bush** was doing everything he could to improve matters. **U.S. President George W. Bush** pushed out **Treasury Secretary Paul O'Neill** and top economic adviser Lawrence Lindsey on Friday, launching the first shake - up of his administration to tackle the ailing economy before the 2004 election campaign.

Rewritten summary:

Presidential advisers do not blame **Treasury Secretary Paul O'Neill**, but they've long recognized that a shakeup of the economic team would help indicate **U.S. President George W. Bush** was doing everything he could to improve matters. **Bush** pushed out **O'Neill** and White House economic adviser Lawrence Lindsey on Friday, launching the first shake-up of his administration to tackle the ailing economy before the 2004 election campaign.

Figure 23.20 Rewriting references, from Nenkova and McKeown (2003)

SENTENCE FUSION

Recent research has also focused on a finer granularity for realization than the extracted sentence, by using **sentence fusion** algorithms to combine phrases or clauses from different sentences into one new sentence. The sentence fusion algorithm of Barzilay and McKeown (2005) parses each sentence, uses multiple-sequence alignment of the parses to find areas of common information, builds a fusion lattice with overlapping information, and creates a fused sentence by linearizing a string of words from the lattice.

23.5 Between Question Answering and Summarization: Query-Focused Summarization

As noted in at the beginning of this chapter, most interesting questions are not factoid questions. User needs require longer, more informative answers than a single phrase can provide. For example, while a DEFINITION question might be answered by a short phrase like "*Autism is* **a developmental disorder**" or "*A caldera is* **a volcanic crater**", a user might want more information, as in the following definition of *water spinach*:

> *Water spinach* (ipomoea aquatica) is a semi-aquatic leafy green plant characterized by long hollow stems and spear-shaped or heart-shaped leaves which is widely grown throughout Asia as a leaf vegetable. The leaves and stems are often eaten stir-fried as greens with salt or salty sauces, or in soups. Other common names include *morning glory vegetable*, *kangkong* (Malay), *rau muong* (Vietnamese), *ong choi* (Cantonese), and *kong xin cai* (Mandarin). It is not related to spinach, but is closely related to sweet potato and convolvulus.

Complex questions can also be asked in domains like medicine, such as this question about a particular drug intervention:

(23.41) In children with an acute febrile illness, what is the efficacy of single-medication therapy with acetaminophen or ibuprofen in reducing fever?

For this medical question, we'd like to be able to extract an answer of the following type, perhaps giving the document id(s) that the extract came from, and some estimate of our confidence in the result:

> Ibuprofen provided greater temperature decrement and longer duration of antipyresis than acetaminophen when the two drugs were administered in approximately equal doses. (PubMedID: 1621668, Evidence Strength: A)

Questions can be even more complex, such as this one from the Document Understanding Conference annual summarization competition:

(23.42) Where have poachers endangered wildlife, what wildlife has been endangered and what steps have been taken to prevent poaching?

Where a factoid answer might be found in a single phrase in a single document or web page, these kinds of complex questions are likely to require much longer answers which are synthesized from many documents or pages.

For this reason, summarization techniques are often used to build answers to these kinds of complex questions. But unlike the summarization algorithms introduced above, the summaries produced for complex question answering must be relevant to some user question. When a document is summarized for the purpose of answering some user query or information need, we call the goal **query-focused summarization** or sometimes just **focused summarization**. (The terms **topic-based summarization** and **user-focused summarization** are also used.) A query-focused summary is thus really a kind of longer, non-factoid answer to a user question or information need.

QUERY-FOCUSED SUMMARIZATION
FOCUSED SUMMARIZATION

SNIPPET

One kind of query-focused summary is a **snippet**, the kind that web search engines like Google return to the user to describe each retrieved document. Snippets are query-focused summaries of a single document. But since for complex queries we will want to aggregate information from multiple documents, we'll need to summarize multiple documents.

Indeed, the simplest way to do query-focused summarization is to slightly modify the algorithms for multiple document summarization that we introduced in the previous section to make use of the query. For example, when ranking sentences from all the returned documents in the content selection phase, we can require that any extracted sentence must contain at least one word overlapping with the query. Or we can just add the cosine distance from the query as one of the relevance features in sentence extraction. We can characterize such a method of query-focused summarization as a bottom-up, domain-independent method.

An alternative way to do query-focused summarization is to make additional use of top-down or information-extraction techniques, building specific content selection algorithms for different types of complex questions. Thus we could specifically build a query-focused summarizer for the kinds of advanced questions introduced above, like definition questions, biography questions, certain medical questions. In each case, we use our top-down expectations for what makes a good definition, biography, or medical answer to guide what kinds of sentences we extract.

GENUS

SPECIES

For example, a **definition** of a term often includes information about the term's **genus** and **species**. The genus is the hypernym or superordinate of the word; thus a sentence like *The Hajj is a type of ritual* is a genus sentence. The species gives important additional properties of the term that differentiate the term from other hyponyms of the genus; an example is *"The annual hajj begins in the twelfth month of the Islamic year"*. Other kinds of information that can occur in a definition include **synonyms**, **etymology**, **subtypes**, and so on.

In order to build extractive answers for definition questions, we'll need to make sure we extract sentences with the genus information, the species information, and other generally informative sentences. Similarly, a good **biography** of a person contains information such as the person's **birth/death**, **fame factor**, **education**, **nationality** and so on; we'll need to extract sentences with each of these kinds of information. A medical answer that summarizes the results of a study on applying a drug to a medical problem would need to contain information like the **problem** (the medical condition), the **intervention** (the drug or procedure), and the **outcome** (the result of the study).

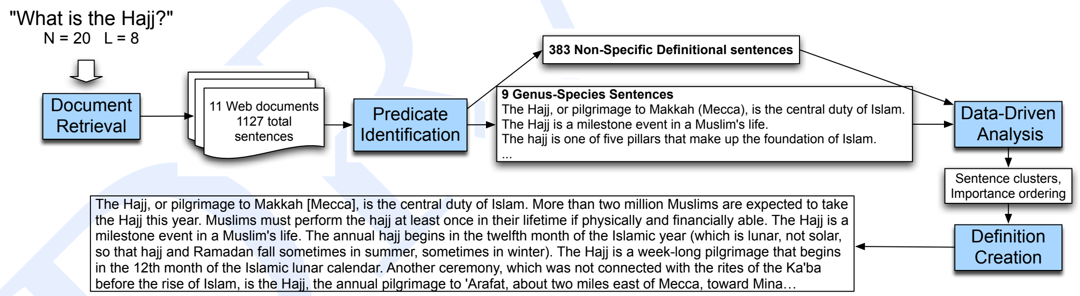
Fig. 23.21 shows some example predicates for definition, biography, and medical intervention questions.

In each case we we use the **information extraction** methods of Ch. 22 to find specific sentences for genus and species (for definitions), or dates, nationality, and education (for biographies), or problems, interventions and outcomes (for medical questions). We can then use standard domain-independent content selection algorithms to find other good sentences to add on to these.

A typical architecture consists of the four steps shown in Fig. 23.22 from the definition extraction system of Blair-Goldensohn et al. (2004). The input is a definition question T, the number N of documents to retrieve, and the length L of the answer (in sentences).

Definition	
genus	`The Hajj is a type of ritual`
species	`the annual hajj begins in the twelfth month of the Islamic year`
synonym	`The Hajj, or Pilgrimage to Mecca, is the central duty of Islam`
subtype	`Qiran, Tamattu’, and Ifrad are three different types of Hajj`
Biography	
dates	`was assassinated on April 4, 1968`
nationality	`was born in Atlanta, Georgia`
education	`entered Boston University as a doctoral student`
Drug efficacy	
population	`37 otherwise healthy children aged 2 to 12 years`
problem	`acute, intercurrent, febrile illness`
intervention	`acetaminophen (10 mg/kg)`
outcome	`ibuprofen provided greater temperature decrement and longer duration of antipyresis than acetaminophen when the two drugs were administered in approximately equal doses`

Figure 23.21 Examples of some different types of information that must be extracted in order to produce answer to certain kinds of complex questions.


"What is the Hajj?"
N = 20 L = 8
Document Retrieval
11 Web documents 1127 total sentences
Predicate Identification
383 Non-Specific Definitional sentences
9 Genus-Species Sentences
Data-Driven Analysis
Sentence clusters, Importance ordering
Definition Creation

The Hajj, or pilgrimage to Makkah (Mecca), is the central duty of Islam.
The Hajj is a milestone event in a Muslim's life.
The hajj is one of five pillars that make up the foundation of Islam.
...

The Hajj, or pilgrimage to Makkah [Mecca], is the central duty of Islam. More than two million Muslims are expected to take the Hajj this year. Muslims must perform the hajj at least once in their lifetime if physically and financially able. The Hajj is a milestone event in a Muslim's life. The annual hajj begins in the twelfth month of the Islamic year (which is lunar, not solar, so that hajj and Ramadan fall sometimes in summer, sometimes in winter). The Hajj is a week-long pilgrimage that begins in the 12th month of the Islamic lunar calendar. Another ceremony, which was not connected with the rites of the Ka'ba before the rise of Islam, is the Hajj, the annual pilgrimage to 'Arafat, about two miles east of Mecca, toward Mina…

Figure 23.22 Architecture of a query-focused summarizer for definition questions (Blair-Goldensohn et al., 2004).

The first step in any IE-based complex question answering system is information retrieval. In this case a handwritten set of patterns is used to extract the term to be defined from the query T (*Hajj*) and generate a series of queries that are sent to an IR engine. Similarly, in a biography system it would be the name that would be extracted and passed to the IR engine. The returned documents are broken up into sentences.

In the second stage, we apply classifiers to label each sentence with an appropriate set of classes for the domain. For definition questions, Blair-Goldensohn et al. (2004) used of four classes: **genus**, **species**, **other definitional**, or **other**. The third class,

other definitional, is used to select other sentences that might be added into the summary. These classifiers can be based on any of the information extraction techniques introduced in Ch. 22, including hand-written rules, or supervised machine learning techniques.

In the third stage, we can use the methods described in the section on generic (non-query-focused) multiple domain summarization content selection to add additional sentences to our answer that might not fall into a specific information extraction type. For example for definition questions, all the sentences that are classified as *other definitional* are examined, and a set of relevant sentences is selected from them. This selection can be done by the centroid method, in which we form a TF-IDF vector for each sentence, find the centroid of all the vectors, and then choose the K sentences closest to the centroid. Alternatively we can use a method for avoiding redundancy, like clustering the vectors and choosing the best sentence from each cluster.

Because query-focused summarizers of this type or domain-specific, we can use domain-specific methods for information ordering as well, such as using a fixed hand-built template. For biography questions we might use a template like the following:

(23.43) <NAME> is <WHY FAMOUS>. She was born on <BIRTHDATE> in
<BIRTHLOCATION>. She <EDUCATION>. <DESCRIPTIVE SENTENCE>.
<DESCRIPTIVE SENTENCE>.

The various sentences or phrases selected in the content selection phase can then be fit into this template. These templates can also be somewhat more abstract. For example, for definitions, we could place a genus-species sentence first, followed by remaining sentences ordered by their saliency scores.

23.6 Summarization Evaluation

As is true for other speech and language processing areas like machine translation, there are a wide variety of evaluation metrics for summarization, metrics requiring human annotation, as well as completely automatic metrics.[3]

As we have seen for other tasks, we can evaluate a system via **extrinsic** (task-based) or **intrinsic** (task-independent) methods. We described a kind of extrinsic evaluation of multi-document summarization in Sec. 23.4, in which subjects were asked to perform time-restricted fact-gathering tasks, and were given full documents together with either no summaries, human summaries, or automatically generated summaries to read. The subjects had to answer three related questions about an event in the news. For query-focused single-document summarization (like the task of generating web **snippets**), we can measure how different summarization algorithms affect human performance at the task of deciding if a document is relevant/not-relevant to a query by looking solely at the summary.

ROUGE

The most common intrinsic summarization evaluation metric is an automatic method called **ROUGE**, **Recall-Oriented Understudy for Gisting Evaluation** (Lin and Hovy,

[3] We focus here on evaluation of entire summarization algorithms and ignore evaluation of subcomponents such as information ordering, although see for example (Lapata, 2006) on the use of Kendall's τ, a metric of

2003; Lin, 2004). ROUGE is inspired by the BLEU metric used for evaluating machine translation output, and like BLEU, automatically scores a machine-generated candidate summary by measuring the amount of N-gram overlap between the candidate and human-generated summaries (the references).

ROUGE-1
ROUGE-2

Recall that BLEU is computed by averaging the number of overlapping N-grams of different length between the hypothesis and reference translations. In ROUGE, by contrast, the length of the N-gram is fixed; **ROUGE-1** uses unigram overlap, while **ROUGE-2** uses bigram overlap. We'll choose to define ROUGE-2; the definitions of all the other ROUGE-N metrics follows. ROUGE-2 is a measure of the bigram recall between the candidate summary and the set of human reference summaries:

$$ROUGE2 = \frac{\sum\limits_{S\in\{ReferenceSummaries\}} \sum\limits_{bigram\in S} \text{Count}_{\text{match}}(bigram)}{\sum\limits_{S\in\{ReferenceSummaries\}} \sum\limits_{bigram\in S} \text{Count}(bigram)} \tag{23.44}$$

The function $\text{Count}_{\text{match}}(bigram)$ returns the maximum number of bigrams that co-occur in the candidate summary and the set of reference summaries. ROUGE-1 is the same but counting unigrams instead of bigrams.

Note that ROUGE is a recall-oriented measure, where BLEU is a precision-oriented measure. This is because the denominator of (23.44) is the total sum of the number of bigrams in the reference summaries. By contrast, in BLEU the denominator is the total sum of the number of N-grams in the candidates. Thus ROUGE is measuring something like how many of the human reference summary bigrams are covered by the candidate summary, where BLEU is measuring something like how many of the candidate translation bigrams occurred in the human reference translations.

ROUGE-L
ROUGE-S
ROUGE-SU
SKIP BIGRAMS

Variants of ROUGE include **ROUGE-L**, which measure the **longest common subsequence** between the reference and candidate summaries, and **ROUGE-S** and **ROUGE-SU** which measure the number of **skip bigrams** between the reference and candidate summaries. A skip bigram is a pair of words in their sentence order, but allowing for any number of other words to appear between the pair.

While ROUGE is the most commonly applied automatic baseline, it is not as applicable to summarization as similar metrics like BLEU are to machine translation. This is because human summarizers seem to disagree strongly about which sentences to include in a summary, making even the overlap of humans with each other very low.

PYRAMID METHOD
SUMMARY CONTENT UNITS

This difference in which sentences humans choose to extract has motivated human evaluation methods which attempt to focus more on meaning. One metric, the **Pyramid Method**, is a way of measuring how many units of meaning are shared between the candidate and reference summaries, and also weights the units of meaning by importance; units of meaning which occur in more of the human summaries are weighted more highly. The units of meaning are called **Summary Content Units** (SCU), which are sub-sentential semantic units which roughly correspond to propositions or coherent pieces of propositions.

In the Pyramid Method, humans label the Summary Content Units in each reference and candidate summary, and then an overlap measure is computed.

rank correlation, for information ordering.

Let's see an example from Nenkova et al. (2007) of how two SCUs are labeled in sentences from six human abstracts. We'll first show sentences from the human summaries indexed by a letter (corresponding to one of the 6 human summaries) and a number (the position of the sentence in the human summary):

A1. The industrial espionage case involving GM and VW began with the hiring of Jose Ignacio Lopez, an employee of GM subsidiary Adam Opel, by VW as a production director.

B3. However, he left GM for VW under circumstances, which along with ensuing events, were described by a German judge as "potentially the biggest-ever case of industrial espionage".

C6. He left GM for VW *in March 1993*.

D6. The issue stems from the alleged recruitment of GM's eccentric and visionary Basque-born procurement chief Jose Ignacio Lopez de Arriortura and seven of Lopez's business colleagues.

E1. *On March 16, 1993*, with Japanese car import quotas to Europe expiring in two years, renowned cost-cutter, Agnacio Lopez De Arriortura, left his job as head of purchasing at General Motor's Opel, Germany, to become Volkswagen's Purchasing and Production director.

F3. *In March 1993*, Lopez and seven other GM executives moved to VW overnight.

The annotators first identify similar sentences, like those above, and then label SCUs. The underlined and italicized spans of words in the above sentences result in the following two SCUs, each one with a weight corresponding to the number of summaries it appears in (6 for the first SCU, and 3 for the second):

SCU1 (w=6): *Lopez left GM for VW*
A1. the hiring of Jose Ignacio Lopez, an employee of GM . . . by VW
B3. he left GM for VW
C6. He left GM for VW
D6. recruitment of GMs . . . Jose Ignacio Lopez
E1. Agnacio Lopez De Arriortura, left his job . . . at General Motors Opel
. . . to become Volkswagens . . . director
F3. Lopez . . . GM . . . moved to VW

SCU2 (w=3) *Lopez changes employers in March 1993*
C6. in March, 1993
E1. On March 16, 1993
F3. In March 1993

Once the annotation is done, the informativeness of a given summary can be measured as the ratio of the sum of the weights of its SCUs to the weight of an optimal summary with the same number of SCUs. See the end of the chapter for more details and pointers to the literature.

RANDOM SENTENCES
LEADING SENTENCES

The standard baselines for evaluating summaries are the **random sentences** baseline and the **leading sentences** baseline. Assuming we are evaluating summaries of length N sentences, the random baseline just chooses N random sentences, while the leading baseline chooses the first N sentences. The leading sentences method, in particular, is quite a strong baseline and many proposed summarization algorithms fail to beat it.

23.7 Summary

- The dominant models of information retrieval represent the meanings of documents and queries as bags of words.
- The **vector space model** views documents and queries as vectors in a large multi-dimensional space. In this model, the similarity between documents and queries, or other documents, can be measured by the cosine of the angle between the vectors.
- The main components of a factoid question answering system are the **question classification** module to determine the named-entity type of the answer, a **passage retrieval** module to identify relevant passages, and an answer processing module to extract and format the final answer.
- Factoid question answers can be evaluated via **mean reciprocal rank** (**MRR**).
- Summarization can be **abstractive** or **extractive**; most current algorithms are extractive.
- Three components of **summarization algorithms** include **content selection**, **information ordering**, and **sentence realization**.
- Current single document summarization algorithms focus mainly on **sentence extraction**, relying on features like **position** in the discourse, **word informativeness**, **cue phrases**, and **sentence length**.
- Multiple document summarization algorithms often perform **sentence simplification** on document sentences.
- **Redundancy avoidance** is important in multiple document summarization; it is often implemented by adding a redundancy penalization term like **MMR** into sentence extraction.
- **Information ordering** algorithms in multi-document summarization are often based on maintaining **coherence**.
- **Query-focused summarization** can be done using slight modifications to **generic summarization** algorithms, or by using information-extraction methods.

Bibliographical and Historical Notes

Luhn (1957) is generally credited with first advancing the notion of fully automatic indexing of documents based on their contents. Over the years Salton's SMART project (Salton, 1971) at Cornell developed or evaluated many of the most important notions in information retrieval including the vector model, term weighting schemes, relevance feedback, and the use of cosine as a similarity metric. The notion of using inverse document frequency in term weighting is due to Sparck Jones (1972). The original notion of relevance feedback is due to Rocchio (1971).

PROBABILISTIC MODEL

An alternative to the vector model that we have not covered is the **probabilistic model** originally shown effective by Robinson and Sparck Jones (1976). See Crestani et al. (1998) and Chapter 11 of Manning et al. (2008) on probabilistic models in information retrieval.

Manning et al. (2008) is a comprehensive modern text on information retrieval. Good but slightly older texts include Baeza-Yates and Ribeiro-Neto (1999) and Frakes and Baeza-Yates (1992); older classic texts include Salton and McGill (1983) and van Rijsbergen (1975). Many of the classic papers in the field can be found in Sparck Jones and Willett (1997). Current work is published in the annual proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR). The US National Institute of Standards and Technology (NIST) has run an annual evaluation project for text information retrieval and extraction called the Text REtrieval Conference (TREC) since the early 1990s; the conference proceedings from TREC contain results from these standardized evaluations. The primary journals in the field are the *Journal of the American Society of Information Sciences*, *ACM Transactions on Information Systems*, *Information Processing and Management*, and *Information Retrieval*.

Question answering was one of the earliest tasks for NLP systems in the 1960's and 1970's (Green et al., 1961; Simmons, 1965; Woods et al., 1972; Lehnert, 1977), but the field lay dormant for a few decades until the need for querying the Web brought the task back into focus. The U.S. government-sponsored TREC (Text REtrieval Conference) QA track began in 1999 and a wide variety of factoid and non-factoid systems have been competing in annual evaluations since then. See the references in the chapter and Strzalkowski and Harabagiu (2006) for a collection of recent research papers.

Research on text summarization began with the work of Luhn (1958) on extractive methods for the automatic generation of abstracts, focusing on surface features like term frequency, and the later work of Edmunson (1969) incorporating positional features as well. Term-based features were also used in the early application of automatic summarization at Chemical Abstracts Service (Pollock and Zamora, 1975). The 1970s and 1980s saw a number of approaches grounded in AI methodology such as scripts DeJong (1982), semantic networks Reimer and Hahn (1988), or combinations of AI and statistical methods Rau et al. (1989).

The work of Kupiec et al. (1995) on training a sentence classifier with supervised machine learning led to many statistical methods for sentence extraction. Around the turn of the century, the growth of the Web led naturally to interest in multi-document summarization and query-focused summarization.

SUMBASIC

CENTROID

There have naturally been a wide variety of algorithms for the main components of summarizers. The simple unsupervised log-linear content selection algorithm we describe is simplified from the **SumBasic** algorithm of Nenkova and Vanderwende (2005), Vanderwende et al. (2007b) and the **centroid** algorithm of Radev et al. (2000) and Radev et al. (2001). A number of algorithms for information ordering have used entity coherence, including Kibble and Power (2000), Lapata (2003), Karamanis and Manurung (2002), Karamanis (2003), Barzilay and Lapata (2005, 2007). Algorithms for combining multiple cues for coherence and searching for the optimal ordering include Althaus et al. (2004), based on linear programming, the genetic algorithms of

Mellish et al. (1998) and Karamanis and Manurung (2002), and the Soricut and Marcu (2006) algorithm, which uses A* search based on IDL-expressions. Karamanis (2007) showed that adding coherence based on rhetorical relations to entity coherence didn't improve sentence ordering. See Lapata (2006, 2003), Karamanis et al. (2004), Karamanis (2006) on methods for evaluating information ordering.

Sentence compression is a very popular area of research. Early algorithms focused on the use of syntactic knowledge for eliminating less important words or phrases Grefenstette (1998), Mani et al. (1999), Jing (2000). Recent research has focused on using supervised machine learning, in which a parallel corpus of documents together with their human summaries is used to compute the probability that particular words or parse nodes will be pruned. Methods include the use of maximum entropy Riezler et al. (2003), the noisy channel model and synchronous context-free grammars (Galley and McKeown, 2007; Knight and Marcu, 2000; Turner and Charniak, 2005; Daumé III and Marcu, 2002), Integer Linear Programming Clarke and Lapata (2007), and large-margin learning McDonald (2006). These methods rely on various features, especially including syntactic or parse knowledge Jing (2000), Dorr et al. (2003), Siddharthan et al. (2004), Galley and McKeown (2007), Zajic et al. (2007), Conroy et al. (2006), Vanderwende et al. (2007a), but also including coherence information Clarke and Lapata (2007). Alternative recent methods are able to function without these kinds of parallel document/summary corpora (Hori and Furui, 2004; Turner and Charniak, 2005; Clarke and Lapata, 2006).

See Daumé III and Marcu (2006) for a recent Bayesian model of query-focused summarization.

For more information on summarization evaluation, see Nenkova et al. (2007), Passonneau et al. (2005), and Passonneau (2006) for details on the Pyramid method, van Halteren and Teufel (2003) and Teufel and van Halteren (2004) on related semantic-coverage evaluation methods, and Lin and Demner-Fushman (2005) on the link between evaluations for summarization and question answering. A NIST program starting in 2001, the Document Understanding Conference (DUC), has sponsored an annual evaluation of summarization algorithms. These have included single document, multiple document, and query-focused summarization; proceedings from the annual workshop are available online.

Mani and Maybury (1999) is the definitive collection of classic papers on summarization. Sparck Jones (2007) is a good recent survey, and Mani (2001) is the standard textbook.

PARAPHRASE DETECTION

The task of **paraphrase detection** is an important task related to improving recall in question answering and avoiding redundancy in summarization, and also very relevant for tasks like textual entailment. See Lin and Pantel (2001), Barzilay and Lee (2003), Pang et al. (2003), Dolan et al. (2004), Quirk et al. (2004) for representative papers on techniques for detecting paraphrases.

TEXT CATEGORIZATION

SPAM DETECTION

Another task related to information retrieval and summarization is the **text categorization** task, which is to assign a new document to one of a pre-existing set of document classes. The standard approach is to use supervised machine learning to train classifiers on a set of documents that have been labeled with the correct class. A very important application of text categorization is for **spam detection**.

EXERCISES

23.1 Do some error analysis on web-based question answering. Choose 10 questions and type them all into two different search engines. Analyze the errors (e.g., what kinds of questions could neither system answer; which kinds of questions did one work better on; was there a type of question that could be answered just from the snippets, etc).

23.2 Read Brill et al. (2002) and reimplement a simple version of the AskMSR system.

Agichtein, E. and Gravano, L. (2000). Snowball: Extracting relations from large plain-text collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*.

Althaus, E., Karamanis, N., and Koller, A. (2004). Computing locally coherent discourses. In *ACL-04*.

Attar, R. and Fraenkel, A. S. (1977). Local feedback in full-text retrieval systems. *Journal of the ACM*, *24*(3), 398–417.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. ACM Press, New York.

Barzilay, R. and Elhadad, M. (1997). Using lexical chains for text summarization. In *Proceedings of the ACL Workshop on Intelligent Scalable Text Summarization*, pp. 10–17.

Barzilay, R., Elhadad, N., and McKeown, K. R. (2002). Inferring strategies for sentence ordering in multidocument news summarization. *Journal of Artificial Intelligence Research*, *17*, 35–55.

Barzilay, R. and Lapata, M. (2005). Modeling local coherence: an entity-based approach. In *ACL-05*, pp. 141–148.

Barzilay, R. and Lapata, M. (2007). Modeling local coherence: an entity-based approach. *Computational Linguistics*. To appear.

Barzilay, R. and Lee, L. (2003). Learning to paraphrase: an unsupervised approach using multiple-sequence alignment. In *HLT-NAACL-03*, pp. 16–23.

Barzilay, R. and Lee, L. (2004). Catching the drift: Probabilistic content models, with applications to generation and summarization. In *HLT-NAACL-04*, pp. 113–120.

Barzilay, R. and McKeown, K. R. (2005). Sentence fusion for multidocument news summarization. *Computational Linguistics*, *31*(3), 297–328.

Blair-Goldensohn, S., McKeown, K. R., and Schlaikjer, A. H. (2004). Answering definitional questions: A hybrid approach. In Maybury, M. T. (Ed.), *New Directions in Question Answering*, pp. 47–58. AAAI Press.

Brew, C. (1992). Letting the cat out of the bag: generation for shake-and-bake mt. In *COLING-92*, pp. 610–616.

Brill, E., Dumais, S. T., and Banko, M. (2002). An analysis of the AskMSR question-answering system. In *EMNLP 2002*, pp. 257–264.

Brin, S. (1998). Extracting patterns and relations from the World Wide Web. In *Proceedings World Wide Web and Databases International Workshop, Number 1590 in LNCS*, pp. 172–183. Springer.

Carbonell, J. and Goldstein, J. (1998). The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR 1998*, pp. 335–336.

Clarke, J. and Lapata, M. (2006). Models for sentence compression: A comparison across domains, training requirements and evaluation measures. In *COLING/ACL 2006*, pp. 377–384.

Clarke, J. and Lapata, M. (2007). Modelling compression with discourse constraints. In *EMNLP/CoNLL 2007*, Prague, pp. 667–677.

Cohen, W. W., Schapire, R. E., and Singer, Y. (1999). Learning to order things. *Journal of Artificial Intelligence Research*, *10*, 243–270.

Conroy, J. M., Schlesinger, J. D., and Goldstein, J. (2006). Classy tasked based summarization: Back to basics. In *DUC-06*.

Crestani, F., Lemas, M., Rijsbergen, C. J. V., and Campbell, I. (1998). “Is This Document Relevant? . . . Probably”: a survey of probabilistic models in information retrieval. *ACM Computing Surveys*, *30*(4), 528–552.

Crouch, C. J. and Yang, B. (1992). Experiments in automatic statistical thesaurus construction. In *SIGIR-92*, Copenhagen, Denmark, pp. 77–88. ACM.

Daumé III, H. and Marcu, D. (2002). A noisy-channel model for document compression. In *ACL-02*.

Daumé III, H. and Marcu, D. (2005). Induction of word and phrase alignments for automatic document summarization. *Computational Linguistics*, *31*(4), 505–530.

Daumé III, H. and Marcu, D. (2006). Bayesian query-focused summarization. In *COLING/ACL 2006*, Sydney, Australia.

DeJong, G. F. (1982). An overview of the FRUMP system. In Lehnert, W. G. and Ringle, M. H. (Eds.), *Strategies for Natural Language Processing*, pp. 149–176. Lawrence Erlbaum, New Jersey.

Dolan, W. B., Quirk, C., and Brockett, C. (2004). Unsupervised construction of large paraphrase corpora: exploiting massively parallel news sources. In *COLING-04*.

Dorr, B., Zajic, D., and Schwartz, R. (2003). Hedge trimmer: a parse-and-trim approach to headline generation. In *HLT-NAACL Workshop on Text Summarization*, pp. 1–8.

Dunning, T. (1993). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, *19*(1), 61–74.

Echihabi, A., Hermjakob, U., Hovy, E. H., Marcu, D., Melz, E., and Ravichandran, D. (2005). How to select an answer string?. In Strzalkowski, T. and Harabagiu, S. (Eds.), *Advances in Textual Question Answering*. Kluwer.

Edmunson, H. (1969). New methods in automatic extracting. *Journal of the ACM*, *16*(2), 264–285.

Erkan, G. and Radev, D. R. (2004). Lexrank: Graph-based centrality as salience in text summarization. *Journal of Artificial Intelligence Research (JAIR)*, *22*, 457–479.

Frakes, W. B. and Baeza-Yates, R. (1992). *Information Retrieval: Data Structures and Algorithms*. Prentice Hall.

Galley, M. and McKeown, K. R. (2007). Lexicalized Markov grammars for sentence compression. In *NAACL-HLT 07*, Rochester, NY, pp. 180–187.

Goldstein, J., Mittal, V., Carbonell, J., and Kantrowitz, M. (2000). Multi-document summarization by sentence extraction. In *Proceedings of the ANLP/NAACL Workshop on Automatic Summarization*.

Green, B. F., Wolf, A. K., Chomsky, C., and Laughery, K. (1961). Baseball: An automatic question answerer. In *Proceedings of the Western Joint Computer Conference 19*, pp. 219–224. Reprinted in Grosz et al. (1986).

Grefenstette, G. (1998). Producing intelligent telegraphic text reduction to provide an audio scanning service for the blind. In *AAAI 1998 Spring Symposium on Intelligent Text Summarization*, pp. 102–108.

Hachey, B. and Grover, C. (2005). Sentence extraction for legal text summarization. In *IJCAI-05*, pp. 1686–1687.

Harabagiu, S., Pasca, M., and Maiorano, S. (2000). Experiments with open-domain textual question answering. In *COLING-00*, Saarbrücken, Germany.

Hori, C. and Furui, S. (2004). Speech summarization: an approach through word extraction and a method for evaluation. *IEICE Transactions on Information and Systems*, *87*, 15–25.

Hovy, E., Hermjakob, U., and Ravichandran, D. (2002). A question/answer typology with surface text patterns. In *HLT-01*.

Hovy, E. H. and Lin, C.-Y. (1999). Automated text summarization in SUMMARIST. In Mani, I. and Maybury, M. T. (Eds.), *Advances in Automatic Text Summarization*, pp. 81–94. MIT Press.

Jing, H. (2000). Sentence reduction for automatic text summarization. In *ANLP 2000*, Seattle, WA, pp. 310–315.

Jing, H. (2002). Using hidden Markov modeling to decompose human-written summaries. *Computational Linguistics*, *28*(4), 527–543.

Karamanis, N. (2003). *Entity Coherence for Descriptive Text Structuring*. Ph.D. thesis, University of Edinburgh.

Karamanis, N. (2006). Evaluating centering for sentence ordering in two new domains. In *HLT-NAACL-06*.

Karamanis, N. (2007). Supplementing entity coherence with local rhetorical relations for information ordering. *Journal of Logic, Language and Information*. To appear.

Karamanis, N. and Manurung, H. M. (2002). Stochastic text structuring using the principle of continuity. In *INLG 2002*, pp. 81–88.

Karamanis, N., Poesio, M., Mellish, C., and Oberlander, J. (2004). Evaluating centering-based metrics of coherence for text structuring using a reliably annotated corpus. In *ACL-04*.

Kibble, R. and Power, R. (2000). An integrated framework for text planning and pronominalisation. In *INLG 2000*, pp. 77–84.

Knight, K. (1999). Decoding complexity in word-replacement translation models. *Computational Linguistics*, *25*(4), 607–615.

Knight, K. and Marcu, D. (2000). Statistics-based summarization - step one: Sentence compression. In *AAAI-00*, pp. 703–710.

Krovetz, R. and Croft, W. B. (1992). Lexical ambiguity and information retrieval. *ACM Transactions on Information Systems*, *10*(2), 115–141.

Kupiec, J., Pedersen, J., and Chen, F. (1995). A trainable document summarizer. In *SIGIR 1995*, pp. 68–73.

Lapata, M. (2003). Probabilistic text structuring: experiments with sentence ordering. In *ACL-03*, Sapporo, Japan, pp. 545–552.

Lapata, M. (2006). Automatic evaluation of information ordering. *Computational Linguistics*, *32*(4), 471–484.

Lehnert, W. G. (1977). A conceptual theory of question answering. In *IJCAI-77*, pp. 158–164. Morgan Kaufmann.

Li, X. and Roth, D. (2002). Learning question classifiers. In *COLING-02*, pp. 556–562.

Li, X. and Roth, D. (2005). Learning question classifiers: The role of semantic information. *Journal of Natural Language Engineering*, *11*(4).

Lin, C.-Y. (2004). ROUGE: A package for automatic evaluation of summaries. In *ACL 2004 Workshop on Text Summarization Branches Out*.

Lin, C.-Y. and Hovy, E. (2000). The automated acquisition of topic signatures for text summarization. In *COLING-00*, pp. 495–501.

Lin, C.-Y. and Hovy, E. H. (2003). Automatic evaluation of summaries using N-gram co-occurrence statistics. In *HLT-NAACL-03*, Edmonton, Canada.

Lin, D. and Pantel, P. (2001). Discovery of inference rules for question-answering. *Natural Language Engineering*, *7*(4), 343–360.

Lin, J. and Demner-Fushman, D. (2005). Evaluating summaries and answers: Two sides of the same coin?. In *ACL 2005 Workshop on Measures for MT and Summarization*.

Lin, J. (2007). An exploration of the principles underlying redundancy-based factoid question answering. *ACM Transactions on Information Systems*.

Luhn, H. P. (1957). A statistical approach to the mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, *1*(4), 309–317.

Luhn, H. P. (1958). The automatic creation of literature abstracts. *IBM Journal of Research and Development*, *2*(2), 159–165.

Mani, I. (2001). *Automatic Summarization*. John Benjamins.

Mani, I. and Bloedorn, E. (1999). Summarizing similarities and differences among related documents. *Information Retrieval*, *1*(1-2), 35–67.

Mani, I., Gates, B., and Bloedorn, E. (1999). Improving summaries by revising them. In *ACL-99*, pp. 558–565.

Mani, I. and Maybury, M. T. (1999). *Advances in Automatic Text Summarization*. MIT Press.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Marcu, D. (1995). Discourse trees are good indicators of importance in text. In Mani, I. and Maybury, M. T. (Eds.), *Advances in Automatic Text Summarization*, Cambridge, MA, pp. 123–136. MIT Press.

Marcu, D. (1999). The automatic construction of large-scale corpora for summarization research. In *SIGIR 1999*, Berkeley, CA, pp. 137–144.

Marcu, D. (Ed.). (2000). *The Theory and Practice of Discourse Parsing and Summarization*. MIT Press.

McDonald, R. (2006). Discriminative sentence compression with soft syntactic constraints. In *EACL-06*.

McKeown, K. R., Passonneau, R., Elson, D., Nenkova, A., and Hirschberg, J. (2005). Do summaries help? A task-based evaluation of multi-document summarization. In *SIGIR 2005*, Salvador, Brazil.

Mellish, C., Knott, A., Oberlander, J., , and O'Donnell, M. (1998). Experiments using stochastic search for text planning. In *INLG 1998*, pp. 98–107.

Monz, C. (2004). Minimal span weighting retrieval for question answering. In *SIGIR Workshop on Information Retrieval for Question Answering*, pp. 23–30.

Moore, R. C. (2004). On log-likelihood-ratios and the significance of rare events. In *EMNLP 2004*, Barcelona, pp. 333–340.

Nenkova, A. and Vanderwende, L. (2005). The Impact of Frequency on Summarization. Tech. rep. MSR-TR-2005-101, Microsoft Research, Redmond, WA.

Nenkova, A. and McKeown, K. R. (2003). References to named entities: a corpus study. In *HLT-NAACL-03*, pp. 70–72.

Nenkova, A., Passonneau, R., and McKeown, K. R. (2007). The pyramid method: Incorporating human content selection variation in summarization evaluation. *ACM TSLP*, *4*(2).

Norvig, P. (2005). The gettysburgh powerpoint presentation. `http://norvig.com/Gettysburg/`.

Pang, B., Knight, K., and Marcu, D. (2003). Syntax-based alignment of multiple translations: extracting paraphrases and generating new sentences. In *HLT-NAACL-03*, pp. 102–109.

Pasca, M. (2003). *Open-Domain Question Answering from Large Text Collections*. CSLI.

Passonneau, R. (2006). Measuring agreement on set-valued items (masi) for semantic and pragmatic annotation. In *LREC-06*.

Passonneau, R., Nenkova, A., McKeown, K. R., and Sigleman, S. (2005). Applying the pyramid method in duc 2005. In *In Proceedings of the Document Understanding Conference (DUC'05)*.

Pollock, J. J. and Zamora, A. (1975). Automatic abstracting research at Chemical Abstracts Service. *Journal of Chemical Information and Computer Sciences*, *15*(4), 226–232.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, *14*(3), 130–127.

Quirk, C., Brockett, C., and Dolan, W. B. (2004). Monolingual machine translation for paraphrase generation. In *EMNLP 2004*, pp. 142–149.

Radev, D., Blair-Goldensohn, S., and Zhang, Z. (2001). Experiments in single and multi-document summarization using MEAD. In *DUC-01*, New Orleans, LA.

Radev, D. R., Jing, H., and Budzikowska, M. (2000). Summarization of multiple documents: clustering, sentence extraction, and evaluation. In *ANLP-NAACL Workshop on Automatic Summarization*, Seattle, WA.

Rau, L. F., Jacobs, P. S., and Zernik, U. (1989). Information extraction and text summarization using linguistic knowledge acquisition. *Information Processing and Management*, *25*(4), 419–428.

Ravichandran, D. and Hovy, E. (2002). Learning surface text patterns for a question answering system. In *ACL-02*, Philadelphia, PA, pp. 41–47.

Reimer, U. and Hahn, U. (1988). Text condensation as knowledge base abstraction. In *CAIA-88*, pp. 14–18.

Riezler, S., King, T. H., Crouch, R., and Zaenen, A. (2003). Statistical sentence condensation using ambiguity packing and stochastic disambiguation methods for Lexical-Functional Grammar. In *HLT-NAACL-03*, Edmonton, Canada.

Robinson, S. E. and Sparck Jones, K. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Science*, *27*, 129–146.

Rocchio, J. J. (1971). Relevance feedback in information retrieval. In *The SMART Retrieval System: Experiments in Automatic Indexing*, pp. 324–336. Prentice Hall.

Salton, G. (1971). *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice Hall.

Salton, G. and Buckley, C. (1990). Improving retrieval performance by relevance feedback. *Information Processing and Management*, *41*, 288–297.

Salton, G. and McGill, M. J. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY.

Sanderson, M. (1994). Word sense disambiguation and information retrieval. In *SIGIR-94*, Dublin, Ireland, pp. 142–151. ACM.

Schütze, H. and Pedersen, J. (1995). Information retrieval based on word senses. In *Proceedings of the Fourth Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, pp. 161–175.

Siddharthan, A., Nenkova, A., and McKeown, K. R. (2004). Syntactic simplification for improving content selection in multi-document summarization. In *COLING-04*, p. 896.

Simmons, R. F. (1965). Answering English questions by computer: A survey. *Communications of the ACM*, *8*(1), 53–70.

Soricut, R. and Marcu, D. (2006). Discourse generation using utility-trained coherence models. In *COLING/ACL 2006*, pp. 803–810.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, *28*(1), 11–21.

Sparck Jones, K. (2007). Automatic summarising: The state of the art. *Information Processing and Management*, *43*(6), 1449–1481.

Sparck Jones, K. and Willett, P. (Eds.). (1997). *Readings in Information Retrieval*. Morgan Kaufmann, San Francisco, CA.

Strzalkowski, T. and Harabagiu, S. (Eds.). (2006). *Advances in Open Domain Question Answering*. Springer.

Teufel, S. and van Halteren, H. (2004). Evaluating information content by factoid analysis: human annotation and stability. In *EMNLP 2004*, Barcelona.

Teufel, S. and Moens, M. (2002). Summarizing scientific articles: experiments with relevance and rhetorical status. *Computational Linguistics*, *28*(4), 409–445.

Turner, J. and Charniak, E. (2005). Supervised and unsupervised learning for sentence compression. In *ACL-05*, pp. 290–297.

van Halteren, H. and Teufel, S. (2003). Examining the consensus between human summaries: initial experiments with factoid analysis. In *HLT-NAACL-03 Workshop on Text Summarization*.

van Rijsbergen, C. J. (1975). *Information Retrieval*. Butterworths, London.

Vanderwende, L., Suzuki, H., Brockett, C., and Nenkova, A. (2007a). Beyond sumbasic: Task-focused summarization with sentence simplification and lexical expansion. *Information Processing and Management*, *43*(6), 1606–1618.

Vanderwende, L., Suzuki, H., Brockett, C., and Nenkova, A. (2007b). Beyond sumbasic: Task-focused summarization with sentence simplification and lexical expansion. *Information Processing and Management, Special issue on summarization*, *43*(6).

Voorhees, E. M. (1998). Using WordNet for text retrieval. In Fellbaum, C. (Ed.), *WordNet: An Electronic Lexical Database*, pp. 285–303. MIT Press.

Voorhees, E. M. and Harman, D. K. (2005). *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press.

Woods, W. A., Kaplan, R. M., and Nash-Webber, B. L. (1972). The lunar sciences natural language information system: Final report.. BBN Report 2378.

Zajic, D., Dorr, B., Lin, J., and Schwartz, R. (2007). Multi-candidate reduction: Sentence compression as a tool for document summarization tasks. *Information Processing and Management*, *43*(6), 1549–1570.

24 DIALOGUE AND CONVERSATIONAL AGENTS

C: I want you to tell me the names of the fellows on the St. Louis team.
A: I'm telling you. Who's on first, What's on second, I Don't Know is on third.
C: You know the fellows' names?
A: Yes.
C: Well, then, who's playing first?
A: Yes.
C: I mean the fellow's name on first.
A: Who.
C: The guy on first base.
A: Who is on first.
C: Well what are you askin' *me* for?
A: I'm not asking you – I'm telling you. Who is on first.

Who's on First – Bud Abbott and Lou Costello's version of an old burlesque standard.

CONVERSATION
DIALOGUE

The literature of the fantastic abounds in inanimate objects magically endowed with sentience and the gift of speech. From Ovid's statue of Pygmalion to Mary Shelley's Frankenstein, Cao Xue Qin's Divine Luminescent Stone-in-Waiting to Snow White's mirror, there is something deeply touching about creating something and then having a chat with it. Legend has it that after finishing his sculpture of *Moses*, Michelangelo thought it so lifelike that he tapped it on the knee and commanded it to speak. Perhaps this shouldn't be surprising. Language itself has always been the mark of humanity and sentience, and **conversation** or **dialogue** is the most fundamental and specially privileged arena of language. It is certainly the first kind of language we learn as children, and for most of us, it is the kind of language we most commonly indulge in, whether we are ordering curry for lunch or buying spinach, participating in business meetings or talking with our families, booking airline flights or complaining about the weather.

This chapter introduces the fundamental structures and algorithms in **conversational agents**. Conversational agents most often communicate via speech rather than text, and so they are also known as **spoken dialogue systems**, or **spoken language systems**. These programs communicate with users in spoken natural language in order to make travel arrangements, answer questions about weather or sports, route telephone calls, act as a general telephone assistant, or perform even more sophisticated tasks.

Many of these tasks involve telephony, and also involve mobile situations where a large screen and keyboard may not be available (Cohen and Oviatt, 1994). Applications relating to the domain of travel are thus very common. Indeed travel planning and management has been a key concern of dialogue systems since the early influential GUS system for planning airline travel (Bobrow et al., 1977). More recently, a number of spoken language systems for booking airline flights were built under the Air Traffic Information System (ATIS) (Hemphill et al., 1990) and Communicator (Walker, 2001) programs. We will use this travel domain as an example throughout the chapter (as we used it to discuss grammar rules for parsing ATIS sentences in Ch. 12). Fig. 24.1 shows a sample dialogue from a Communicator system, taken from Xu and Rudnicky (2000).

S1: Hello. You've reached the [Communicator]. Tell me your full name
U2: *hi i'd like to fly to seattle tuesday morning*
S3: Traveling to Seattle on Tuesday, August eleventh in the morning. Your full name?
U4: *[John Doe]*
S5: Hello, [John], I'm glad to hear from you again. You can now ask me for help.
S6: OK. Let's see I have a United flight departing Pittsburgh at ten forty five a.m. arriving into Seattle at one fifty seven p.m. Is that OK?
U7: *Yes*
S8: Will you return to Pittsburgh from Seattle?
U9: *no i will go to Los Angeles*
S10: Traveling to Los Angeles. on what date
U11: *Sorry can we go back to talk about the previous leg again*
S12: For the first leg, you are on a United flight 1446 departing Pittsburgh at ten forty five A.M. tomorrow, arriving in Seattle at one fifty seven P.M. Do you want to take that?
U13: *actually i want to leave on wednesday*
S14: a flight on wednesday.
S15: I have a flight departing Pittsburgh at ten forty five a.m., arrives Seattle at one fifty seven p.m. Is that OK?
U16: *Yes*

Figure 24.1 The travel domain: a fragment from a successful conversation between a user (U) and the Communicator system (S) of Xu and Rudnicky (2000).

Another promising domain is automatic call routing. A call routing system directs incoming calls in a telephone call center, transferring the call to the appropriate human. Call routing dialogue systems generally begin by asking the user open-ended questions like *How may I help you?* (Gorin et al., 1997), or *How may I direct your call?* (Chu-Carroll and Carpenter, 1999). Fig. 24.2 shows sample conversations from the AT&T How May I Help You (HMIHY) system (Gorin et al., 1997).

A somewhat different domain, which is very far from the telephony area, is automatic tutoring systems. Fig. 24.3 shows part of a dialogue from the ITSPOKE dialogue system (Litman and Silliman, 2004), which uses the Why2-Atlas text-based tutoring

S:	How may I help you?
U:	*can you tell me how much it is to tokyo?*
S:	You want to know the cost of a call?
U:	*yes that's right*
S:	Please hold on for rate information.
S:	How may I help you?
U:	*what's the area code for chicago?*
S:	You want an area code.
U:	*(silence)*
S:	The area code for Chicago is 312.

Figure 24.2 The call routing domain: two conversations between a user (U) and the AT&T How May I Help You (HMIHY) system (S) (Gorin et al., 1997).

system as its tutoring backend (VanLehn et al., 2002).

S:	I didn't understand. What force acts on all objects within close proximity to earth?
U:	*uh the force of gravity*
S:	Fine. Besides the Earth's gravitational force, are there any other forces acting on the person, keys, and elevator after the release?
U:	*no*
S:	Yep. Thus, keep in mind that after the release, the only force on the person, keys, and elevator is the force of gravity. If gravity is the only force acting on an object, will it be moving or staying still?
U:	*it will be moving*
S:	Fine. Not only are the person, keys, and elevator moving, they have only gravitational forces on them. When an object is falling and has only gravitational force on it, it is said to be in what?
U:	*freefall*

Figure 24.3 The tutoring domain: part of an conversation between a student user (U) and the ITSPOKE system (S) of Litman and Silliman (2004).

Sec. 24.1 starts out with a summary of facts about human conversation, including the idea of turns and utterances, speech acts, grounding, dialogue structure, and conversational implicature. The next few sections introduce the components of spoken language systems and some evaluation metrics. We then turn in Sec. 24.5 and Sec. 24.6 to the more sophisticated information-state and Markov decision processes models of conversational agents, and we conclude with some advanced topics like the BDI (belief-desire-intention) paradigm.

24.1 Properties of Human Conversations

Conversation between humans is an intricate and complex joint activity. Because of the limitations of our current technologies, conversations between humans and machines are vastly simpler and more constrained than these human conversations. Nonetheless, before we attempt to design a conversational agent to converse with humans, it is crucial to understand something about how humans converse with each other.

In this section we discuss some properties of human-human conversation that distinguish it from the kinds of (text-based) discourses we have seen so far. The main difference is that conversation is a kind of **joint activity** between two (or more) interlocutors. This basic fact has a number of ramifications; conversations are built up out of consecutive **turns**, each turn consists of **joint action** of the speaker and hearer, and the hearer make special inferences called **conversational implicatures** about the speaker's intended meaning.

24.1.1 Turns and Turn-Taking

TURN-TAKING

Dialogue is characterized by **turn-taking**; Speaker A says something, then speaker B, then speaker A, and so on. If having a turn (or "taking the floor") is a resource to be allocated, what is the process by which turns are allocated? How do speakers know when it is the proper time to contribute their turn?

It turns out that conversation and language itself are structured in such a way as to deal efficiently with this resource allocation problem. One source of evidence for this is the timing of the utterances in normal human conversations. While speakers can overlap each other while talking, it turns out that on average the total amount of overlap is remarkably small; perhaps less than 5% (Levinson, 1983). If speakers aren't overlapping, do they figure out when to talk by waiting for a pause after the other speaker finishes? This is also very rare. The amount of time between turns is quite small, generally less than a few hundred milliseconds even in multi-party discourse. Since it may take more than this few hundred milliseconds for the next speaker to plan the motor routines for producing their utterance, this means that speakers begin motor planning for their next utterance before the previous speaker has finished. For this to be possible, natural conversation must be set up in such a way that (most of the time) people can quickly figure out **who** should talk next, and exactly **when** they should talk. This kind of turn-taking behavior is generally studied in the field of **Conversation Analysis** (**CA**). In a key conversation-analytic paper, Sacks et al. (1974) argued that turn-taking behavior, at least in American English, is governed by a set of turn-taking rules. These rules apply at a **transition-relevance place**, or **TRP**; places where the structure of the language allows speaker shift to occur. Here is a version of the turn-taking rules simplified from Sacks et al. (1974):

CONVERSATION ANALYSIS

(24.1) **Turn-taking Rule.** At each TRP of each turn:

a. If during this turn the current speaker has selected A as the next speaker then A must speak next.
b. If the current speaker does not select the next speaker, any other speaker may take the next turn.
c. If no one else takes the next turn, the current speaker may take the next turn.

There are a number of important implications of rule (24.1) for dialogue modeling. First, subrule (24.1a) implies that there are some utterances by which the speaker specifically selects who the next speaker will be. The most obvious of these are questions, in which the speaker selects another speaker to answer the question. Two-part

ADJACENCY PAIRS
DIALOGIC PAIR

structures like QUESTION-ANSWER are called **adjacency pairs** (Schegloff, 1968) or **dialogic pair** (Harris, 2005). Other adjacency pairs include GREETING followed by GREETING, COMPLIMENT followed by DOWNPLAYER, REQUEST followed by GRANT. We will see that these pairs and the dialogue expectations they set up will play an important role in dialogue modeling.

SIGNIFICANT SILENCE

Subrule (24.1a) also has an implication for the interpretation of silence. While silence can occur after any turn, silence in between the two parts of an adjacency pair is **significant silence**. For example Levinson (1983) notes this example from Atkinson and Drew (1979); pause lengths are marked in parentheses (in seconds):

(24.2) A: Is there something bothering you or not?
(1.0)
A: Yes or no?
(1.5)
A: Eh?
B: No.

DISPREFERRED

Since A has just asked B a question, the silence is interpreted as a refusal to respond, or perhaps a **dispreferred** response (a response, like saying “no” to a request, which is stigmatized). By contrast, silence in other places, for example a lapse after a speaker finishes a turn, is not generally interpretable in this way. These facts are relevant for user interface design in spoken dialogue systems; users are disturbed by the pauses in dialogue systems caused by slow speech recognizers (Yankelovich et al., 1995).

UTTERANCE

Another implication of (24.1) is that transitions between speakers don’t occur just anywhere; the **transition-relevance places** where they tend to occur are generally at **utterance** boundaries. Recall from Ch. 12 that spoken utterances differ from written sentences in a number of ways. They tend to be shorter, are more likely to be single clauses or even just single words, the subjects are usually pronouns rather than full lexical noun phrases, and they include filled pauses and repairs. A hearer must take all this (and other cues like prosody) into account to know where to begin talking.

24.1.2 Language as Action: Speech Acts

The previous section showed that conversation consists of a sequence of turns, each of which consists of one or more utterance. A key insight into conversation due to Wittgenstein (1953) but worked out more fully by Austin (1962) is that an utterance in a dialogue is a kind of **action** being performed by the speaker.

PERFORMATIVE

The idea that an utterance is a kind of action is particularly clear in **performative** sentences like the following:

(24.3) I name this ship the *Titanic*.

(24.4) I second that motion.

(24.5) I bet you five dollars it will snow tomorrow.

When uttered by the proper authority, for example, (24.3) has the effect of changing the state of the world (causing the ship to have the name *Titanic*) just as any action can change the state of the world. Verbs like *name* or *second* which perform this kind of action are called performative verbs, and Austin called these kinds of actions **speech**

SPEECH ACTS

acts. What makes Austin's work so far-reaching is that speech acts are not confined to this small class of performative verbs. Austin's claim is that the utterance of any sentence in a real speech situation constitutes three kinds of acts:

- **locutionary act:** the utterance of a sentence with a particular meaning.
- **illocutionary act:** the act of asking, answering, promising, etc., in uttering a sentence.
- **perlocutionary act:** the (often intentional) production of certain effects upon the feelings, thoughts, or actions of the addressee in uttering a sentence.

ILLOCUTIONARY FORCE

For example, Austin explains that the utterance of example (24.6) might have the **illocutionary force** of protesting and the perlocutionary effect of stopping the addressee from doing something, or annoying the addressee.

(24.6) You can't do that.

The term **speech act** is generally used to describe illocutionary acts rather than either of the other two types of acts. Searle (1975b), in modifying a taxonomy of Austin's, suggests that all speech acts can be classified into one of five major classes:

- **Assertives:** committing the speaker to something's being the case (*suggesting*, *putting forward*, *swearing*, *boasting*, *concluding*).
- **Directives:** attempts by the speaker to get the addressee to do something (*asking*, *ordering*, *requesting*, *inviting*, *advising*, *begging*).
- **Commissives:** committing the speaker to some future course of action (*promising*, *planning*, *vowing*, *betting*, *opposing*).
- **Expressives:** expressing the psychological state of the speaker about a state of affairs *thanking*, *apologizing*, *welcoming*, *deploring*.
- **Declarations:** bringing about a different state of the world via the utterance (including many of the performative examples above; *I resign*, *You're fired*.)

24.1.3 Language as Joint Action: Grounding

COMMON GROUND

GROUND

The previous section suggested that each turn or utterance could be viewed as an action by a speaker. But dialogue is not a series of unrelated independent acts. Instead, dialogue is a collective act performed by the speaker and the hearer. One implication of joint action is that, unlike in monologue, the speaker and hearer must constantly establish **common ground** (Stalnaker, 1978), the set of things that are mutually believed by both speakers. The need to achieve common ground means that the hearer must **ground** the speaker's utterances, making it clear that the hearer has understood the speaker's meaning and intention.

As Clark (1996) points out, people need closure or grounding for non-linguistic actions as well. For example, why does a well-designed elevator button light up when it's pressed? Because this indicates to the would-be elevator traveler that she has successfully called the elevator. Clark phrases this need for closure as follows (after (Norman, 1988)):

> **Principle of closure.** Agents performing an action require evidence, sufficient for current purposes, that they have succeeded in performing it.

Grounding is also important when the hearer needs to indicate that the speaker has *not* succeeded in performing an action. If the hearer has problems in understanding, she must indicate these problems to the speaker, again so that mutual understanding can eventually be achieved.

How is closure achieved? Clark and Schaefer (1989) introduce the idea that each joint linguistic act or **contribution** has two phases, called **presentation** and **acceptance**. In the first phase, a speaker presents the hearer with an utterance, performing a sort of speech act. In the acceptance phase, the hearer has to ground the utterance, indicating to the speaker whether understanding was achieved.

CONTRIBUTION

What methods can the hearer (call her B) use to ground the speaker A's utterance? Clark and Schaefer (1989) discuss five main types of methods, ordered from weakest to strongest:

1. **Continued attention:** B shows she is continuing to attend and therefore remains satisfied with A's presentation.
2. **Relevant next contribution:** B starts in on the next relevant contribution.
3. **Acknowledgement:** B nods or says a continuer like *uh-huh*, *yeah*, or the like, or an **assessment** like *that's great*.
4. **Demonstration:** B demonstrates all or part of what she has understood A to mean, for example by **reformulating** (paraphrasing) A's utterance, or by **collaborative completion** of A's utterance.
5. **Display:** B displays verbatim all or part of A's presentation.

REFORMULATING

COLLABORATIVE COMPLETION

Let's look for examples of these in a human-human dialogue example. We'll be returning to this example throughout the chapter; in order to design a more sophisticated machine dialogue agent, it helps to look at how a human agent performs similar tasks. Fig. 24.4 shows part of a dialogue between a human travel agent and a human client.

C_1: . . . I need to travel in May.

A_1: And, what day in May did you want to travel?

C_2: OK uh I need to be there for a meeting that's from the 12th to the 15th.

A_2: And you're flying into what city?

C_3: Seattle.

A_3: And what time would you like to leave Pittsburgh?

C_4: Uh hmm I don't think there's many options for non-stop.

A_4: Right. There's three non-stops today.

C_5: What are they?

A_5: The first one departs PGH at 10:00am arrives Seattle at 12:05 their time. The second flight departs PGH at 5:55pm, arrives Seattle at 8pm. And the last flight departs PGH at 8:15pm arrives Seattle at 10:28pm.

C_6: OK I'll take the 5ish flight on the night before on the 11th.

A_6: On the 11th? OK. Departing at 5:55pm arrives Seattle at 8pm, U.S. Air flight 115.

C_7: OK.

Figure 24.4 Part of a conversation between a travel agent (A) and client (C).

Utterance A_1, in which the agent repeats *in May*, repeated below in boldface, shows the strongest form of grounding, in which the hearer displays their understanding by repeating verbatim part of the speakers words:

C_1: . . . I need to travel **in May**.

A_1: And, what day **in May** did you want to travel?

This particular fragment doesn't have an example of an *acknowledgement*, but there's an example in another fragment:

C: He wants to fly from Boston

A: **Mm hmm**

C: to Baltimore Washington International

CONTINUER
BACKCHANNEL

The word *mm-hmm* here is a **continuer**, also often called a **backchannel** or an **acknowledgement token**. A continuer is a (short) optional utterance which acknowledges the content of the utterance of the other, and which doesn't require an acknowledgement by the other (Yngve, 1970; Jefferson, 1984; Schegloff, 1982; Ward and Tsukahara, 2000).

In Clark and Schaefer's third method, the speaker starts in on their relevant next contribution. We see a number of examples of this in the sample dialogue above, for example where the speaker asks a question and the hearer answers it. We mentioned these **adjacency pairs** above; other examples include PROPOSAL followed by ACCEPTANCE or REJECTION, APOLOGY followed by ACCEPTANCE/REJECTION, SUMMONS followed by ANSWER, and so on.

In a more subtle but very important kind of grounding act, the speaker can combine this method with the previous one. For example notice that whenever the client answers a question, the agent begins the next question with *And*. The *And* indicates to the client that the agent has successfully understood the answer to the last question:

And, what day in May did you want to travel?

...

And you're flying into what city?

...

And what time would you like to leave Pittsburgh?

As we will see in Sec. 24.5, the notions of grounding and contributions can be combined with speech acts to give a more sophisticated model of joint action in conversation; these more sophisticated models are called **dialogue acts**.

Grounding is just as crucial in human-machine conversation as it is in human conversation. The examples below, from Cohen et al. (2004), suggest how unnatural it sounds when a machine doesn't ground properly. The use of *Okay* makes (24.7) a much more natural response than (24.8) to ground a user's rejection:

(24.7) System: Did you want to review some more of your personal profile?
Caller: No.
System: *Okay,* what's next?

(24.8) System: Did you want to review some more of your personal profile?
Caller: No.
System: What's next?

Indeed, this kind of lack of grounding can cause errors. Stifelman et al. (1993) and Yankelovich et al. (1995) found that humans get confused when a conversational system system doesn't give explicit acknowledgements.

24.1.4 Conversational Structure

We have already seen how conversation is structured by adjacency pairs and contributions. Here we'll briefly discuss one aspect of the **overall organization** of a conversation: conversational openings. The openings of telephone conversations, for example, tend to have a 4-part structure (Clark, 1994; Schegloff, 1968, 1979):

Stage 1: Enter a conversation, with summons-response adjacency pair

Stage 2: Identification

Stage 3: Establish joint willingness to converse

Stage 4: The first topic is raised, usually by the caller.

These four stages appear in the opening of this short task-oriented conversation from Clark (1994).

Stage	Speaker & Utterance
1	A_1: (rings B's telephone)
1,2	B_1: Benjamin Holloway
2	A_1: this is Professor Dwight's secretary, from Polymania College
2,3	B_1: ooh yes –
4	A_1: uh:m . about the: lexicology *seminar*
4	B_1: *yes*

It is common for the person who answers the phone to speak first (since the caller's ring functions as the first part of the adjacency pair) but for the caller to bring up the first topic, as the caller did above concerning the "lexicology seminar". This fact that the caller usually brings up the first topic causes confusion when the answerer brings up the first topic instead; here's an example of this from the British directory enquiry service from Clark (1994):

Customer:	(rings)
Operator:	Directory Enquiries, for which town please?
Customer:	Could you give me the phone number of um: Mrs. um: Smithson?
Operator:	Yes, which town is this at please?
Customer:	Huddleston.
Operator:	Yes. And the name again?
Customer:	Mrs. Smithson.

In the conversation above, the operator brings up the topic (*for which town please?*) in her first sentence, confusing the caller, who ignores this topic and brings up her own. This fact that callers expect to bring up the topic explains why conversational agents for call routing or directory information often use very open prompts like *How may I help you?* or *How may I direct your call?* rather than a directive prompt like *For which town please?*. Open prompts allow the caller to state their own topic, reducing recognition errors caused by customer confusion.

Conversation has many other kinds of structure, including the intricate nature of conversational closings and the wide use of presequences. We will discuss structure based on **coherence** in Sec. 24.7.

24.1.5 Conversational Implicature

We have seen that conversation is a kind of joint activity, in which speakers produce turns according to a systematic framework, and that the contributions made by these turns include a presentation phase of performing a kind of action, and an acceptance phase of grounding the previous actions of the interlocutor. So far we have only talked about what might be called the 'infrastructure' of conversation. But we have so far said nothing about the actual information that gets communicated from speaker to hearer in dialogue.

While Ch. 17 showed how we can compute meanings from sentences, it turns out that in conversation, the meaning of a contribution is often quite a bit extended from the compositional meaning that might be assigned from the words alone. This is because inference plays a crucial role in conversation. The interpretation of an utterance relies on more than just the literal meaning of the sentences. Consider the client's response C_2 from the sample conversation in Fig. 24.4, repeated here:

A_1: And, what day in May did you want to travel?

C_2: OK uh I need to be there for a meeting that's from the 12th to the 15th.

Notice that the client does not in fact answer the question. The client merely states that he has a meeting at a certain time. The semantics for this sentence produced by a semantic interpreter will simply mention this meeting. What is it that licenses the agent to infer that the client is mentioning this meeting so as to inform the agent of the travel dates?

Now consider another utterance from the sample conversation, this one by the agent:

A_4: . . . There's three non-stops today.

Now this statement would still be true if there were seven non-stops today, since if there are seven of something, there are by definition also three. But what the agent means here is that there are three **and not more than three** non-stops today. How is the client to infer that the agent means **only three** non-stops?

These two cases have something in common; in both cases the speaker seems to expect the hearer to draw certain inferences; in other words, the speaker is communicating more information than seems to be present in the uttered words. These kind of examples were pointed out by Grice (1975, 1978) as part of his theory of **conversational** IMPLICATURE **implicature**. **Implicature** means a particular class of licensed inferences. Grice proposed that what enables hearers to draw these inferences is that conversation is guided MAXIMS by a set of **maxims**, general heuristics which play a guiding role in the interpretation of conversational utterances. He proposed the following four maxims:

QUANTITY

- **Maxim of Quantity:** Be exactly as informative as is required:
  1. Make your contribution as informative as is required (for the current purposes of the exchange).

2. Do not make your contribution more informative than is required.

QUALITY

- **Maxim of Quality:** Try to make your contribution one that is true:
  1. Do not say what you believe to be false.
  2. Do not say that for which you lack adequate evidence.

RELEVANCE

- **Maxim of Relevance:** Be relevant.

MANNER

- **Maxim of Manner:** Be perspicuous:
  1. Avoid obscurity of expression.
  2. Avoid ambiguity.
  3. Be brief (avoid unnecessary prolixity).
  4. Be orderly.

It is the Maxim of Quantity (specifically Quantity 1) that allows the hearer to know that *three non-stops* did not mean *seven non-stops*. This is because the hearer assumes the speaker is following the maxims, and thus if the speaker meant seven non-stops she would have said seven non-stops ("as informative as is required"). The Maxim of Relevance is what allows the agent to know that the client wants to travel by the 12th. The agent assumes the client is following the maxims, and hence would only have mentioned the meeting if it was relevant at this point in the dialogue. The most natural inference that would make the meeting relevant is the inference that the client meant the agent to understand that his departure time was before the meeting time.

24.2 Basic Dialogue Systems

We've now seen a bit about how human dialogue works, although as we'll see, not every aspect of human-human conversation is modeled in human-machine conversation. Let's therefore turn now to the spoken dialogue systems used in commercial applications today.

Fig. 24.5 shows a typical architecture for a dialogue system. It has six components. The speech recognition and understanding components extract meaning from the input, while the generation and TTS components map from meaning to speech. The dialogue manager controls the whole process, along with a task manager which has knowledge about the task domain (such as air travel). We'll go through each of these components in the next sections. Then we'll explore more sophisticated research systems in following sections.

24.2.1 ASR component

The ASR (automatic speech recognition) component takes audio input, generally from a telephone, or from a PDA or desktop microphone, and returns a transcribed string of words, as discussed in chapter Ch. 9.

Various aspects of the ASR system may be optimized specifically for use in conversational agents. For example, the large vocabulary speech recognizers we discussed in Ch. 9 for dictation or transcription focused on transcribing any sentence on any topic using any English word. But for domain-dependent dialogue systems it is of little use

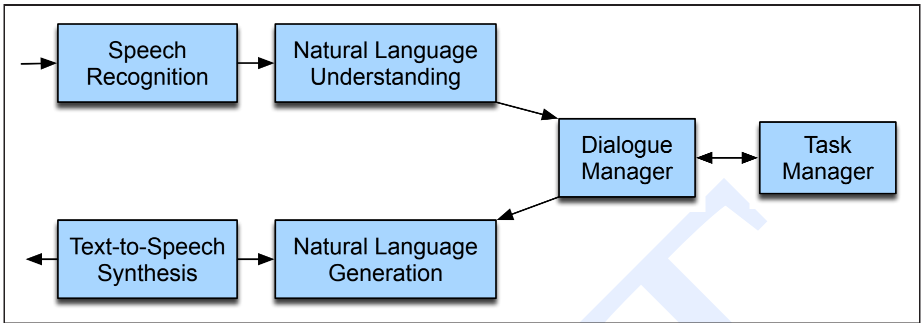


Figure 24.5 Simplified architecture of the components of a conversational agent.

to be able to transcribe such a wide variety of sentences. The sentences that the speech recognizer needs to be able to transcribe need are just those that can be understood by the natural language understanding component. For this reason commercial dialogue systems generally use non-probabilistic language models based on finite-state grammars. These grammars are generally hand-written, and specify all possible responses that the system understands. We'll see an example of such a hand-written grammar for a VoiceXML system in Sec. 24.3. Such grammars-based language models can also be compiled automatically from, e.g., unification grammars used for natural language understanding (Rayner et al., 2006).

Because what the user says to the system is related to what the system has just said, language models in conversational agent are usually *dialogue-state dependent*. For example, if the system has just asked the user "What city are you departing from?", the ASR language model can be constrained to only consist of city names, or perhaps sentences of the form 'I want to (leave|depart) from [CITYNAME]'. These dialogue-state-specific language models often consist of hand-written finite-state (or even context-free) grammars as discussed above, one for each dialogue state.

In some systems, the understanding component is more powerful, and the set of sentences the system can understand is larger. In such cases, instead of a finite-state grammar, we can use an N-gram language model whose probabilities are similarly conditioned on the dialogue state.

RESTRICTIVE GRAMMAR

Whether we use a finite-state, context-free, or an N-gram language model, we call such a dialogue-state dependent language model a **restrictive grammar**. When the system wants to constrain the user to respond to the system's last utterance, it can use a restrictive grammar. When the system wants to allow the user more options, it might mix this state-specific language model with a more general language model. As we will see, the choice between these strategies can be tuned based on how much *initiative* the user is allowed.

Speech recognition in dialogue, as well as in many other applications like dictation, has the advantage that the identity of the speaker remains constant across many utterances. This means that speaker adaptation techniques like MLLR and VTLN (Ch. 9) can be applied to improve recognition as the system hears more and more speech from the user.

Embedding an ASR engine in a dialogue system also requires that an ASR engine to have realtime response, since users are unwilling to accept long pauses before responses. Dialogue systems also generally require that an ASR system return a **confidence** value for a sentence, which can then be used for example for deciding whether to ask the user to confirm a response.

24.2.2 NLU component

The NLU (natural language understanding) component of dialogue systems must produce a semantic representation which is appropriate for the dialogue task. Many speech-based dialogue systems, since as far back as the GUS system (Bobrow et al., 1977), are based on the frame-and-slot semantics discussed in Chapter 15. A travel system, for example, which has the goal of helping a user find an appropriate flight, would have a frame with slots for information about the flight; thus a sentence like *Show me morning flights from Boston to San Francisco on Tuesday* might correspond to the following filled-out frame (from Miller et al. (1994)):

```
SHOW:
FLIGHTS:
    ORIGIN:
        CITY:  Boston
        DATE:
            DAY-OF-WEEK:  Tuesday
        TIME:
            PART-OF-DAY:  morning
    DEST:
        CITY:  San Francisco
```

How does the NLU component generate this semantic representation? Some dialogue systems use general-purpose unification grammars with semantic attachments, such as the Core Language Engine introduced in Ch. 18. A parser produces a sentence meaning, from which the slot-fillers are extracted (Lewin et al., 1999).

Other dialogue systems rely on simpler domain-specific semantic analyzers, such as **semantic grammars**. A semantic grammar is a CFG in which the actual node names in the parse tree correspond to the semantic entities which are being expressed, as in the following grammar fragments:

```
SHOW               →  show me | i want | can i see|...
DEPART_TIME_RANGE  →  (after|around|before) HOUR |
                      morning | afternoon | evening
HOUR               →  one|two|three|four...|twelve (AMPM)
FLIGHTS            →  (a) flight | flights
AMPM               →  am | pm
ORIGIN             →  from CITY
DESTINATION        →  to CITY
CITY               →  Boston | San Francisco | Denver | Washington
```

These grammars take the form of context-free grammars or recursive transition networks (Issar and Ward, 1993; Ward and Issar, 1994), and hence can be parsed by any standard CFG parsing algorithm, such as the CKY or Earley algorithms introduced in Ch. 13. The result of the CFG or RTN parse is a hierarchical labeling of the input string with semantic node labels:

```
SHOW     FLIGHTS       ORIGIN  DESTINATION       DEPART_DATE  DEPART_TIME
                               to CITY
Show me  flights  from boston  to san francisco  on tuesday   morning
```

Since semantic grammar nodes like ORIGIN correspond to the slots in the frame, the slot-fillers can be read almost directly off the resulting parse above. It remains only to put the fillers into some sort of canonical form (for example dates can be **normalized** into a DD:MM:YY form, times can be put into 24-hour time, etc).

NORMALIZED

The semantic grammar approach is very widely used, but is unable to deal with ambiguity, and requires hand-written grammars that can be expensive and slow to create.

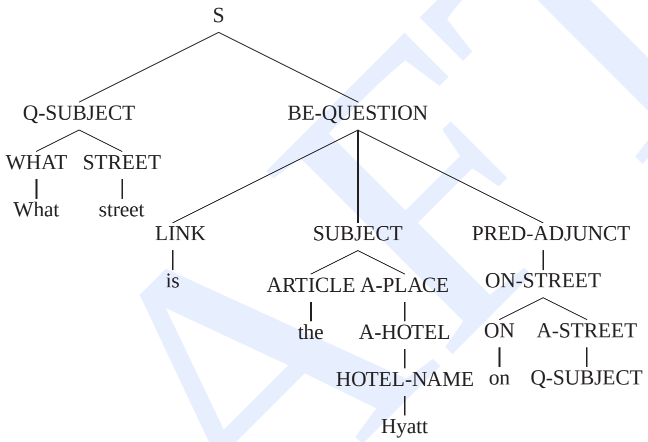


Figure 24.6 A parse of a sentence in the TINA semantic grammar, after Seneff (1995).

Ambiguity can be addressed by adding probabilities to the grammar; one such probabilistic semantic grammar system is the TINA system (Seneff, 1995) shown in Fig. 24.6; note the mix of syntactic and semantic node names. The grammar rules in TINA are written by hand, but parse tree node probabilities are trained by a modified version of the SCFG method described in Ch. 14.

An alternative to semantic grammars that is probabilistic and also avoids hand-coding of grammars is the semantic HMM model of Pieraccini et al. (1991). The hidden states of this HMM are semantic slot labels, while the observed words are the fillers of the slots. Fig. 24.7 shows how a sequence of hidden states, corresponding to slot names, could be decoded from (or could generate) a sequence of observed words. Note that the model includes a hidden state called DUMMY which is used to generate words which do not fill any slots in the frame.

The goal of the HMM model is to compute the labeling of semantic roles $C = c_1, c_2, \ldots, c_i$ (C for 'cases' or 'concepts') that has the highest probability $P(C|W)$ given some words $W = w_1, w_2, \ldots, w_n$. As usual, we use Bayes Rule as follows:

$$\begin{aligned}\operatorname*{argmax}_C P(C|W) &= \operatorname*{argmax}_C \frac{P(W|C)P(C)}{P(W} \\ &= \operatorname*{argmax}_C P(W|C)P(C) \qquad (24.9)\end{aligned}$$

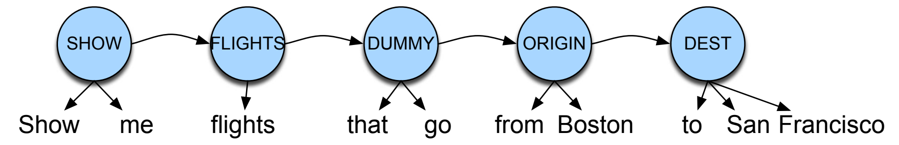


Figure 24.7 The Pieraccini et al. (1991) HMM model of semantics for filling slots in frame-based dialogue systems. Each hidden state can generate a sequence of words; such a model, in which a single hidden state can correspond to multiple observations, is technically called a **semi-HMM**.

$$= \prod_{i=2}^{N} P(w_i|w_{i-1}...w_1,C)P(w_1|C)\prod_{i=2}^{M} P(c_i|c_{i-1}...c_1) \tag{24.10}$$

The Pieraccini et al. (1991) model makes a simplification that the concepts (the hidden states) are generated by a Markov process (a concept M-gram model), and that the observation probabilities for each state are generated by a state-dependent (concept-dependent) word N-gram word model:

$$P(w_i|w_{i-1},...,w_1,C) = P(w_i|w_{i-1},...,w_{i-N+1},c_i) \tag{24.11}$$
$$P(c_i|c_{i-1},...,c_1) = P(c_i|c_{i-1},...,c_{i-M+1}) \tag{24.12}$$

Based on this simplifying assumption, the final equations used in the HMM model are as follows:

$$\underset{C}{\text{argmax}}\, P(C|W) = \prod_{i=2}^{N} P(w_i|w_{i-1}...w_{i-N+1},c_i)\prod_{i=2}^{M} P(c_i|c_{i-1}...c_{i-M+1}) \tag{24.13}$$

These probabilities can be trained on a labeled training corpus, in which each sentence is hand-labeled with the concepts/slot-names associated with each string of words. The best sequence of concepts for a sentence, and the alignment of concepts to word sequences, can be computed by the standard Viterbi decoding algorithm.

In summary, the resulting HMM model is a generative model with two components. The $P(C)$ component represents the choice of what meaning to express; it assigns a prior over sequences of semantic slots, computed by a concept N-gram. $P(W|C)$ represents the choice of what words to use to express that meaning; the likelihood of a particular string of words being generated from a given slot. It is computed by a word N-gram conditioned on the semantic slot. This model is very similar to the HMM model for **named entity** detection we saw in Ch. 22. Technically, HMM models like this, in which each hidden state correspond to multiple output observations, are called **semi-HMMs**. In a classic HMM, by contrast, each hidden state corresponds to a single output observation.

SEMI-HMMS

Many other kinds of statistical models have been proposed for the semantic understanding component of dialogue systems. These include the Hidden Understanding

Model (HUM), which adds hierarchical structure to the HMM to combine the advantages of the semantic grammar and semantic HMM approaches (Miller et al., 1994, 1996, 2000), or the decision-list method of Rayner and Hockey (2003).

24.2.3 Generation and TTS components

The generation component of a conversational agent chooses the concepts to express to the user, plans out how to express these concepts in words, and assigns any necessary prosody to the words. The TTS component then takes these words and their prosodic annotations and synthesizes a waveform, as described in Ch. 8.

The generation task can be separated into two tasks: *what to say*, and *how to say it*. The **content planner** module addresses the first task, decides what content to express to the user, whether to ask a question, present an answer, and so on. The content planning component of dialogue systems is generally merged with the dialogue manager, and we will return to it below.

The **language generation** module addresses the second task, choosing the syntactic structures and words needed to express the meaning. Language generation modules are implemented in one of two ways. In the simplest and most common method, all or most of the words in the sentence to be uttered to the user are prespecified by the dialogue designer. This method is known as template-based generation, and the sentences created by these templates are often called **prompts**. While most of the words in the template are fixed, templates can include some variables which are filled in by the generator, as in the following:

PROMPTS

What time do you want to leave CITY-ORIG?
Will you return to CITY-ORIG from CITY-DEST?

A second method for language generation relies on techniques from the field **natural language generation**. Here the dialogue manager builds a representation of the meaning of the utterance to be expressed, and passes this meaning representation to a full generator. Such generators generally have three components, a sentence planner, surface realizer, and prosody assigner. A sketch of this architecture is shown in Fig. 24.8. See Reiter and Dale (2000) for further information on natural language generation systems and their use in dialogue.

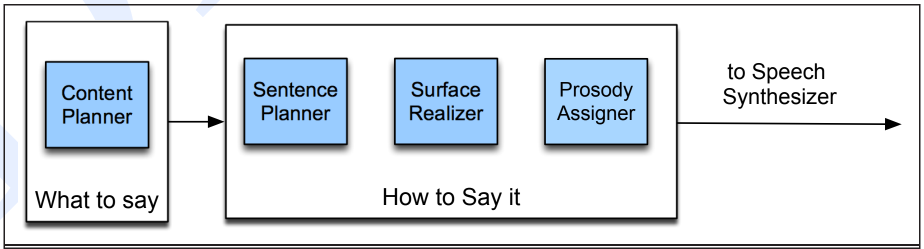


Figure 24.8 Architecture of a natural language generation system for a dialogue system, after Walker and Rambow (2002).

In the hand-designed prompts that are common in current systems, there are a number of important conversational and discourse constraints that must be implemented.

Like any discourse, a conversation needs to be coherent (Ch. 21). For example, as Cohen et al. (2004) show, the use of discourse markers and pronouns in the hand-built system prompts makes the dialogue in (24.15) more natural than the dialogue in (24.14):

(24.14) Please say the data.
...
Please say the start time.
...
Please say the duration.
...
Please say the subject.

(24.15) First, tell me the date.
...
Next, I'll need the time it starts.
...
Thanks. <pause> Now, how long is it supposed to last?
...
Last of all, I just need a brief description...

TAPERED PROMPTS

Another important case of discourse coherence occurs when particular prompts may need to be said to the user repeatedly. In these cases, it is standard in dialogue systems to use **tapered prompts**, prompts which get incrementally shorter. The following example from Cohen et al. (2004) shows a series of (hand-designed) tapered prompts:

(24.16) System: Now, what's the first company to add to your watch list?
Caller: Cisco
System: What's the next company name? (Or, you can say, "Finished.")
Caller: IBM
System: Tell me the next company name, or say, "Finished."
Caller: Intel
System: Next one?
Caller: America Online.
System: Next?
Caller: ...

Other constraints on generation are more specific to spoken dialogue, and refer to facts about human memory and attentional processes. For example, when humans are prompted to give a particular response, it taxes their memory less if the suggested response is the last thing they hear. Thus as Cohen et al. (2004) point out, the prompt "To hear the list again, say 'Repeat list'" is easier for users than "Say 'Repeat list' to hear the list again."

Similarly, presentation of long lists of query results (e.g., potential flights, or movies) can tax users. Thus most dialogue systems have content planning rules to deal with this. In the Mercury system for travel planning described in Seneff (2002), for example, a content planning rule specifies that if there are more than three flights to describe to the user, the system will just list the available airlines and describe explicitly only the earliest flight.

24.2.4 Dialogue Manager

The final component of a dialogue system is the dialogue manager, which controls the architecture and structure of the dialogue. The dialogue manager takes input from the ASR/NLU components, maintains some sort of state, interfaces with the task manager, and passes output to the NLG/TTS modules.

We saw a very simple dialogue manager in Chapter 2's ELIZA, whose architecture was a simple read-substitute-print loop. The system read in a sentence, applied a series of text transformations to the sentence, and then printed it out. No state was kept; the transformation rules were only aware of the current input sentence. In addition to its ability to interact with a task manager, a modern dialogue manager is very different than ELIZA's manager in both the amount of state that the manager keeps about the conversation, and the ability of the manager to model structures of dialogue above the level of a single response.

Four kinds of dialogue management architectures are most common. The simplest and most commercially developed architectures, finite-state and frame-based, are discussed in this section. Later sections discuss the more powerful information-state dialogue managers, including a probabilistic version of information-state managers based on Markov Decision Processes, and finally the more classic plan-based architectures.

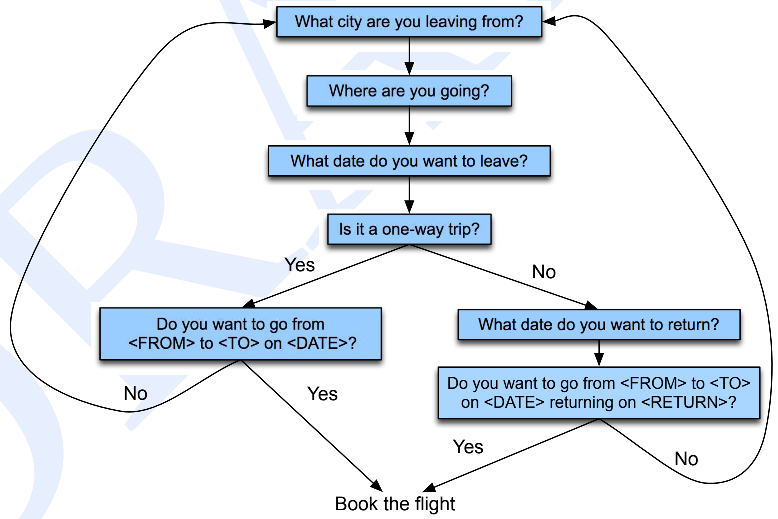

What city are you leaving from?
Where are you going?
What date do you want to leave?
Is it a one-way trip?
Yes
No
Do you want to go from <FROM> to <TO> on <DATE>?
What date do you want to return?
No
Yes
Do you want to go from <FROM> to <TO> on <DATE> returning on <RETURN>?
Yes
No
Book the flight

Figure 24.9 A simple finite-state automaton architecture for a dialogue manager.

The simplest dialogue manager architecture is a finite-state manager. For example, imagine a trivial airline travel system whose job was to ask the user for a departure city, a destination city, a time, and whether the trip was round-trip or not. Fig. 24.9 shows a sample dialogue manager for such a system. The states of the FSA correspond to questions that the dialogue manager asks the user, and the arcs correspond to actions to take depending on what the user responds. This system completely controls the conversation with the user. It asks the user a series of questions, ignoring (or misinterpreting) anything the user says that is not a direct answer to the system's question, and then going on to the next question.

SYSTEM INITIATIVE
SINGLE INITIATIVE
INITIATIVE
UNIVERSAL

Systems that control the conversation in this way are called **system initiative** or **single initiative** systems. We say that the speaker that is in control of the conversation has the **initiative**; in normal human-human dialogue, initiative shifts back and forth between the participants (Walker and Whittaker, 1990).[1] The limited single-initiative finite-state dialogue manager architecture has the advantage that the system always knows what question the user is answering. This means the system can prepare the speech recognition engine with a specific language model tuned to answers for this question. Knowing what the user is going to be talking about also makes the task of the natural language understanding engine easier. Most finite-state systems also allow allow **universal** commands. Universals are commands that can be said anywhere in the dialogue; every dialogue state recognizes the universal commands in addition to the answer to the question that the system just asked. Common universals include **help**, which gives the user a (possibly state-specific) help message, **start over** (or **main menu**), which returns the user to some specified main start state, and some sort of command to correct the system's understanding of the users last statement (San-Segundo et al., 2001). System-initiative finite-state dialogue managers with universals may be sufficient for very simple tasks such as entering a credit card number, or a name and password, on the phone.

Pure system-initiative finite-state dialogue manager architectures are probably too restricted, however, even for the relatively uncomplicated task of a spoken dialogue travel agent system. The problem is that pure system-initiative systems require that the user answer exactly the question that the system asked. But this can make a dialogue awkward and annoying. Users often need to be able to say something that is not exactly the answer to a single question from the system. For example, in a travel planning situation, users often want to express their travel goals with complex sentences that may answer more than one question at a time, as in Communicator example (24.17) repeated from Fig. 24.1, or ATIS example (24.18).

(24.17) Hi I'd like to fly to Seattle Tuesday morning

(24.18) I want a flight from Milwaukee to Orlando one way leaving after five p.m. on Wednesday.

A finite state dialogue system, as typically implemented, can't handle these kinds of utterances since it requires that the user answer each question as it is asked. Of course it is theoretically possible to create a finite state architecture which has a separate state for each possible subset of questions that the user's statement could be answering, but this would require a vast explosion in the number of states, making this a difficult architecture to conceptualize.

MIXED INITIATIVE

Therefore, most systems avoid the pure system-initiative finite-state approach and use an architecture that allows **mixed initiative**, in which conversational initiative can shift between the system and user at various points in the dialogue.

[1] Single initiative systems can also be controlled by the user, in which case they are called **user initiative** systems. Pure user initiative systems are generally used for stateless database querying systems, where the user asks single questions of the system, which the system converts into SQL database queries, and returns the results from some database.

FRAME-BASED
FORM-BASED

One common mixed initiative dialogue architecture relies on the structure of the frame itself to guide the dialogue. These **frame-based** or **form-based** dialogue managers asks the user questions to fill slots in the frame, but allow the user to guide the dialogue by giving information that fills other slots in the frame. Each slot may be associated with a question to ask the user, of the following type:

Slot	Question
ORIGIN CITY	"From what city are you leaving?"
DESTINATION CITY	"Where are you going?"
DEPARTURE TIME	"When would you like to leave?"
ARRIVAL TIME	"When do you want to arrive?"

A frame-based dialogue manager thus needs to ask questions of the user, filling any slot that the user specifies, until it has enough information to perform a data base query, and then return the result to the user. If the user happens to answer two or three questions at a time, the system has to fill in these slots and then remember not to ask the user the associated questions for the slots. Not every slot need have an associated question, since the dialogue designer may not want the user deluged with questions. Nonetheless, the system must be able to fill these slots if the user happens to specify them. This kind of form-filling dialogue manager thus does away with the strict constraints that the finite-state manager imposes on the order that the user can specify information.

While some domains may be representable with a single frame, others, like the travel domain, seem to require the ability to deal with multiple frames. In order to handle possible user questions, we might need frames with general route information (for questions like *Which airlines fly from Boston to San Francisco?*), information about airfare practices (for questions like *Do I have to stay a specific number of days to get a decent airfare?*) or about car or hotel reservations. Since users may switch from frame to frame, the system must be able to disambiguate which slot of which frame a given input is supposed to fill, and then switch dialogue control to that frame.

Because of this need to dynamically switch control, frame-based systems are often implemented as **production rule** systems. Different types of inputs cause different productions to fire, each of which can flexibly fill in different frames. The production rules can then switch control based on factors such as the user's input and some simple dialogue history like the last question that the system asked. The Mercury flight reservation system (Seneff and Polifroni, 2000; Seneff, 2002) uses a large 'dialogue control table' to store 200-350 rules, covering request for help, rules to determine if the user is referring to a flight in a list ("I'll take that nine a.m. flight"), and rules to decide which flights to describe to the user first.

Now that we've seen the frame-based architecture, let's return to our discussion of conversational initiative. It's possible in the same agent to allow system-initiative, user-initiative, and mixed-initiative interactions. We said earlier that initiative refers to who has control of the conversation at any point. The phrase **mixed initiative** is generally used in two ways. It can mean that the system or the user could arbitrarily take or give up the initiative in various ways (Walker and Whittaker, 1990; Chu-Carroll and Brown, 1997). This kind of mixed initiative is difficult to achieve in current dialogue systems. In form-based dialogue system, the term mixed initiative is used for a more limited kind of shift, operationalized based on a combination of prompt type (open versus directive) and the type of grammar used in the ASR. An **open prompt** is one in which the system gives the user very few constraints, allowing the user to respond however they please, as in:

OPEN PROMPT

How may I help you?

DIRECTIVE PROMPT

A **directive prompt** is one which explicitly instructs the user how to respond:

Say *yes* if you accept the call; otherwise, say *no*.

In Sec. 24.2.1 we defined a **restrictive** grammar as a language model which strongly constrains the ASR system, only recognizing proper responses to a given prompt.

	Prompt Type	
Grammar	**Open**	**Directive**
Restrictive	*Doesn't make sense*	System Initiative
Non-Restrictive	User Initiative	Mixed Initiative

Figure 24.10 Operational definition of initiative, following Singh et al. (2002).

In Fig. 24.10 we then give the definition of initiative used in form-based dialogue systems, following Singh et al. (2002) and others. Here a system initiative interaction uses a directive prompt and a restrictive grammar; the user is told how to respond, and the ASR system is constrained to only recognize the responses that are prompted for. In user initiative, the user is given an open prompt, and the grammar must recognize any kind of response, since the user could say anything. Finally, in a mixed initiative interaction, the system gives the user a directive prompt with particular suggestions for response, but the non-restrictive grammar allows the user to respond outside the scope of the prompt.

Defining initiative as a property of the prompt and grammar type in this way allows systems to dynamically change their initiative type for different users and interactions. Novice users, or users with high speech recognition error, might be better served by more system initiative. Expert users, or those who happen to speak more recognizably, might do well with mixed or user initiative interactions. We will see in Sec. 24.6 how machine learning techniques can be used to choose initiative.

24.2.5 Dialogue Manager Error Handling: Confirmation/Rejection

In a dialogue system, mishearings are a particularly important class of problems, because speech recognition has such a high error rate. It is therefore important for dialogue systems to make sure that they have achieved the correct interpretation of the user's input. This is generally done by two methods: **confirming** understandings with the user, and **rejecting** utterances that the system is likely to have misunderstood.

EXPLICIT CONFIRMATION

Various strategies can be employed for confirmation with the user. A system using the **explicit confirmation** strategy asks the user a direct question to confirm their understanding. Here are two examples of explicit confirmations from travel planning systems. The (boldface) confirmation questions are both yes-no questions, one using a single sentence, the other presenting a declarative sentence followed by a tag question (a short question phrase like “right?” or “isn’t it?”):

S: Which city do you want to leave from?
U: Baltimore.
S: **Do you want to leave from Baltimore?**
U: Yes.

U: I’d like to fly from Denver Colorado to New York City on September twenty first in the morning on United Airlines
S: **Let’s see then. I have you going from Denver Colorado to New York on September twenty first. Is that correct?**
U: Yes

IMPLICIT CONFIRMATION

A system using the **implicit confirmation** strategy, rather than asking a direct question, uses the *demonstration* or *display* grounding strategies described above, confirming to the user by repeating back what the system understood the user to have said:

U: I want to travel to Berlin
S: **When do you want to travel to Berlin?**

U2: Hi I’d like to fly to Seattle Tuesday Morning
A3: **Traveling to Seattle on Tuesday, August eleventh in the morning.** Your full name?

Explicit and implicit confirmation have complementary strengths. Explicit confirmation makes it easier for users to correct the system’s misrecognitions since the user can just answer ‘no’ to the confirmation question. But explicit confirmation is awkward and increases the length of the conversation (Danieli and Gerbino, 1995; Walker et al., 1998). The explicit confirmation dialogue fragments above sound non-natural and definitely non-human; implicit confirmation is much more conversationally natural.

REJECTION

Confirmation is just one kind of conversational action that a system has to express lack of understanding. Another option is **rejection**. An ASR system rejects an utterance by giving the user a prompt like *I’m sorry, I didn’t understand that*.

PROGRESSIVE PROMPTING

Sometimes utterances are rejected multiple times. This might mean that the user is using language that the system is unable to follow. Thus when an utterance is rejected, systems often follow a strategy of **progressive prompting** or **escalating detail** (Yankelovich et al., 1995; Weinschenk and Barker, 2000) as in this example from Cohen et al. (2004):

System: When would you like to leave?
Caller: Well, um, I need to be in New York in time for the first World Series game.
System: <reject>. Sorry, I didn’t get that. Please say the month and day you’d like to leave.
Caller: I wanna go on October fifteenth.

In this example, instead of just repeating ‘When would you like to leave?’, the rejection prompt gives the caller more guidance about how to formulate an utterance the system will understand. These *you-can-say* help messages are important in helping improve systems understanding performance (Bohus and Rudnicky, 2005). If the caller’s utterance gets rejected yet again, the prompt can reflect this (‘I *still* didn’t get that’), and give the caller even more guidance.

RAPID REPROMPTING

An alternative strategy for error handling is **rapid reprompting**, in which the system rejects an utterance just by saying “I’m sorry?” or “What was that?”. Only if the caller’s utterance is rejected a second time does the system start applying progressive prompting. Cohen et al. (2004) summarizes experiments showing that users greatly prefer rapid reprompting as a first-level error prompt.

24.3 VoiceXML

VOICEXML

VXML

VoiceXML is the Voice Extensible Markup Language, an XML-based dialogue design language released by the W3C, and the most commonly used of the various speech markup languages (such as SALT). The goal of VoiceXML (or **vxml**) is to create simple audio dialogues of the type we have been describing, making use of ASR and TTS, and dealing with very simple mixed-initiative in a frame-based architecture. While VoiceXML is more common in the commercial rather than academic setting, it is a good way for the student to get a hands-on grasp of dialogue system design issues.

```
<form>
  <field name="transporttype">
     <prompt>
        Please choose airline, hotel, or rental car.
     </prompt>
     <grammar type="application/x=nuance-gsl">
       [airline hotel "rental car"]
     </grammar>
  </field>
  <block>
    <prompt>
     You have chosen <value expr="transporttype">.
    </prompt>
  </block>
</form>
```

Figure 24.11 A minimal VoiceXML script for a form with a single field. User is prompted, and the response is then repeated back.

A VoiceXML document contains a set of dialogues, each of which can be a *form* or a *menu*. We will limit ourselves to introducing forms; see `http://www.voicexml.org/` for more information on VoiceXML in general. The VoiceXML document in Fig. 24.11 defines a form with a single field named ‘transporttype’. The field has an attached prompt, *Please choose airline, hotel, or rental car*, which can be passed to the TTS system. It also has a grammar (language model) which is passed to the speech recognition engine to specify which words the recognizer is allowed to recognize. In the example in Fig. 24.11, the grammar consists of a disjunction of the three words *airline*, *hotel*, and *rental car*.

A `<form>` generally consists of a sequence of `<field>`s, together with a few other commands. Each field has a name (the name of the field in Fig. 24.11 is `transporttype`) which is also the name of the variable where the user’s response will be stored. The prompt associated with the field is specified via the `<prompt>` command. The grammar associated with the field is specified via the `<grammar>` command. VoiceXML supports various ways of specifying a grammar, including XML Speech Grammar, ABNF, and commercial standards, like Nuance GSL. We will be using the Nuance GSL format in the following examples.

The VoiceXML interpreter walks through a form in document order, repeatedly selecting each item in the form. If there are multiple fields, the interpreter will visit each one in order. The interpretation order can be changed in various ways, as we will see later. The example in Fig. 24.12 shows a form with three fields, for specifying the origin, destination, and flight date of an airline flight.

The prologue of the example shows two global defaults for error handling. If the user doesn't answer after a prompt (i.e., silence exceeds a timeout threshold), the VoiceXML interpreter will play the `<noinput>` prompt. If the user says something, but it doesn't match the grammar for that field, the VoiceXML interpreter will play the `<nomatch>` prompt. After any failure of this type, it is normal to re-ask the user the question that failed to get a response. Since these routines can be called from any field, and hence the exact prompt will be different every time, VoiceXML provides a `<reprompt\>` command, which will repeat the prompt for whatever field caused the error.

The three fields of this form show another feature of VoiceXML, the `<filled>` tag. The `<filled>` tag for a field is executed by the interpreter as soon as the field has been filled by the user. Here, this feature is used to give the user a confirmation of their input.

The last field, `departdate`, shows another feature of VoiceXML, the `type` attribute. VoiceXML 2.0 specifies seven built-in grammar types, `boolean`, `currency`, `date`, `digits`, `number`, `phone`, and `time`. Since the type of this field is `date`, a data-specific language model (grammar) will be automatically passed to the speech recognizer, so we don't need to specify the grammar here explicitly.

Fig. 24.13 gives a final example which shows mixed initiative. In a mixed initiative dialogue, users can choose not to answer the question that was asked by the system. For example, they might answer a different question, or use a long sentence to fill in multiple slots at once. This means that the VoiceXML interpreter can no longer just evaluate each field of the form in order; it needs to skip fields whose values are set. This is done by a *guard condition*, a test that keeps a field from being visited. The default guard condition for a field tests to see if the field's form item variable has a value, and if so the field is not interpreted.

Fig. 24.13 also shows a much more complex use of a grammar. This grammar is a CFG grammar with two rewrite rules, named `Flight` and `City`. The Nuance GSL grammar formalism uses parentheses () to mean concatenation and square brackets [] to mean disjunction. Thus a rule like (24.19) means that `Wantsentence` can be expanded as `i want to fly` or `i want to go`, and `Airports` can be expanded as `san francisco` or `denver`.

(24.19)
```
Wantsentence (i want to [fly go])
Airports [(san francisco) denver]
```

Grammar rules can refer to other grammar rules recursively, and so in the grammar

```
<noinput>
I’m sorry, I didn’t hear you.  <reprompt/>
</noinput>

<nomatch>
I’m sorry, I didn’t understand that.  <reprompt/>
</nomatch>

<form>
  <block>    Welcome to the air travel consultant.  </block>
  <field name="origin">
      <prompt>     Which city do you want to leave from?  </prompt>
      <grammar type="application/x=nuance-gsl">
        [(san francisco) denver (new york) barcelona]
      </grammar>
      <filled>
         <prompt>  OK, from <value expr="origin">  </prompt>
      </filled>
  </field>
  <field name="destination">
      <prompt>  And which city do you want to go to?   </prompt>
      <grammar type="application/x=nuance-gsl">
        [(san francisco) denver (new york) barcelona]
      </grammar>
      <filled>
         <prompt>   OK, to <value expr="destination">   </prompt>
      </filled>
  </field>
  <field name="departdate" type="date">
      <prompt>  And what date do you want to leave?  </prompt>
      <filled>
         <prompt>   OK, on <value expr="departdate">   </prompt>
      </filled>
  </field>
  <block>
      <prompt> OK, I have you are departing from  <value expr="origin">
              to <value expr="destination"> on <value expr="departdate">
      </prompt>
      send the info to book a flight...
  </block>
</form>
```

Figure 24.12 A VoiceXML script for a form with 3 fields, which confirms each field and handles the `noinput` and `nomatch` situations.

in Fig. 24.13 we see the grammar for `Flight` referring to the rule for `City`.

VoiceXML grammars take the form of CFG grammars with optional semantic attachments. The semantic attachments are generally either a text string (such as `"denver, colorado"`) or a slot and a filler. We can see an example of the former in the semantic attachments for the `City` rule (the `return` statements at the end of each line), which pass up the city and state name. The semantic attachments for the `Flight` rule shows the latter case, where the slot (`<origin>` or `<destination>` or both) is filled with the value passed up in the variable *x* from the `City` rule.

Because Fig. 24.13 is a mixed initiative grammar, the grammar has to be applicable to any of the fields. This is done by making the expansion for `Flight` a disjunction; note that it allows the user to specify only the origin city, only the destination city, or both.

```
<noinput>    I’m sorry, I didn’t hear you.  <reprompt/>  </noinput>

<nomatch> I’m sorry, I didn’t understand that.  <reprompt/> </nomatch>

<form>
   <grammar type="application/x=nuance-gsl">
    <![ CDATA[
    Flight (  ?[
             (i [wanna (want to)] [fly go])
             (i’d like to [fly go])
             ([(i wanna)(i’d like a)] flight)
          ]
          [
            ( [from leaving departing] City:x) {<origin $x>}
            ( [(?going to)(arriving in)] City:x) {<destination $x>}
            ( [from leaving departing] City:x
              [(?going to)(arriving in)] City:y) {<origin $x> <destination $y>}
          ]
          ?please
     )
     City [ [(san francisco) (s f o)] {return( "san francisco, california")}
            [(denver) (d e n)] {return( "denver, colorado")}
            [(seattle) (s t x)] {return( "seattle, washington")}
     ]
     ]]> </grammar>

  <initial name="init">
     <prompt> Welcome to the air travel consultant. What are your travel plans?  </prompt>
  </initial>

  <field name="origin">
     <prompt> Which city do you want to leave from?  </prompt>
     <filled>
        <prompt> OK, from <value expr="origin"> </prompt>
     </filled>
  </field>
  <field name="destination">
     <prompt> And which city do you want to go to?  </prompt>
     <filled>
        <prompt> OK, to <value expr="destination"> </prompt>
     </filled>
  </field>
  <block>
     <prompt> OK, I have you are departing from  <value expr="origin">
              to  <value expr="destination">.  </prompt>
    send the info to book a flight...
  </block>
</form>
```

Figure 24.13 A mixed initiative VoiceXML dialogue. The grammar allows sentences which specify the origin or destination cities or both. User can respond to the initial prompt by specifying origin city, destination city, or both.

24.4 DIALOGUE SYSTEM DESIGN AND EVALUATION

24.4.1 Designing Dialogue Systems

How does a dialogue system developer choose dialogue strategies, architectures, prompts, error messages, and so on? This process is often called **VUI** (**Voice User Interface**) design. The **user-centered design** principles of Gould and Lewis (1985) are:

VUI

1. Study the user and task: Understand the potential users and the nature of the task, via interviews with users and investigation of similar systems, and study of related human-human dialogues.

WIZARD-OF-OZ

2. Build simulations and prototypes: In **Wizard-of-Oz systems** (WOZ) or PNAMBIC (Pay No Attention to the Man BehInd the Curtain) systems, the users interact with what they think is a software system, but is in fact a human operator ("wizard") behind some disguising interface software (e.g. Gould et al., 1983; Good et al., 1984; Fraser and Gilbert, 1991). The name comes from the children's book *The Wizard of Oz* (Baum, 1900), in which the Wizard turned out to be just a simulation controlled by a man behind a curtain. A WOZ system can be used to test out an architecture before implementation; only the interface software and databases need to be in place. The wizard's linguistic output can be be disguised by a text-to-speech system, or via text-only interactions. It is difficult for the wizard to exactly simulate the errors, limitations, or time constraints of a real system; results of WOZ studies are thus somewhat idealized, but still can provide a useful first idea of the domain issues.

BARGE-IN

DIRECTIVE PROMPTS

3. Iteratively test the design on users: An iterative design cycle with embedded user testing is essential in system design (Nielsen, 1992; Cole et al., 1994, 1997; Yankelovich et al., 1995; Landauer, 1995). For example Stifelman et al. (1993) built a system that originally required the user to press a key to interrupt the system. They found in user testing that users instead tried to interrupt the system (**barge-in**), suggesting a redesign of the system to recognize overlapped speech. The iterative method is also very important for designing prompts which cause the user to respond in understandable or normative ways: Kamm (1994) and Cole et al. (1993) found that **directive prompts** ("Say *yes* if you accept the call, otherwise, say *no*") or the use of constrained forms (Oviatt et al., 1993) produced better results than open prompts like "Will you accept the call?". Simulations can also be used at this stage; user simulations that interact with a dialogue system can help test the interface for brittleness or errors (Chung, 2004).

See Cohen et al. (2004), Harris (2005) for more details on conversational interface design.

24.4.2 Dialogue System Evaluation

As the previous section suggested, user testing and evaluation is crucial in dialogue system design. Computing a *user satisfaction rating* can be done by having users interact with a dialogue system to perform a task, and then having them complete a questionnaire (Shriberg et al., 1992; Polifroni et al., 1992; Stifelman et al., 1993; Yankelovich et al., 1995; Möller, 2002). For example Fig. 24.14 shows multiple-choice questions adapted from Walker et al. (2001); responses are mapped into the range of 1 to 5, and then averaged over all questions to get a total user satisfaction rating.

It is often economically infeasible to run complete user satisfaction studies after every change in a system. For this reason it is often useful to have performance evaluation heuristics which correlate well with human satisfaction. A number of such factors and heuristics have been studied. One method that has been used to classify these factors is based on the idea that an optimal dialogue system is one which allows a user to accomplish their goals (maximizing task success) with the least problems (minimizing costs). Then we can study metrics which correlate with these two criteria.

TTS Performance	Was the system easy to understand ?
ASR Performance	Did the system understand what you said?
Task Ease	Was it easy to find the message/flight/train you wanted?
Interaction Pace	Was the pace of interaction with the system appropriate?
User Expertise	Did you know what you could say at each point?
System Response	How often was the system sluggish and slow to reply to you?
Expected Behavior	Did the system work the way you expected it to?
Future Use	Do you think you'd use the system in the future?

Figure 24.14 User satisfaction survey, adapted from Walker et al. (2001).

Task Completion Success: Task success can be measured by evaluating the correctness of the total solution. For a frame-based architecture, this might be the percentage of slots that were filled with the correct values, or the percentage of subtasks that were completed (Polifroni et al., 1992). Since different dialogue systems may be applied to different tasks, it is hard to compare them on this metric, so Walker et al. (1997) suggested using the Kappa coefficient, κ, to compute a completion score which is normalized for chance agreement and better enables cross-system comparison.

Efficiency Cost: Efficiency costs are measures of the system's efficiency at helping users. This can be measured via the total elapsed time for the dialogue in seconds, the number of total turns or of system turns, or the total number of queries (Polifroni et al., 1992). Other metrics include the number of system non-responses, and the "turn correction ratio": the number of system or user turns that were used solely to correct errors, divided by the total number of turns (Danieli and Gerbino, 1995; Hirschman and Pao, 1993).

BARGE-IN

CONCEPT ACCURACY

Quality Cost: Quality cost measures other aspects of the interaction that affect users' perception of the system. One such measure is the number of times the ASR system failed to return any sentence, or the number of ASR rejection prompts. Similar metrics include the number of times the user had to **barge-in** (interrupt the system), or the number of time-out prompts played when the user didn't respond quickly enough. Other quality metrics focus on how well the system understood and responded to the user. This can include the inappropriateness (verbose or ambiguous) of the system's questions, answers, and error messages (Zue et al., 1989), or the correctness of each question, answer, or error message (Zue et al., 1989; Polifroni et al., 1992). A very important quality cost is **concept accuracy** or **concept error rate**, which measures the percentage of semantic concepts that the NLU component returns correctly. For frame-based architectures this can be measured by counting the percentage of slots that are filled with the correct meaning. For example if the sentence 'I want to arrive in Austin at 5:00' is misrecognized to have the semantics "DEST-CITY: Boston, Time: 5:00" the concept accuracy would be 50% (one of two slots are wrong).

How should these success and cost metrics be combined and weighted? One approach is the PARADISE algorithm (PARAdigm for DIalogue System Evaluation), which applies multiple regression to this problem. The algorithm first assigns each dialogue a user satisfaction rating using questionnaires like the one in Fig. 24.14. A set of cost and success factors like those above is then treated as a set of independent

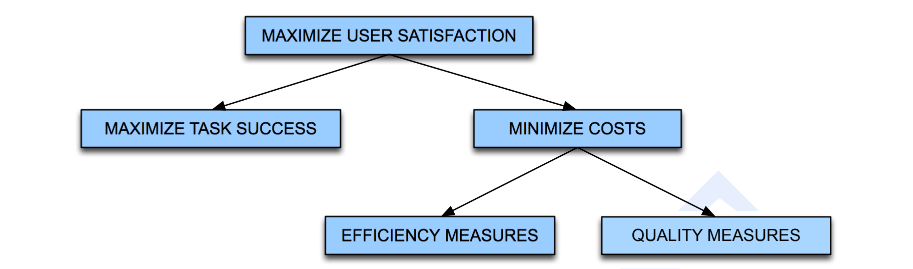


Figure 24.15 PARADISE's structure of objectives for spoken dialogue performance. After Walker et al. (2001).

factors; multiple regression is used to train a weight for each factor, measuring its importance in accounting for user satisfaction. Fig. 24.15 shows the particular model of performance that the PARADISE experiments have assumed. Each box is related to a set of factors that we summarized on the previous page. The resulting metric can be used to compare quite different dialogue strategies; evaluations using methods like PARADISE have suggested that task completion and concept accuracy are may be the most important predictors of user satisfaction; see Walker et al. (1997) and Walker et al. (2001, 2002).

A wide variety of other evaluation metrics and taxonomies have been proposed for describing the quality of spoken dialogue systems (Fraser, 1992; Möller, 2002, 2004, inter alia).

24.5 Information-state & Dialogue Acts

The basic frame-based dialogue systems we have introduced so far are only capable of limited domain-specific conversations. This is because the semantic interpretation and generation processes in frame-based dialogue systems are based only on what is needed to fill slots. In order to be be usable for more than just form-filling applications, a conversational agent needs to be able to do things like decide when the user has asked a question, made a proposal, or rejected a suggestion, and needs to be able to ground a users utterance, ask clarification questions, and suggest plans. This suggests that a conversational agent needs sophisticated models of interpretation and generation in terms of speech acts and grounding, and a more sophisticated representation of the dialogue context than just a list of slots.

In this section we sketch a more advanced architecture for dialogue management which allows for these more sophisticated components. This model is generally called the **information-state** architecture (Traum and Larsson, 2003, 2000), although we will use the term loosely to include architectures such as Allen et al. (2001). A probabilistic architecture which can be seen as an extension of the information-state approach, the **Markov decision process** model, will be described in the next section. The term **information-state architecture** is really a cover term for a number of quite different efforts toward more sophisticated agents; we'll assume here a structure consisting of 5

INFORMATION-STATE

components:

- the information state (the ‘discourse context’ or ‘mental model’)
- a dialogue act interpreter (or “interpretation engine”)
- a dialogue act generator (or “generation engine”)
- a set of update rules, which update the information state as dialogue acts are interpreted, and which include rules to generate dialogue acts.
- a control structure to select which update rules to apply

The term **information state** is intended to be very abstract, and might include things like the discourse context and the common ground of the two speakers, the beliefs or intentions of the speakers, user models, and so on. Crucially, information state is intended to be a more complex notion than the static states in a finite-state dialogue manager; the current state includes the values of many variables, the discourse context, and other elements that are not easily modeled by a state-number in a finite network.

Dialogue acts are an extension of speech acts which integrate ideas from grounding theory, and will be defined more fully fully in the next subsection. The interpretation engine takes speech as input and figures out sentential semantics and an appropriate dialogue act. The dialogue act generator takes dialogue acts and sentential semantics as input and produces text/speech as output.

Finally, the update rules modify the information state with the information from the dialogue acts. These update rules are a generalization of the production rules used in frame-based dialogue systems described above (Seneff and Polifroni, 2000, *inter alia*). A subset of update rules, called **selection rules**, are used to generate dialogue acts. For example, an update rule might say that when the interpretation engine recognizes an assertion, that the information state should be updated with the information in the assertion, and an obligation to perform a grounding act needs to be added to the information state. When a question is recognized, an update rule might specify the need to answer the question. We can refer to the combination of the update rules and control structure as the *Behavioral Agent* (Allen et al., 2001), as suggested in Fig. 24.16.

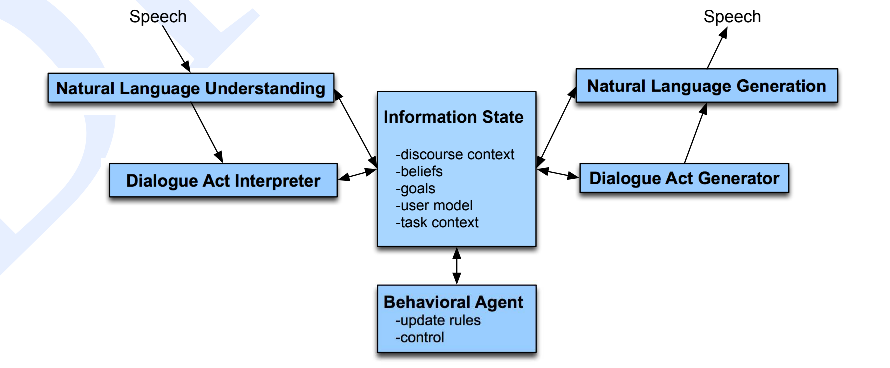


Figure 24.16 A version of the information-state approach to dialogue architecture.

While the intuition of the information-state model is quite simple, the details can be quite complex. The information state might involve rich discourse models such as Discourse Representation Theory or sophisticated models of the user's belief, desire, and intention (which we will return to in Sec. 24.7). Instead of describing a particular implementation here, we will focus in the next few sections on the dialogue act interpretation and generation engines, and a probabilistic information-state architecture via Markov decision processes.

24.5.1 Dialogue Acts

As we implied above, the speech acts as originally defined by Austin don't model key features of conversation such as grounding, contributions, adjacency pairs and so on. In order to capture these conversational phenomena, we use an extension of speech acts called **dialogue acts** (Bunt, 1994) (or **dialogue moves** or **conversational moves** (Power, 1979; Carletta et al., 1997b). A dialogue act extends speech acts with internal structure related specifically to these other conversational functions (Allen and Core, 1997; Bunt, 2000).

DIALOGUE ACT

MOVES

A wide variety of dialogue act tagsets have been proposed. Fig. 24.17 shows a very domain-specific tagset for the Verbmobil two-party scheduling domain, in which speakers were asked to plan a meeting at some future date. Notice that it has many very domain-specific tags, such as SUGGEST, used for when someone proposes a particular date to meet, and ACCEPT and REJECT, used to accept or reject a proposal for a date. Thus it has elements both from the presentation and acceptance phases of the Clark contributions discussed on page 7.

Tag	Example
THANK	*Thanks*
GREET	*Hello Dan*
INTRODUCE	*It's me again*
BYE	*Allright bye*
REQUEST-COMMENT	*How does that look?*
SUGGEST	*from thirteenth through seventeenth June*
REJECT	*No Friday I'm booked all day*
ACCEPT	*Saturday sounds fine,*
REQUEST-SUGGEST	*What is a good day of the week for you?*
INIT	*I wanted to make an appointment with you*
GIVE_REASON	*Because I have meetings all afternoon*
FEEDBACK	*Okay*
DELIBERATE	*Let me check my calendar here*
CONFIRM	*Okay, that would be wonderful*
CLARIFY	*Okay, do you mean Tuesday the 23rd?*
DIGRESS	*[we could meet for lunch] and eat lots of ice cream*
MOTIVATE	*We should go to visit our subsidiary in Munich*
GARBAGE	*Oops, I-*

Figure 24.17 The 18 high-level dialogue acts used in Verbmobil-1, abstracted over a total of 43 more specific dialogue acts. Examples are from Jekat et al. (1995).

There are a number of more general and domain-independent dialogue act tagsets. In the DAMSL (Dialogue Act Markup in Several Layers) architecture inspired by the

Act type	Sample Acts
turn-taking	take-turn, keep-turn, release-turn, assign-turn
grounding	acknowledge, repair, continue
core speech acts	inform, wh-question, accept, request, offer
argumentation	elaborate, summarize, question-answer, clarify

Figure 24.18 Conversation act types, from Traum and Hinkelman (1992).

work of Clark and Schaefer (1989) and Allwood et al. (1992), Allwood (1995) each utterance is tagged for two functions, **forward looking functions** like speech act functions, and the **backward looking** function, like grounding and answering, which 'look back' to the interlocutor's previous utterance (Allen and Core, 1997; Walker et al., 1996; Carletta et al., 1997a; Core et al., 1999).

Traum and Hinkelman (1992) proposed that the core speech acts and grounding acts that constitute dialogue acts could fit into an even richer hierarchy of **conversation acts**. Fig. 24.18 shows the four levels of act types they propose, with the two middle levels corresponding to DAMSL dialogue acts (grounding and core speech acts). The two new levels include turn-taking acts and *argumentation* relation, a conversational version of the coherence relations of Ch. 21.

CONVERSATION ACTS

The acts form a hierarchy, in that performance of an act at a higher level (for example a core speech act) entails performance of a lower level act (taking a turn). We will see the use of conversational acts in generation later on in this section, and will return to the question of coherence and dialogue structure in Sec. 24.7.

24.5.2 Interpreting Dialogue Acts

How can we do dialogue act interpretation, deciding whether a given input is a QUESTION, a STATEMENT, a SUGGEST (directive), or an ACKNOWLEDGEMENT? Perhaps we can just rely on surface syntax? We saw in Ch. 12 that yes-no-questions in English have **aux-inversion** (the auxiliary verb precedes the subject) statements have declarative syntax (no aux-inversion), and commands have no syntactic subject:

(24.20) YES-NO-QUESTION Will breakfast be served on USAir 1557?
STATEMENT I don't care about lunch
COMMAND Show me flights from Milwaukee to Orlando.

Alas, as is clear from Abbott and Costello's famous *Who's on First* routine at the beginning of the chapter, the mapping from surface form to illocutionary act is complex. For example, the following ATIS utterance looks like a YES-NO-QUESTION meaning something like *Are you capable of giving me a list of. . . ?*:

(24.21) Can you give me a list of the flights from Atlanta to Boston?

In fact, however, this person was not interested in whether the system was *capable* of giving a list; this utterance was a polite form of a REQUEST, meaning something more like *Please give me a list of*. . . . Thus what looks on the surface like a QUESTION can really be a REQUEST.

Similarly, what looks on the surface like a STATEMENT can really be a QUESTION. The very common CHECK question (Carletta et al., 1997b; Labov and Fanshel, 1977), is used to ask an interlocutor to confirm something that she has privileged knowledge about. CHECKS have declarative surface form:

A	OPEN-OPTION	I was wanting to make some arrangements for a trip that I'm going to be taking uh to LA uh beginning of the week after next.
B	HOLD	OK uh let me pull up your profile and I'll be right with you here. [pause]
B	CHECK	**And you said you wanted to travel next week?**
A	ACCEPT	Uh yes.

INDIRECT SPEECH ACTS

Utterances that use a surface statement to ask a question, or a surface question to issue a request, are called **indirect speech acts**.

MICROGRAMMAR

In order to resolve these dialogue act ambiguities we can model dialogue act interpretation as a supervised classification task, with dialogue act labels as hidden classes to be detected. We train classifiers on a corpus in which each utterance is hand-labeled for dialogue acts. The features used for dialogue act interpretation derive from the conversational context and from the act's **microgrammar** (Goodwin, 1996) (its characteristic lexical, grammatical, and prosodic properties):

FINAL LOWERING

1. **Words and Collocations:** *Please* or *would you* is a good cue for a REQUEST, *are you* for YES-NO-QUESTIONS, detected via **dialogue-specific *N*-gram** grammars.
2. **Prosody:** Rising pitch is a good cue for a YES-NO-QUESTION, while declarative utterances (like STATEMENTS) have **final lowering**: a drop in F0 at the end of the utterance. Loudness or stress can help distinguish the *yeah* that is an AGREEMENT from the *yeah* that is a BACKCHANNEL. We can extract acoustic correlates of prosodic features like F0, duration, and energy.
3. **Conversational Structure:** A *yeah* following a proposal is probably an AGREEMENT; a *yeah* after an INFORM is likely a BACKCHANNEL. Drawing on the idea of adjacency pairs (Schegloff, 1968; Sacks et al., 1974), we can model conversational structure as a bigram of dialogue acts

Formally our goal is to find the dialogue act d^* that has the highest posterior probability $P(d|o)$ given the observation of a sentence,

$$\begin{aligned} d^* &= \operatorname*{argmax}_d P(d|o) \\ &= \operatorname*{argmax}_d \frac{P(d)P(o|d)}{P(o)} \\ &= \operatorname*{argmax}_d P(d)P(o|d) \end{aligned} \tag{24.22}$$

Making some simplifying assumptions (that the prosody of the sentence f and the word sequence W are independent, and that the prior of a dialogue act can be modeled by the conditional given the previous dialogue act) we can estimate the observation likelihood for a dialogue act d as in (24.23):

$$P(o|d) = P(f|d)P(W|d) \tag{24.23}$$

$$d^* = \operatorname*{argmax}_d P(d|d_{t-1})P(f|d)P(W|d) \tag{24.24}$$

where

$$P(W|d) \;=\; \prod_{i=2}^{N} P(w_i|w_{i-1}...w_{i-N+1},d) \qquad (24.25)$$

Training the prosodic predictor to compute $P(f|d)$ has often been done with a decision tree. Shriberg et al. (1998), for example, built a CART tree to distinguish the four dialogue acts STATEMENT (S), YES-NO QUESTION (QY), DECLARATIVE-QUESTION like CHECK (QD) and WH-QUESTION (QW) based on acoustic features as the slope of F0 at the end of the utterance, the average energy at different places in the utterance, and various normalized duration measures. Fig. 24.19 shows the decision tree which gives the posterior probability $P(d|f)$ of a dialogue act d type given a set of acoustic features f. Note that the difference between S and QY toward the right of the tree is based on the feature `norm_f0_diff` (normalized difference between mean F0 of end and penultimate regions), while the difference between QW and QD at the bottom left is based on `utt_grad`, which measures F0 slope across the whole utterance.

Since decision trees produce a posterior probability $P(d|f)$, and equation (24.24) requires a likelihood $P(f|d)$, we need to massage the output of the decision tree by Bayesian inversion (dividing by the prior $P(d_i)$ to turn it into a likelihood); we saw this same process with the use of SVMs and MLPs instead of Gaussian classifiers in speech recognition in Sec. **??**. After all our simplifying assumptions the resulting equation for choosing a dialogue act tag would be:

$$\begin{aligned} d^* &= \underset{d}{\operatorname{argmax}} P(d)P(f|d)P(W|d) \\ &= \underset{d}{\operatorname{argmax}} P(d|d_{t-1}) \frac{P(d|f)}{P(d)} \prod_{i=2}^{N} P(w_i|w_{i-1}...w_{i-N+1},d) \end{aligned} \qquad (24.26)$$

24.5.3 Detecting Correction Acts

In addition to general-purpose dialogue act interpretation, we may want to build special-purpose detectors for particular acts. Let's consider one such detector, for the recognition of user **correction** of system errors. If a dialogue system misrecognizes an utterance (usually as a result of ASR errors) the user will generally correct the error by repeating themselves, or rephrasing the utterance. Dialogue systems need to recognize that users are doing a correction, and then figure out what the user is trying to correct, perhaps by interacting with the user further.

CORRECTION

Unfortunately, corrections are actually *harder* to recognize than normal sentences. Swerts et al. (2000) found that corrections in the TOOT dialogue system were misrecognized about twice as often (in terms of WER) as non-corrections. One reason for this is that speakers use a very different prosodic style called **hyperarticulation** for corrections. In hyperarticulated speech, some part of the utterance has exaggerated energy, duration, or F0 contours, such as *I said BAL-TI-MORE, not Boston* (Wade et al., 1992; Oviatt et al., 1998; Levow, 1998; Hirschberg et al., 2001).

HYPERARTICULATION

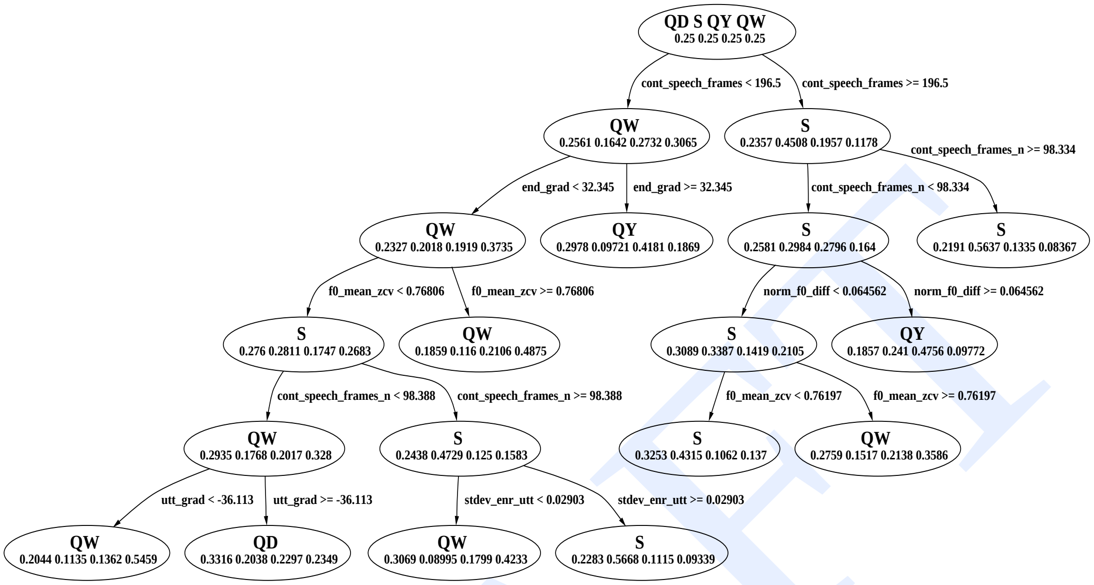


Figure 24.19 Decision tree for classifying DECLARATIVE QUESTIONS (QD), STATEMENT (S), YES-NO QUESTIONS (QY), and WH-QUESTIONS (QW), after Shriberg et al. (1998). Each node in the tree shows four probabilities, one for each of the four dialogue acts in the order QD, S, QY, QW; the most likely of the four is shown as the label for the node. Note that questions are shorter than statements (cont_speech_frames), that QY's rise at the end (end_grad), and that QD's rise throughout the utterance (utt_grad).

Speakers use various strategies to make corrections; Swerts et al. (2000) categorize these as repetition (REP), paraphrase (PAR), adding content (ADD), and omitting content (OMIT), as shown in Fig. 24.20. They found that most corrections were repetitions and omissions rather than paraphrases.

Various machine learning classifiers have been applied to the problem of detecting correction acts. Useful features include lexical information (words like "no", "correction", "I don't", swear words), prosodic and hyperarticulation features (increases in F0 range, pause duration, and word duration, generally normalized by the values for previous sentences), features indicating utterance length, ASR features (confidence, language model probability), and various dialogue features (Levow, 1998; Hirschberg et al., 2001; Bulyko et al., 2005).

In addition to correction detection, a conversational agent also needs appropriate control or update rules in the dialogue manager (Bulyko et al., 2005).

24.5.4 Generating Dialogue Acts: Confirmation and Rejection

Deciding which dialogue acts to generate has received much less attention than dialogue act interpretation. Stent (2002) is one recent model of dialogue act generation in the TRIPS system (Allen et al., 2001), based on Conversation Acts (page 32) and the BDI model to be described in Sec. 24.7. Stent uses a set of update rules for content

	Turn	Corr Type
S	On which day of the week do you want to leave?	
S	Sorry, I didn’t hear anything from you. Please tell me your departure date	
U	*Um tomorrow*	NA
S	You can say the departure date. For example, you can say ’today’ or ’tomorrow’. Or you can say ’Monday’, ’Tuesday’, ’Wednesday’...	
U	*Tomorrow*	REP
S	You can say the departure date. For example, you can say ’today’ or ’tomorrow’. Or you can say ’Monday’, ’Tuesday’, ’Wednesday’...	
U	*Saturday*	PAR
S	Do you want to leave on Saturday?	
U	Yes	
S	What time do you want to leave?	
U	Eight o’clock A M	
S	Do you want to leave around ten oclock p-m	
U	*Eight o’clock*	OMIT
S	Do you want to leave around ten oclock p-m	
U	*No*	OMIT
S	What time do you want to leave?	
U	*Eight o’clock A M*	ADD

Figure 24.20 Dialogue fragment from the TOOT system labeled for correction type, from Swerts et al. (2000).

Cue	**Turn-taking acts signaled**
um	KEEP-TURN, TAKE-TURN, RELEASE-TURN
<lipsmack>, <click>, so, uh	KEEP-TURN, TAKE-TURN
you know, isn’t that so	ASSIGN-TURN

Figure 24.21 Language used to perform turn-taking acts, from Stent (2002).

planning. One such rule says that if a user has just released the turn, the system can perform a TAKE-TURN act. Another rule says that if the system has a problem-solving need to summarize some information for the user, then it should use the ASSERT conversation act with that information as the semantic content. The content is then mapped into words using the standard techniques of natural language generation systems (see e.g., Reiter and Dale (2000)) After an utterance is generated, the information state (discourse context) is updated with its words, syntactic structure, semantic form, and semantic and conversation act structure. We will sketch in Sec. 24.7 some of the issues in modeling and planning that make generation a tough ongoing research effort.

Stent showed that a crucial issue in dialogue generation that doesn’t occur in monologue text generation is turn-taking acts. Fig. 24.21 shows some example of the turn-taking function of various linguistic forms, from her labeling of conversation acts in the Monroe corpus.

A focus of much work on dialogue act generation is the task of of generating the **confirmation** and **rejection** acts discussed in Sec. 24.2.5. Because this task is often solved by probabilistic methods, we’ll begin this discussion here, but continue it in the following section.

For example, while early dialogue systems tended to fix the choice of **explicit** versus **implicit** confirmation, recent systems treat the question of how to confirm more like a dialogue act generation task, in which the confirmation strategy is adaptive, changing from sentence to sentence.

Various factors can be included in the information-state and then used as features to a classifier in making this decision:

ASR confidence: The **confidence** that the ASR system assigns to an utterance can be used by explicitly confirming low-confidence sentences (Bouwman et al., 1999; San-Segundo et al., 2001; Litman et al., 1999; Litman and Pan, 2002). Recall that we briefly defined confidence on page **??** as a metric that the speech recognizer can give to a higher-level process (like dialogue) to indicate how confident the recognizer is that the word string that it returns is a good one. Confidence is often computed from the acoustic log-likelihood of the utterance (greater probability means higher confidence), but prosodic features can also be used in confidence prediction. For example utterances preceded by longer pauses, or with large F0 excursions, or longer durations are likely to be misrecognized (Litman et al., 2000).

Error cost: Confirmation is more important if an error would be costly. Thus explicit confirmation is common before actually booking a flight or moving money in an account (Kamm, 1994; Cohen et al., 2004).

A system can also choose to **reject** an utterance when the ASR confidence is so low, or the best interpretation is so semantically ill-formed, that the system can be relatively sure that the user's input was not recognized at all. Systems thus might have a three-tiered level of confidence; below a certain confidence threshold, an utterance is rejected. Above the threshold, it is explicitly confirmed. If the confidence is even higher, the utterance is implicitly confirmed.

CLARIFICATION SUBDIALOGUES

Instead of rejecting or confirming entire utterances, it would be nice to be able to clarify only the parts of the utterance that the system didn't understand. If a system can assign confidence at a more fine-grained level than the utterance, it can clarify such individual elements via **clarification subdialogues**.

Much of the recent work on generating dialogue acts has been within the Markov Decision Process framework, which we therefore turn to next.

24.6 Markov Decision Process Architecture

One of the fundamental insights of the information-state approach to dialogue architecture is that the choice of conversational actions is dynamically dependent on the current information state. The previous section discussed how dialogue systems could change confirmation and rejection strategies based on context. For example if the ASR or NLU confidence is low, we might choose to do explicit confirmation. If confidence is high, we might chose implicit confirmation, or even decide not to confirm at all. Using a dynamic strategy lets us choose the action which maximizes dialogue success, while minimizing costs. This idea of changing the actions of a dialogue system based on

MARKOV DECISION PROCESS

optimizing some kinds of rewards or costs is the fundamental intuition behind modeling dialogue as a **Markov decision process**. This model extends the information-state model by adding a probabilistic way of deciding on the proper actions given the current state.

MDP

A Markov decision process or **MDP** is characterized by a set of **states** S an agent can be in, a set of **actions** A the agent can take, and a **reward** r(a,s) that the agent receives for taking an action in a state. Given these factors, we can compute a **policy** π which specifies which action a the agent should take when in a given state s, so as to receive the best reward. To understand each of these components, we'll need to look at a tutorial example in which the state space is extremely reduced. Thus we'll return to the simple frame-and-slot world, looking at a pedagogical MDP implementation taken from Levin et al. (2000). Their tutorial example is a "Day-and-Month" dialogue system, whose goal is to get correct values of day and month for a two-slot frame via the shortest possible interaction with the user.

In principle, a state of an MDP could include any possible information about the dialogue, such as the complete dialogue history so far. Using such a rich model of state would make the number of possible states extraordinarily large. So a model of state is usually chosen which encodes a much more limited set of information, such as the values of the slots in the current frame, the most recent question asked to the user, the users most recent answer, the ASR confidence, and so on. For the Day-and-Month example let's represent the state of the system as the values of the two slots *day* and *month*. If we assume a special initial state s_i and final state s_f, there are a total of 411 states (366 states with a day and month (counting leap year), 12 states with a month but no day (d=0, m= 1,2,...12), and 31 states with a day but no month (m=0, d=1,2,...31)).

Actions of a MDP dialogue system might include generating particular speech acts, or performing a database query to find out information. For the Day-and-Month example, Levin et al. (2000) propose the following actions:

- a_d: a question asking for the day
- a_m: a question asking for the month
- $a_d m$: a question asking for both the day and the month
- a_f: a final action submitting the form and terminating the dialogue

Since the goal of the system is to get the correct answer with the shortest interaction, one possible reward function for the system would integrate three terms:

$$R = -(w_i n_i + w_e n_e + w_f n_f) \tag{24.27}$$

The term n_i is the number of interactions with the user, n_e is the number of errors, n_f is the number of slots which are filled (0, 1, or 2), and the *w*s are weights.

Finally, a dialogue policy π specifies which actions to apply in which state. Consider two possible policies: (1) asking for day and month separately, and (2) asking for them together. These might generate the two dialogues shown in Fig. 24.22.

In policy 1, the action specified for the no-date/no-month state is to ask for a day, while the action specified for any of the 31 states where we have a day but not a month is to ask for a month. In policy 2, the action specified for the no-date/no-month state is to ask an open-ended question (*Which date*) to get both a day and a month. The two policies have different advantages; an open prompt can leads to shorter dialogues but

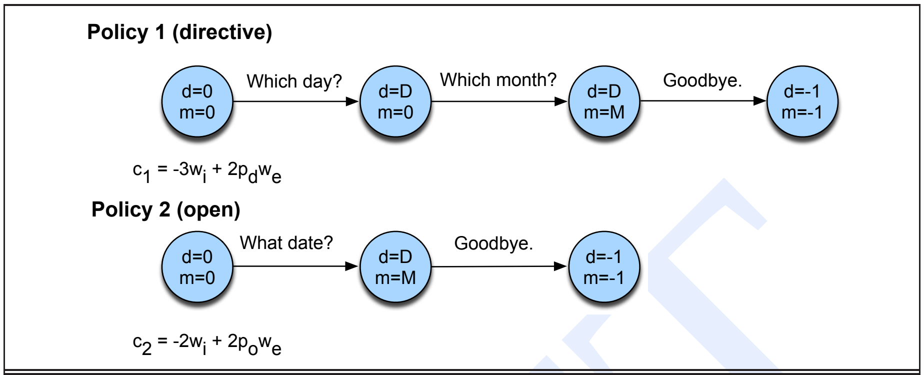


Figure 24.22 Two policies for getting a month and a day. After Levin et al. (2000).

is likely to cause more errors, while a directive prompt is slower but less error-prone. Thus the optimal policy depends on the values of the weights w, and also on the error rates of the ASR component. Let's call p_d the probability of the recognizer making an error interpreting a month or a day value after a directive prompt. The (presumably higher) probability of error interpreting a month or day value after an open prompt we'll call p_o. The reward for the first dialogue in Fig. 24.22 is thus $-3 \times w_i + 2 \times p_d \times w_e$. The reward for the second dialogue in Fig. 24.22 is $-2 \times w_i + 2 \times p_o \times w_e$. The directive prompt policy, policy 1, is thus better than policy 2 when the improved error rate justifies the longer interaction, i.e., when $p_d - p_o > \frac{w_i}{2w_e}$.

In the example we've seen so far, there were only two possible actions, and hence only a tiny number of possible policies. In general, the number of possible actions, states, and policies is quite large, and so the problem of finding the optimal policy π^* is much harder.

Markov decision theory together with classical reinforcement learning gives us a way to think about this problem. First, generalizing from Fig. 24.22, we can think of any particular dialogue as a trajectory in state space:

(24.28) $$s_1 \rightarrow_{a1,r1} s_2 \rightarrow_{a2,r2} s_3 \rightarrow_{a3,r3} \cdots$$

The best policy π^* is the one with the greatest expected reward over all trajectories. What is the expected reward for a given state sequence? The most common way to assign utilities or rewards to sequences is to use **discounted rewards**. Here we compute the expected cumulative reward Q of a sequence as a discounted sum of the utilities of the individual states:

DISCOUNTED REWARDS

(24.29) $$Q([s_0, a_0, s_1, a_1, s_2, a_2 \cdots]) = R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots,$$

The discount factor γ is a number between 0 and 1. This makes the agent care more about current rewards than future rewards; the more future a reward, the more discounted its value.

Given this model, it is possible to show that the expected cumulative reward $Q(s,a)$ for taking a particular action from a particular state is the following recursive equation called the **Bellman equation**:

BELLMAN EQUATION

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a') \tag{24.30}$$

What the Bellman equation says is that the expected cumulative reward for a given state/action pair is the immediate reward for the current state plus the expected discounted utility of all possible next states s', weighted by the probability of moving to that state s', and assuming once there we take the optimal action a'.

Equation (24.30) makes use of two parameters. We need a model of $P(s'|s,a)$, i.e. how likely a given state/action pair (s,a) is to lead to a new state s'. And we also need a good estimate of $R(s,a)$. If we had lots of labeled training data, we could simply compute both of these from labeled counts. For example, with labeled dialogues, we could simply count how many times we were in a given state s, and out of that how many times we took action a to get to state s', to estimate $P(s'|s,a)$. Similarly, if we had a hand-labeled reward for each dialogue, we could build a model of $R(s,a)$.

VALUE ITERATION

Given these parameters, it turns out that there is an iterative algorithm for solving the Bellman equation and determining proper Q values, the **value iteration** algorithm (Sutton and Barto, 1998; Bellman, 1957). We won't present this here, but see Chapter 17 of Russell and Norvig (2002) for the details of the algorithm as well as further information on Markov Decision Processes.

How do we get enough labeled training data to set these parameters? This is especially worrisome in any real problem, where the number of states s is extremely large. Two methods have been applied in the past. The first is to carefully hand-tune the states and policies so that there are a very small number of states and policies that need to be set automatically. In this case we can build a dialogue system which explore the state space by generating random conversations. Probabilities can then be set from this corpus of conversations. The second is to build a simulated user. The user interacts with the system millions of times, and the system learns the state transition and reward probabilities from this corpus.

The first approach, using real users to set parameters in a small state space, was taken by Singh et al. (2002). They used reinforcement learning to make a small set of optimal policy decisions. Their NJFun system learned to choose actions which varied the initiative (system, user, or mixed) and the confirmation strategy (explicit or none). The state of the system was specified by values of 7 features including which slot in the frame is being worked on (1-4), the ASR confidence value (0-5), how many times a current slot question had been asked, whether a restrictive or non-restrictive grammar was used, and so on. The result of using only 7 features with a small number of attributes resulted in a small state space (62 states). Each state had only 2 possible actions (system versus user initiative when asking questions, explicit versus no confirmation when receiving answers). They ran the system with real users, creating 311 conversations. Each conversation had a very simple binary reward function; 1 if the user completed the task (finding specified museums, theater, winetasting in the New Jersey area), 0 if the user did not. The system successful learned a good dialogue policy (roughly, start with user initiative, then back of to either mixed or system initiative when reasking for an attribute; confirm only at lower confidence values; both initiative and confirmation policies, however, are different for different attributes). They showed that their policy actually was more successful based on various objective measures than many hand-designed policies reported in the literature.

The simulated user strategy was taken by Levin et al. (2000), in their MDP model with reinforcement learning in the ATIS task. Their simulated user was a generative stochastic model that given the system's current state and actions, produces a frame-slot representation of a user response. The parameters of the simulated user were estimated from a corpus of ATIS dialogues. The simulated user was then used to interact with the system for tens of thousands of conversations, leading to an optimal dialogue policy.

While the MDP architecture offers a powerful new way of modeling dialogue behavior, it relies on the problematic assumption that the system actually knows what state it is in. This is of course not true in a number of ways; the system never knows the true internal state of the user, and even the state in the dialogue may be obscured by speech recognition errors. Recent attempts to relax this assumption have relied on Partially Observable Markov Decision Processes, or POMDPs (sometimes pronounced 'pom-deepeez'). In a POMDP, we model the user output as an observed signal generated from yet another hidden variable. There are also problems with MDPs and POMDPs related to computational complexity and simulations which aren't reflective of true user behavior; See the end notes for references.

24.7 Advanced: Plan-based Dialogue Agents

One of the earliest models of conversational agent behavior, and also one of the most sophisticated, is based on the use of AI planning techniques. For example, the Rochester TRIPS agent (Allen et al., 2001) simulates helping with emergency management, planning where and how to supply ambulances or personnel in a simulated emergency situation. The same planning algorithms that reason how to get an ambulance from point A to point B can be applied to conversation as well. Since communication and conversation are just special cases of rational action in the world, these actions can be planned like any other. So an agent seeking to find out some information can come up with the plan of asking the interlocutor for the information. An agent hearing an utterance can interpret a speech act by running the planner 'in reverse', using inference rules to infer what plan the interlocutor might have had to cause them to say what they said.

BDI

Using plans to generate and interpret sentences in this way require that the planner have good models of its **beliefs**, **desires**, and **intentions** (BDI), as well as those of the interlocutor. Plan-based models of dialogue are thus often referred to as **BDI** models. BDI models of dialogue were first introduced by Allen, Cohen, Perrault, and their colleagues and students in a number of influential papers showing how speech acts could be generated (Cohen and Perrault, 1979), and interpreted (Perrault and Allen, 1980; Allen and Perrault, 1980). At the same time, Wilensky (1983) introduced plan-based models of understanding as part of the task of interpreting stories. In another related line of research, Grosz and her colleagues and students showed how using similar notions of intention and plans allowed ideas of discourse structure and coherence to be applied to dialogue.

24.7.1 Plan-Inferential Interpretation and Production

Let's first sketch out the ideas of plan-based comprehension and production. How might a plan-based agent act as the human travel agent to understand sentence C_2 in the dialogue repeated below?

C_1: I need to travel in May.

A_1: And, what day in May did you want to travel?

C_2: OK uh I need to be there for a meeting that's from the 12th to the 15th.

The Gricean principle of Relevance can be used to infer that the client's meeting is relevant to the flight booking. The system may know that one precondition for having a meeting (at least before web conferencing) is being at the place where the meeting is in. One way of being at a place is flying there, and booking a flight is a precondition for flying there. The system can follow this chain of inference, abducing that user wants to fly on a date before the 12th.

Next, consider how our plan-based agent could act as the human travel agent to produce sentence A_1 in the dialogue above. The planning agent would reason that in order to help a client book a flight it must know enough information about the flight to book it. It reasons that knowing the month (May) is insufficient information to specify a departure or return date. The simplest way to find out the needed date information is to ask the client.

In the rest of this section, we'll flesh out the sketchy outlines of planning for understanding and generation using Perrault and Allen's formal definitions of belief and desire in the predicate calculus. Reasoning about belief is done with a number of axiom schemas inspired by Hintikka (1969). We'll represent "*S* believes the proposition *P*" as the two-place predicate $B(S,P)$, with axiom schemas such as $B(A,P) \wedge B(A,Q) \Rightarrow B(A,P \wedge Q)$. Knowledge is defined as "true belief"; *S knows that P* will be represented as $KNOW(S,P)$, defined as $\text{KNOW}(S,P) \equiv P \wedge B(S,P)$.

The theory of desire relies on the predicate WANT. If an agent *S* wants *P* to be true, we say $WANT(S,P)$, or $W(S,P)$ for short. *P* can be a state or the execution of some action. Thus if ACT is the name of an action, $W(S,\text{ACT}(H))$ means that *S* wants *H* to do ACT. The logic of WANT relies on its own set of axiom schemas just like the logic of belief.

ACTION SCHEMA

The BDI models also require an axiomatization of actions and planning; the simplest of these is based on a set of **action schema**s based on the simple AI planning model STRIPS (Fikes and Nilsson, 1971). Each action schema has a set of parameters with *constraints* about the type of each variable, and three parts:

- *Preconditions:* Conditions that must already be true to perform the action.
- *Effects:* Conditions that become true as a result of performing the action.
- *Body:* A set of partially ordered goal states that must be achieved in performing the action.

In the travel domain, for example, the action of agent *A* booking flight *F*1 for client *C* might have the following simplified definition:

BOOK-FLIGHT(A,C,F):
Constraints: Agent(A) $\wedge$ Flight(F) $\wedge$ Client(C)
Precondition: Know(A,depart-date(F)) $\wedge$ Know(A,depart-time(F)) $\wedge$ Know(A,origin(F)) $\wedge$ Know(A,flight-type(F)) $\wedge$ Know(A,destination(F)) $\wedge$ Has-Seats(F) $\wedge$ W(C,(BOOK(A,C,F))) $\wedge$ . . .
Effect: Flight-Booked(A,C,F)
Body: Make-Reservation(A,F,C)

This same kind of STRIPS action specification can be used for speech acts. INFORM is the speech act of informing the hearer of some proposition, based on Grice's (1957) idea that a speaker informs the hearer of something merely by causing the hearer to believe that the speaker wants them to know something:

INFORM(S,H,P):
Constraints: Speaker(S) $\wedge$ Hearer(H) $\wedge$ Proposition(P)
Precondition: Know(S,P) $\wedge$ W(S, INFORM(S, H, P))
Effect: Know(H,P)
Body: B(H,W(S,Know(H,P)))

REQUEST is the directive speech act for requesting the hearer to perform some action:

REQUEST(S,H,ACT):
Constraints: Speaker(S) $\wedge$ Hearer(H) $\wedge$ ACT(A) $\wedge$ H is agent of ACT
Precondition: W(S,ACT(H))
Effect: W(H,ACT(H))
Body: B(H,W(S,ACT(H)))

Let's now see how a plan-based dialogue system might interpret the sentence:

C_2: I need to be there for a meeting that's from the 12th to the 15th.

We'll assume the system has the BOOK-FLIGHT plan mentioned above. In addition, we'll need knowledge about meetings and getting to them, in the form of the MEETING, FLY-TO, and TAKE-FLIGHT plans, sketched broadly below:

MEETING(P,L,T1,T2):
Constraints: Person(P) $\wedge$ Location (L) $\wedge$ Time (T1) $\wedge$ Time (T2) $\wedge$ Time (TA)
Precondition: At (P, L, TA)
Before (TA, T1)
Body: ...

FLY-TO(P, L, T):
Constraints: Person(P) $\wedge$ Location (L) $\wedge$ Time (T)
Effect: At (P, L, T)
Body: TAKE-FLIGHT(P, L, T)

TAKE-FLIGHT(P, L, T):
Constraints: Person(P) ∧ Location (L) ∧ Time (T) ∧ Flight (F) ∧ Agent (A)
Precondition: BOOK-FLIGHT (A, P, F)
Destination-Time(F) = T
Destination-Location(F) = L
Body: ...

Now let's assume that an NLU module returns a semantics for the client's utterance which (among other things) includes the following semantic content:

MEETING (P, ?L, T1, T2)
Constraints: P = Client ∧ T1 = May 12 ∧ T2 = May 15

Our plan-based system now has two plans established, one MEETING plan from this utterance, and one BOOK-FLIGHT plan from the previous utterance. The system implicitly uses the Gricean Relevance intuition to try to connect them. Since BOOK-FLIGHT is a precondition for TAKE-FLIGHT, the system may hypothesize (infer) that the user is planning a TAKE-FLIGHT. Since TAKE-FLIGHT is in the body of FLY-TO, the system further infers a FLY-TO plan. Finally, since the effect of FLY-TO is a precondition of the MEETING, the system can unify each of the people, locations, and times of all of these plans. The result will be that the system knows that the client wants to arrive at the destination before May 12th.

Let's turn to the details of our second example:

C_1: I need to travel in May.
A_1: And, what day in May did you want to travel?

How does a plan-based agent know to ask question A_1? This knowledge comes from the BOOK-FLIGHT plan, whose preconditions were that the agent know a variety of flight parameters including the departure date and time, origin and destination cities, and so forth. Utterance C_1 contains the origin city and partial information about the departure date; the agent has to request the rest. A plan-based agent would use an action schema like REQUEST-INFO to represent a plan for asking information questions (simplified from Cohen and Perrault (1979)):

REQUEST-INFO(A,C,I):
Constraints: Agent(A) ∧ Client(C)
Precondition: Know(C,I)
Effect: Know(A,I)
Body: B(C,W(A,Know(A,I)))

Because the effects of REQUEST-INFO match each precondition of BOOK-FLIGHT, the agent can use REQUEST-INFO to achieve the missing information.

24.7.2 The Intentional Structure of Dialogue

In Sec. **??** we introduced the idea that the segments of a discourse are related by **coherence relations** like **Explanation** or **Elaboration** which describe the **informational** relation between discourse segments. The BDI approach to utterance interpretation gives rise to another view of coherence which is particularly relevant for dialogue, the **intentional** approach (Grosz and Sidner, 1986). According to this approach, what makes a dialogue coherent is its **intentional structure**, the plan-based intentions of the speaker underlying each utterance.

INTENTIONAL STRUCTURE

These intentions are instantiated in the model by assuming that each discourse has an underlying purpose held by the person who initiates it, called the **discourse purpose** (DP). Each discourse segment within the discourse has a corresponding purpose, a **discourse segment purpose** (DSP), which has a role in achieving the overall DP. Possible DPs/DSPs include intending that some agent intend to perform some physical task, or that some agent believe some fact.

DISCOURSE PURPOSE
DISCOURSE SEGMENT PURPOSE

As opposed to the larger sets of coherence relations used in informational accounts of coherence, Grosz and Sidner propose only two such relations: **dominance** and **satisfaction-precedence**. DSP$_1$ dominates DSP$_2$ if satisfying DSP$_2$ is intended to provide part of the satisfaction of DSP$_1$. DSP$_1$ satisfaction-precedes DSP$_2$ if DSP$_1$ must be satisfied before DSP$_2$.

C$_1$: I need to travel in May.
A$_1$: And, what day in May did you want to travel?
C$_2$: OK uh I need to be there for a meeting that's from the 12th to the 15th.
A$_2$: And you're flying into what city?
C$_3$: Seattle.
A$_3$: And what time would you like to leave Pittsburgh?
C$_4$: Uh hmm I don't think there's many options for non-stop.
A$_4$: Right. There's three non-stops today.
C$_5$: What are they?
A$_5$: The first one departs PGH at 10:00am arrives Seattle at 12:05 their time. The second flight departs PGH at 5:55pm, arrives Seattle at 8pm. And the last flight departs PGH at 8:15pm arrives Seattle at 10:28pm.
C$_6$: OK I'll take the 5ish flight on the night before on the 11th.
A$_6$: On the 11th? OK. Departing at 5:55pm arrives Seattle at 8pm, U.S. Air flight 115.
C$_7$: OK.

Figure 24.23 A fragment from a telephone conversation between a client (C) and a travel agent (A) (repeated from Fig. 24.4).

Consider the dialogue between a client (C) and a travel agent (A) that we saw earlier, repeated here in Fig. 24.23. Collaboratively, the caller and agent successfully identify a flight that suits the caller's needs. Achieving this joint goal requires that a top-level discourse intention be satisfied, listed as I1 below, in addition to several intermediate intentions that contributed to the satisfaction of I1, listed as I2-I5:

I1: (Intend C (Intend A (A find a flight for C)))

I2: (Intend A (Intend C (Tell C A departure date)))

I3: (Intend A (Intend C (Tell C A destination city)))

I4: (Intend A (Intend C (Tell C A departure time)))

I5: (Intend C (Intend A (A find a nonstop flight for C)))

Intentions I2–I5 are all subordinate to intention I1, as they were all adopted to meet preconditions for achieving intention I1. This is reflected in the dominance relationships below:

I1 dominates I2 $\wedge$ I1 dominates I3 $\wedge$ I1 dominates I4 $\wedge$ I1 dominates I5

Furthermore, intentions I2 and I3 needed to be satisfied before intention I5, since the agent needed to know the departure date and destination in order to start listing nonstop flights. This is reflected in the satisfaction-precedence relationships below:

I2 satisfaction-precedes I5 $\wedge$ I3 satisfaction-precedes I5

The dominance relations give rise to the discourse structure depicted in Figure 24.24. Each discourse segment is numbered in correspondence with the intention number that serves as its DP/DSP.

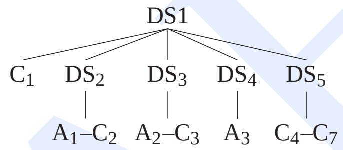


Figure 24.24 Discourse Structure of the Flight Reservation Dialogue

Intentions and their relationships give rise to a coherent discourse based on their role in the overall *plan* that the caller is inferred to have. We assume that the caller and agent have the plan BOOK-FLIGHT described on page 43. This plan requires that the agent know the departure time and date and so on. As we discussed above, the agent can use the REQUEST-INFO action scheme from page 44 to ask the user for this information.

SUBDIALOGUES

Subsidiary discourse segments are also called **subdialogues**; DS2 and DS3 in particular are **information-sharing** (Chu-Carroll and Carberry, 1998) **knowledge precondition** subdialogues (Lochbaum et al., 1990; Lochbaum, 1998), since they are initiated by the agent to help satisfy preconditions of a higher-level goal.

Algorithms for inferring intentional structure in dialogue work similarly to algorithms for inferring dialogue acts, either employing the BDI model (e.g., Litman, 1985; Grosz and Sidner, 1986; Litman and Allen, 1987; Carberry, 1990; Passonneau and Litman, 1993; Chu-Carroll and Carberry, 1998), or machine learning architectures based on cue phrases (Reichman, 1985; Grosz and Sidner, 1986; Hirschberg and Litman, 1993), prosody (Hirschberg and Pierrehumbert, 1986; Grosz and Hirschberg, 1992; Pierrehumbert and Hirschberg, 1990; Hirschberg and Nakatani, 1996), and other cues.

24.8 Summary

Conversational agents are a crucial speech and language processing application that are already widely used commercially. Research on these agents relies crucially on an understanding of human dialogue or conversational practices.

- Dialogue systems generally have 5 components: speech recognition, natural language understanding, dialogue management, natural language generation, and speech synthesis. They may also have a task manager specific to the task domain.
- Dialogue architectures for conversational agents include finite-state systems, **frame-based** production systems, and advanced systems such as information-state, Markov Decision Processes, and **BDI** (**belief-desire-intention**) models.
- Turn-taking, grounding, conversational structure, implicature, and initiative are crucial human dialogue phenomena that must also be dealt with in conversational agents.
- Speaking in dialogue is a kind of action; these acts are referred to as speech acts or **dialogue acts**. Models exist for generating and interpreting these acts.

Bibliographical and Historical Notes

Early work on speech and language processing had very little emphasis on the study of dialogue. The dialogue manager for the simulation of the paranoid agent PARRY (Colby et al., 1971), was a little more complex. Like ELIZA, it was based on a production system, but where ELIZA's rules were based only on the words in the user's previous sentence, PARRY's rules also rely on global variables indicating its emotional state. Furthermore, PARRY's output sometimes makes use of script-like sequences of statements when the conversation turns to its delusions. For example, if PARRY's **anger** variable is high, he will choose from a set of "hostile" outputs. If the input mentions his delusion topic, he will increase the value of his **fear** variable and then begin to express the sequence of statements related to his delusion.

The appearance of more sophisticated dialogue managers awaited the better understanding of human-human dialogue. Studies of the properties of human-human dialogue began to accumulate in the 1970's and 1980's. The Conversation Analysis community (Sacks et al., 1974; Jefferson, 1984; Schegloff, 1982) began to study the interactional properties of conversation. Grosz's (1977) dissertation significantly influenced the computational study of dialogue with its introduction of the study of dialogue structure, with its finding that "task-oriented dialogues have a structure that closely parallels the structure of the task being performed" (p. 27), which led to her work on intentional and attentional structure with Sidner. Lochbaum et al. (2000) is a good recent summary of the role of intentional structure in dialogue. The BDI model integrating earlier AI planning work (Fikes and Nilsson, 1971) with speech act theory (Austin, 1962; Gordon and Lakoff, 1971; Searle, 1975a) was first worked out by Cohen and Perrault (1979), showing how speech acts could be generated, and Perrault and Allen (1980) and Allen and Perrault (1980), applying the approach to speech-act interpretation. Simultaneous work on a plan-based model of understanding was developed by Wilensky (1983) in the Schankian tradition.

Probabilistic models of dialogue act interpretation were informed by linguistic work which focused on the discourse meaning of prosody (Sag and Liberman, 1975; Pierrehumbert, 1980), by Conversation Analysis work on microgrammar (e.g. Goodwin, 1996), by work such as Hinkelman and Allen (1989), who showed how lexical and phrasal cues could be integrated into the BDI model, and then worked out at a number of speech and dialogue labs in the 1990's (Waibel, 1988; Daly and Zue, 1992; Kompe et al., 1993; Nagata and Morimoto, 1994; Woszczyna and Waibel, 1994; Reithinger et al., 1996; Kita et al., 1996; Warnke et al., 1997; Chu-Carroll, 1998; Stolcke et al., 1998; Taylor et al., 1998; Stolcke et al., 2000).

Modern dialogue systems drew on research at many different labs in the 1980's and 1990's. Models of dialogue as collaborative behavior were introduced in the late 1980's and 1990's, including the ideas of common ground (Clark and Marshall, 1981), reference as a collaborative process (Clark and Wilkes-Gibbs, 1986), and models of **joint intentions** (Levesque et al., 1990), and **shared plans** (Grosz and Sidner, 1980). Related to this area is the study of **initiative** in dialogue, studying how the dialogue control shifts between participants (Walker and Whittaker, 1990; Smith and Gordon, 1997; Chu-Carroll and Brown, 1997).

A wide body of dialogue research came out of AT&T and Bell Laboratories around the turn of the century, including much of the early work on MDP dialogue systems as well as fundamental work on cue-phrases, prosody, and rejection and confirmation. Work on dialogue acts and dialogue moves drew from a number of sources, including HCRC's Map Task (Carletta et al., 1997b), and the work of James Allen and his colleagues and students, for example Hinkelman and Allen (1989), showing how lexical and phrasal cues could be integrated into the BDI model of speech acts, and Traum (2000), Traum and Hinkelman (1992), and from Sadek (1991).

Much recent academic work in dialogue focuses on multimodal applications (Johnston et al., 2007; Niekrasz and Purver, 2006, inter alia), on the information-state model (Traum and Larsson, 2003, 2000) or on reinforcement learning architectures including POMDPs (Roy et al., 2000; Young, 2002; Lemon et al., 2006; Williams and Young, 2005, 2000). Work in progress on MDPs and POMDPs focuses on computational complexity (they currently can only be run on quite small domains with limited numbers of slots), and on improving simulations to make them more reflective of true user behavior. Alternative algorithms include SMDPs (Cuayáhuitl et al., 2007). See Russell and Norvig (2002) and Sutton and Barto (1998) for a general introduction to reinforcement learning.

Recent years have seen the widespread commercial use of dialogue systems, often based on VoiceXML. Some more sophisticated systems have also seen deployment. For example **Clarissa**, the first spoken dialogue system used in space, is a speech-enabled procedure navigator that was used by astronauts on the International Space Station (Rayner and Hockey, 2004; Aist et al., 2002). Much research focuses on more mundane in-vehicle applications in cars Weng et al. (2006, *inter alia*). Among the important technical challenges in embedding these dialogue systems in real applications are good techniques for endpointing (deciding if the speaker is done talking) (Ferrer et al., 2003) and for noise robustness.

CLARISSA

Good surveys on dialogue systems include Harris (2005), Cohen et al. (2004), McTear (2002, 2004), Sadek and De Mori (1998), and the dialogue chapter in Allen

(1995).

Exercises

24.1 List the dialogue act misinterpretations in the *Who's On First* routine at the beginning of the chapter.

24.2 Write a finite-state automaton for a dialogue manager for checking your bank balance and withdrawing money at an automated teller machine.

24.3 Dispreferred responses (for example turning down a request) are usually signaled by surface cues, such as significant silence. Try to notice the next time you or someone else utters a dispreferred response, and write down the utterance. What are some other cues in the response that a system might use to detect a dispreferred response? Consider non-verbal cues like eye-gaze and body gestures.

24.4 When asked a question to which they aren't sure they know the answer, people display their lack of confidence via cues that resemble other dispreferred responses. Try to notice some unsure answers to questions. What are some of the cues? If you have trouble doing this, read Smith and Clark (1993) and listen specifically for the cues they mention.

24.5 Build a VoiceXML dialogue system for giving the current time around the world. The system should ask the user for a city and a time format (24 hour, etc) and should return the current time, properly dealing with time zones.

24.6 Implement a small air-travel help system based on text input. Your system should get constraints from the user about a particular flight that they want to take, expressed in natural language, and display possible flights on a screen. Make simplifying assumptions. You may build in a simple flight database or you may use a flight information system on the web as your backend.

24.7 Augment your previous system to work with speech input via VoiceXML. (or alternatively, describe the user interface changes you would have to make for it to work via speech over the phone). What were the major differences?

24.8 Design a simple dialogue system for checking your email over the telephone. Implement in VoiceXML.

24.9 Test your email-reading system on some potential users. Choose some of the metrics described in Sec. 24.4.2 and evaluate your system.

Aist, G., Dowding, J., Hockey, B. A., and Hieronymus, J. L. (2002). An intelligent procedure assistant for astronaut training and support. In *ACL-02*, Philadelphia, PA.

Allen, J. (1995). *Natural Language Understanding*. Benjamin Cummings, Menlo Park, CA.

Allen, J. and Core, M. (1997). Draft of DAMSL: Dialog act markup in several layers. Unpublished manuscript.

Allen, J., Ferguson, G., and Stent, A. (2001). An architecture for more realistic conversational systems. In *IUI '01: Proceedings of the 6th international conference on Intelligent user interfaces*, Santa Fe, New Mexico, United States, pp. 1–8. ACM Press.

Allen, J. and Perrault, C. R. (1980). Analyzing intention in utterances. *Artificial Intelligence*, *15*, 143–178.

Allwood, J. (1995). An activity-based approach to pragmatics. *Gothenburg Papers in Theoretical Linguistics*, *76*.

Allwood, J., Nivre, J., and Ahlsén, E. (1992). On the semantics and pragmatics of linguistic feedback. *Journal of Semantics*, *9*, 1–26.

Atkinson, M. and Drew, P. (1979). *Order in Court*. Macmillan, London.

Austin, J. L. (1962). *How to Do Things with Words*. Harvard University Press.

Baum, L. F. (1900). *The Wizard of Oz*. Available at Project Gutenberg.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., and Winograd, T. (1977). GUS, A frame driven dialog system. *Artificial Intelligence*, *8*, 155–173.

Bohus, D. and Rudnicky, A. I. (2005). Sorry, I Didn't Catch That! - An Investigation of Non-understanding Errors and Recovery Strategies. In *SIGdial-2005*, Lisbon, Portugal.

Bouwman, G., Sturm, J., and Boves, L. (1999). Incorporating confidence measures in the Dutch train timetable information system developed in the Arise project. In *IEEE ICASSP-99*, pp. 493–496.

Bulyko, I., Kirchhoff, K., Ostendorf, M., and Goldberg, J. (2005). Error-sensitive response generation in a spoken langugage dialogue system. *Speech Communication*, *45*(3), 271–288.

Bunt, H. (1994). Context and dialogue control. *Think*, *3*, 19–31.

Bunt, H. (2000). Dynamic interpretation and dialogue theory, volume 2. In Taylor, M. M., Neel, F., and Bouwhuis, D. G. (Eds.), *The structure of multimodal dialogue*, pp. 139–166. John Benjamins, Amsterdam.

Carberry, S. (1990). *Plan Recognition in Natural Language Dialog*. MIT Press.

Carletta, J., Dahlbäck, N., Reithinger, N., and Walker, M. A. (1997a). Standards for dialogue coding in natural language processing. Tech. rep. Report no. 167, Dagstuhl Seminars. Report from Dagstuhl seminar number 9706.

Carletta, J., Isard, A., Isard, S., Kowtko, J. C., Doherty-Sneddon, G., and Anderson, A. H. (1997b). The reliability of a dialogue structure coding scheme. *Computational Linguistics*, *23*(1), 13–32.

Chu-Carroll, J. (1998). A statistical model for discourse act recognition in dialogue interactions. In Chu-Carroll, J. and Green, N. (Eds.), *Applying Machine Learning to Discourse Processing. Papers from the 1998 AAAI Spring Symposium.* Tech. rep. SS-98-01, Menlo Park, CA, pp. 12–17. AAAI Press.

Chu-Carroll, J. and Brown, M. K. (1997). Tracking initiative in collaborative dialogue interactions. In *ACL/EACL-97*, Madrid, Spain, pp. 262–270.

Chu-Carroll, J. and Carberry, S. (1998). Collaborative response generation in planning dialogues. *Computational Linguistics*, *24*(3), 355–400.

Chu-Carroll, J. and Carpenter, B. (1999). Vector-based natural language call routing. *Computational Linguistics*, *25*(3), 361–388.

Chung, G. (2004). Developing a flexible spoken dialog system using simulation. In *ACL-04*, Barcelona, Spain.

Clark, H. H. (1994). Discourse in production. In Gernsbacher, M. A. (Ed.), *Handbook of Psycholinguistics*. Academic Press.

Clark, H. H. (1996). *Using Language*. Cambridge University Press.

Clark, H. H. and Marshall, C. (1981). Definite reference and mutual knowledge. In Joshi, A. K., Webber, B. L., and Sag, I. (Eds.), *Elements of discourse understanding*, pp. 10–63. Cambridge.

Clark, H. H. and Schaefer, E. F. (1989). Contributing to discourse. *Cognitive Science*, *13*, 259–294.

Clark, H. H. and Wilkes-Gibbs, D. (1986). Referring as a collaborative process. *Cognition*, *22*, 1–39.

Cohen, M. H., Giangola, J. P., and Balogh, J. (2004). *Voice User Interface Design*. Addison-Wesley, Boston.

Cohen, P. R. and Oviatt, S. L. (1994). The role of voice in human-machine communication. In Roe, D. B. and Wilpon, J. G. (Eds.), *Voice Communication Between Humans and Machines*, pp. 34–75. National Academy Press, Washington, D.C.

Cohen, P. R. and Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science*, *3*(3), 177–212.

Colby, K. M., Weber, S., and Hilf, F. D. (1971). Artificial paranoia. *Artificial Intelligence*, *2*(1), 1–25.

Cole, R. A., Novick, D. G., Vermeulen, P. J. E., Sutton, S., Fanty, M., Wessels, L. F. A., de Villiers, J. H., Schalkwyk, J., Hansen, B., and Burnett, D. (1997). Experiments with a spoken dialogue system for taking the US census. *Speech Communication*, *23*, 243–260.

Cole, R. A., Novick, D. G., Burnett, D., Hansen, B., Sutton, S., and Fanty, M. (1994). Towards automatic collection of the U.S. census. In *IEEE ICASSP-94*, Adelaide, Australia, Vol. I, pp. 93–96. IEEE.

Cole, R. A., Novick, D. G., Fanty, M., Sutton, S., Hansen, B., and Burnett, D. (1993). Rapid prototyping of spoken language systems: The Year 2000 Census Project. In *Proceedings of the International Symposium on Spoken Dialogue*, Waseda University, Tokyo, Japan.

Core, M., Ishizaki, M., Moore, J. D., Nakatani, C., Reithinger, N., Traum, D. R., and Tutiya, S. (1999). The report of the third workshop of the Discourse Resource Initiative, Chiba University and Kazusa Academia Hall. Tech. rep. No.3 CC-TR-99-1, Chiba Corpus Project, Chiba, Japan.

Cuayáhuitl, H., Renals, S., Lemon, O., and Shimodaira, H. (2007). Hierarchical dialogue optimization using semi-Markov decision processes. In *INTERSPEECH-07*.

Daly, N. A. and Zue, V. W. (1992). Statistical and linguistic analyses of F_0 in read and spontaneous speech. In *ICSLP-92*, Vol. 1, pp. 763–766.

Danieli, M. and Gerbino, E. (1995). Metrics for evaluating dialogue strategies in a spoken language system. In *Proceedings of the 1995 AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation*, Stanford, CA, pp. 34–39. AAAI Press, Menlo Park, CA.

Ferrer, L., Shriberg, E., and Stolcke, A. (2003). A prosody-based approach to end-of-utterance detection that does not require speech recognition. In *IEEE ICASSP-03*.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Fraser, N. (1992). Assessment of interactive systems. In Gibbon, D., Moore, R., and Winski, R. (Eds.), *Handbook on Standards and Resources for Spoken Language Systems*, pp. 564–615. Mouton de Gruyter, Berlin.

Fraser, N. M. and Gilbert, G. N. (1991). Simulating speech systems. *Computer Speech and Language*, *5*, 81–99.

Good, M. D., Whiteside, J. A., Wixon, D. R., and Jones, S. J. (1984). Building a user-derived interface. *Communications of the ACM*, *27*(10), 1032–1043.

Goodwin, C. (1996). Transparent vision. In Ochs, E., Schegloff, E. A., and Thompson, S. A. (Eds.), *Interaction and Grammar*, pp. 370–404. Cambridge University Press.

Gordon, D. and Lakoff, G. (1971). Conversational postulates. In *CLS-71*, pp. 200–213. University of Chicago. Reprinted in Peter Cole and Jerry L. Morgan (Eds.), *Speech Acts: Syntax and Semantics Volume 3*, Academic, 1975.

Gorin, A. L., Riccardi, G., and Wright, J. H. (1997). How may i help you?. *Speech Communication*, *23*, 113–127.

Gould, J. D., Conti, J., and Hovanyecz, T. (1983). Composing letters with a simulated listening typewriter. *Communications of the ACM*, *26*(4), 295–308.

Gould, J. D. and Lewis, C. (1985). Designing for usability: Key principles and what designers think. *Communications of the ACM*, *28*(3), 300–311.

Grice, H. P. (1957). Meaning. *Philosophical Review*, *67*, 377–388. Reprinted in *Semantics*, edited by Danny D. Steinberg & Leon A. Jakobovits (1971), Cambridge University Press, pages 53–59.

Grice, H. P. (1975). Logic and conversation. In Cole, P. and Morgan, J. L. (Eds.), *Speech Acts: Syntax and Semantics Volume 3*, pp. 41–58. Academic Press.

Grice, H. P. (1978). Further notes on logic and conversation. In Cole, P. (Ed.), *Pragmatics: Syntax and Semantics Volume 9*, pp. 113–127. Academic Press.

Grosz, B. J. and Hirschberg, J. (1992). Some intonational characteristics of discourse structure. In *ICSLP-92*, Vol. 1, pp. 429–432.

Grosz, B. J. (1977). *The Representation and Use of Focus in Dialogue Understanding*. Ph.D. thesis, University of California, Berkeley.

Grosz, B. J. and Sidner, C. L. (1980). Plans for discourse. In Cohen, P. R., Morgan, J., and Pollack, M. E. (Eds.), *Intentions in Communication*, pp. 417–444. MIT Press.

Grosz, B. J. and Sidner, C. L. (1986). Attention, intentions, and the structure of discourse. *Computational Linguistics*, *12*(3), 175–204.

Guindon, R. (1988). A multidisciplinary perspective on dialogue structure in user-advisor dialogues. In Guindon, R. (Ed.), *Cognitive Science And Its Applications For Human-Computer Interaction*, pp. 163–200. Lawrence Erlbaum.

Harris, R. A. (2005). *Voice Interaction Design: Crafting the New Conversational Speech Systems*. Morgan Kaufmann.

Hemphill, C. T., Godfrey, J., and Doddington, G. (1990). The ATIS spoken language systems pilot corpus. In *Proceedings DARPA Speech and Natural Language Workshop*, Hidden Valley, PA, pp. 96–101. Morgan Kaufmann.

Hinkelman, E. A. and Allen, J. (1989). Two constraints on speech act ambiguity. In *Proceedings of the 27th ACL*, Vancouver, Canada, pp. 212–219.

Hintikka, J. (1969). Semantics for propositional attitudes. In Davis, J. W., Hockney, D. J., and Wilson, W. K. (Eds.), *Philosophical Logic*, pp. 21–45. D. Reidel, Dordrecht, Holland.

Hirschberg, J. and Litman, D. J. (1993). Empirical studies on the disambiguation of cue phrases. *Computational Linguistics*, *19*(3), 501–530.

Hirschberg, J., Litman, D. J., and Swerts, M. (2001). Identifying user corrections automatically in spoken dialogue systems. In *NAACL*.

Hirschberg, J. and Nakatani, C. (1996). A prosodic analysis of discourse segments in direction-giving monologues. In *ACL-96*, Santa Cruz, CA, pp. 286–293.

Hirschberg, J. and Pierrehumbert, J. B. (1986). The intonational structuring of discourse. In *ACL-86*, New York, pp. 136–144.

Hirschman, L. and Pao, C. (1993). The cost of errors in a spoken language system. In *EUROSPEECH-93*, pp. 1419–1422.

Issar, S. and Ward, W. (1993). Cmu's robust spoken language understanding system. In *Eurospeech 93*, pp. 2147–2150.

Jefferson, G. (1984). Notes on a systematic deployment of the acknowledgement tokens 'yeah' and 'mm hm'. *Papers in Linguistics*, *17*(2), 197–216.

Jekat, S., Klein, A., Maier, E., Maleck, I., Mast, M., and Quantz, J. (1995). Dialogue Acts in VERBMOBIL verbmobil–report–65–95..

Johnston, M., Ehlen, P., Gibbon, D., and Liu, Z. (2007). The multimodal presentation dashboard. In *NAACL HLT 2007 Workshop 'Bridging the Gap'*.

Kamm, C. A. (1994). User interfaces for voice applications. In Roe, D. B. and Wilpon, J. G. (Eds.), *Voice Communication Between Humans and Machines*, pp. 422–442. National Academy Press, Washington, D.C.

Kita, K., Fukui, Y., Nagata, M., and Morimoto, T. (1996). Automatic acquisition of probabilistic dialogue models. In *ICSLP-96*, Philadelphia, PA, Vol. 1, pp. 196–199.

Kompe, R., Kießling, A., Kuhn, T., Mast, M., Niemann, H., Nöth, E., Ott, K., and Batliner, A. (1993). Prosody takes over: A prosodically guided dialog system. In *EUROSPEECH-93*, Berlin, Vol. 3, pp. 2003–2006.

Labov, W. and Fanshel, D. (1977). *Therapeutic Discourse*. Academic Press.

Landauer, T. K. (Ed.). (1995). *The Trouble With Computers: Usefulness, Usability, and Productivity*. MIT Press.

Lemon, O., Georgila, K., Henderson, J., and Stuttle, M. (2006). An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the TALK in-car system. In *EACL-06*.

Levesque, H. J., Cohen, P. R., and Nunes, J. H. T. (1990). On acting together. In *AAAI-90*, Boston, MA, pp. 94–99. Morgan Kaufmann.

Levin, E., Pieraccini, R., and Eckert, W. (2000). A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Transactions on Speech and Audio Processing*, *8*, 11–23.

Levinson, S. C. (1983). *Pragmatics*. Cambridge University Press.

Levow, G.-A. (1998). Characterizing and recognizing spoken corrections in human-computer dialogue. In *COLING-ACL*, pp. 736–742.

Lewin, I., Becket, R., Boye, J., Carter, D., Rayner, M., and Wirén, M. (1999). Language processing for spoken dialogue systems: is shallow parsing enough?. In *Accessing Information in Spoken Audio: Proceedings of ESCA ETRW Workshop, Cambridge, 19 & 20th April 1999*, pp. 37–42.

Litman, D. J. (1985). *Plan Recognition and Discourse Analysis: An Integrated Approach for Understanding Dialogues*. Ph.D. thesis, University of Rochester, Rochester, NY.

Litman, D. J. and Allen, J. (1987). A plan recognition model for subdialogues in conversation. *Cognitive Science*, *11*, 163–200.

Litman, D. J. and Pan, S. (2002). Designing and evaluating an adaptive spoken dialogue system. *User Modeling and User-Adapted Interaction*, *12*(2-3), 111–137.

Litman, D. J. and Silliman, S. (2004). Itspoke: An intelligent tutoring spoken dialogue system. In *HLT-NAACL-04*.

Litman, D. J., Swerts, M., and Hirschberg, J. (2000). Predicting automatic speech recognition performance using prosodic cues. In *NAACL 2000*.

Litman, D. J., Walker, M. A., and Kearns, M. S. (1999). Automatic detection of poor speech recognition at the dialogue level. In *ACL-99*, College Park, MA, pp. 309–316. ACL.

Lochbaum, K. E. (1998). A collaborative planning model of intentional structure. *Computational Linguistics*, *24*(4), 525–572.

Lochbaum, K. E., Grosz, B. J., and Sidner, C. L. (1990). Models of plans to support communication: An initial report. In *AAAI-90*, Boston, MA, pp. 485–490. Morgan Kaufmann.

Lochbaum, K. E., Grosz, B. J., and Sidner, C. L. (2000). Discourse structure and intention recognition. In Dale, R., Somers, H. L., and Moisl, H. (Eds.), *Handbook of Natural Language Processing*. Marcel Dekker.

McTear, M. F. (2002). Spoken dialogue technology: Enabling the conversational interface. *ACM Computing Surveys*, *34*(1), 90–169.

McTear, M. F. (2004). *Spoken Dialogue Technology*. Springer Verlag, London.

Miller, S., Bobrow, R. J., Ingria, R., and Schwartz, R. (1994). Hidden understanding models of natural language. In *Proceedings of the 32nd ACL*, Las Cruces, NM, pp. 25–32.

Miller, S., Fox, H., Ramshaw, L. A., and Weischedel, R. (2000). A novel use of statistical parsing to extract information from text. In *Proceedings of the 1st Annual Meeting of the North American Chapter of the ACL (NAACL)*, Seattle, Washington, pp. 226–233.

Miller, S., Stallard, D., Bobrow, R. J., and Schwartz, R. (1996). A fully statistical approach to natural language interfaces. In *ACL-96*, Santa Cruz, CA, pp. 55–61.

Möller, S. (2002). A new taxonomy for the quality of telephone services based on spoken dialogue systems. In *In Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue*, pp. 142–153.

Möller, S. (2004). *Quality of Telephone-Based Spoken Dialogue Systems*. Springer.

Nagata, M. and Morimoto, T. (1994). First steps toward statistical modeling of dialogue to predict the speech act type of the next utterance. *Speech Communication*, *15*, 193–203.

Niekrasz, J. and Purver, M. (2006). A multimodal discourse ontology for meeting understanding. In Renals, S. and Bengio, S. (Eds.), *Machine Learning for Multimodal Interaction: Second International Workshop MLMI 2005, Revised Selected Papers*, No. 3689 in Lecture Notes in Computer Science, pp. 162–173. Springer-Verlag.

Nielsen, J. (1992). The usability engineering life cycle. *IEEE Computer*, *25*(3), 12–22.

Norman, D. A. (1988). *The Design of Everyday Things*. Basic Books.

Oviatt, S. L., Cohen, P. R., Wang, M. Q., and Gaston, J. (1993). A simulation-based research strategy for designing complex

NL sysems. In *Proceedings DARPA Speech and Natural Language Workshop*, Princeton, NJ, pp. 370–375. Morgan Kaufmann.

Oviatt, S. L., MacEachern, M., and Levow, G.-A. (1998). Predicting hyperarticulate speech during human-computer error resolution. *Speech Communication*, *24*, 87–110.

Passonneau, R. and Litman, D. J. (1993). Intention-based segmentation: Human reliability and correlation with linguistic cues. In *Proceedings of the 31st ACL*, Columbus, Ohio, pp. 148–155.

Perrault, C. R. and Allen, J. (1980). A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics*, *6*(3-4), 167–182.

Pieraccini, R., Levin, E., and Lee, C.-H. (1991). Stochastic representation of conceptual structure in the ATIS task. In *Proceedings DARPA Speech and Natural Language Workshop*, Pacific Grove, CA, pp. 121–124. Morgan Kaufmann.

Pierrehumbert, J. B. and Hirschberg, J. (1990). The meaning of intonational contours in the interpretation of discourse. In Cohen, P. R., Morgan, J., and Pollack, M. (Eds.), *Intentions in Communication*, pp. 271–311. MIT Press.

Pierrehumbert, J. B. (1980). *The Phonology and Phonetics of English Intonation*. Ph.D. thesis, MIT.

Polifroni, J., Hirschman, L., Seneff, S., and Zue, V. W. (1992). Experiments in evaluating interactive spoken language systems. In *Proceedings DARPA Speech and Natural Language Workshop*, Harriman, NY, pp. 28–33. Morgan Kaufmann.

Power, R. (1979). The organization of purposeful dialogs. *Linguistics*, *17*, 105–152.

Rayner, M. and Hockey, B. A. (2003). Transparent combination of rule-based and data-driven approaches in a speech understanding architecture. In *EACL-03*, Budapest, Hungary.

Rayner, M. and Hockey, B. A. (2004). Side effect free dialogue management in a voice enabled procedure browser. In *ICSLP-04*, pp. 2833–2836.

Rayner, M., Hockey, B. A., and Bouillon, P. (2006). *Putting Linguistics into Speech Recognition*. CSLI.

Reichman, R. (1985). *Getting Computers to Talk Like You and Me*. MIT Press.

Reiter, E. and Dale, R. (2000). *Building Natural Language Generation Systems*. Cambridge University Press.

Reithinger, N., Engel, R., Kipp, M., and Klesen, M. (1996). Predicting dialogue acts for a speech-to-speech translation system. In *ICSLP-96*, Philadelphia, PA, Vol. 2, pp. 654–657.

Roy, N., Pineau, J., and Thrun, S. (2000). Spoken dialog management for robots. In *ACL-00*, Hong Kong.

Russell, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall. Second edition.

Sacks, H., Schegloff, E. A., and Jefferson, G. (1974). A simplest systematics for the organization of turn-taking for conversation. *Language*, *50*(4), 696–735.

Sadek, D. and De Mori, R. (1998). Dialogue systems. In De Mori, R. (Ed.), *Spoken Dialogues With Computers*. Academic Press, London.

Sadek, M. D. (1991). Dialogue acts are rational plans. In *ESCA/ETR Workshop on the Structure of Multimodal Dialogue*, pp. 19–48.

Sag, I. A. and Liberman, M. Y. (1975). The intonational disambiguation of indirect speech acts. In *CLS-75*, pp. 487–498. University of Chicago.

San-Segundo, R., Montero, J. M., Ferreiros, J., Còrdoba, R., and Pardo, J. M. (2001). Designing confirmation mechanisms and error recovery techniques in a railway information system for Spanish. In *In Proceedings of the 2nd SIGdial Workshop on Discourse and Dialogue, Aalborg, Denmark*.

Schegloff, E. A. (1968). Sequencing in conversational openings. *American Anthropologist*, *70*, 1075–1095.

Schegloff, E. A. (1979). Identification and recognition in telephone conversation openings. In Psathas, G. (Ed.), *Everyday Language: Studies in Ethnomethodology*, pp. 23–78. Irvington.

Schegloff, E. A. (1982). Discourse as an interactional achievement: Some uses of ‘uh huh’ and other things that come between sentences. In Tannen, D. (Ed.), *Analyzing Discourse: Text and Talk*, pp. 71–93. Georgetown University Press, Washington, D.C.

Searle, J. R. (1975a). Indirect speech acts. In Cole, P. and Morgan, J. L. (Eds.), *Speech Acts: Syntax and Semantics Volume 3*, pp. 59–82. Academic Press.

Searle, J. R. (1975b). A taxonomy of illocutionary acts. In Gunderson, K. (Ed.), *Language, Mind and Knowledge, Minnesota Studies in the Philosophy of Science*, Vol. VII, pp. 344–369. University of Minnesota Press, Amsterdam. Also appears in John R. Searle, *Expression and Meaning: Studies in the Theory of Speech Acts*, Cambridge University Press, 1979.

Seneff, S. (1995). TINA: A natural language system for spoken language application. *Computational Linguistics*, *18*(1), 62–86.

Seneff, S. (2002). Response planning and generation in the MERCURY flight reservation system. *Computer Speech and Language, Special Issue on Spoken Language Generation*, *16*(3-4), 283–312.

Seneff, S. and Polifroni, J. (2000). Dialogue management in the mercury flight reservation system. In *ANLP/NAACL Workshop on Conversational Systems*, Seattle.

Shriberg, E., Bates, R., Taylor, P., Stolcke, A., Jurafsky, D., Ries, K., Coccaro, N., Martin, R., Meteer, M., and Van Ess-Dykema, C. (1998). Can prosody aid the automatic classification of dialog acts in conversational speech?. *Language and Speech (Special Issue on Prosody and Conversation)*, *41*(3-4), 439–487.

Shriberg, E., Wade, E., and Price, P. (1992). Human-machine problem solving using spoken language systems (SLS): Factors affecting performance and user satisfaction. In *Proceedings DARPA Speech and Natural Language Workshop*, Harriman, NY, pp. 49–54. Morgan Kaufmann.

Singh, S. P., Litman, D. J., Kearns, M. J., and Walker, M. A. (2002). Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *J. Artif. Intell. Res. (JAIR)*, *16*, 105–133.

Smith, R. W. and Gordon, S. A. (1997). Effects of variable initiative on linguistic behavior in human-computer spoken natural language dialogue. *Computational Linguistics*, *23*(1), 141–168.

Smith, V. L. and Clark, H. H. (1993). On the course of answering questions. *Journal of Memory and Language*, *32*, 25–38.

Stalnaker, R. C. (1978). Assertion. In Cole, P. (Ed.), *Pragmatics: Syntax and Semantics Volume 9*, pp. 315–332. Academic Press.

Stent, A. (2002). A conversation acts model for generating spoken dialogue contributions. *Computer Speech and Language, Special Issue on Spoken Language Generation*, *16*(3-4).

Stifelman, L. J., Arons, B., Schmandt, C., and Hulteen, E. A. (1993). VoiceNotes: A speech interface for a hand-held voice notetaker. In *Human Factors in Computing Systems: INTERCHI '93 Conference Proceedings*, Amsterdam, pp. 179–186. ACM.

Stolcke, A., Ries, K., Coccaro, N., Shriberg, E., Bates, R., Jurafsky, D., Taylor, P., Martina, R., Meteer, M., and Van Ess-Dykema, C. (2000). Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics*, *26*, 339–371.

Stolcke, A., Shriberg, E., Bates, R., Coccaro, N., Jurafsky, D., Martin, R., Meteer, M., Ries, K., Taylor, P., and Van Ess-Dykema, C. (1998). Dialog act modeling for conversational speech. In Chu-Carroll, J. and Green, N. (Eds.), *Applying Machine Learning to Discourse Processing. Papers from the 1998 AAAI Spring Symposium. Tech. rep. SS-98-01*, Stanford, CA, pp. 98–105. AAAI Press.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Bradford Books (MIT Press).

Swerts, M., Litman, D. J., and Hirschberg, J. (2000). Corrections in spoken dialogue systems. In *ICSLP-00*, Beijing, China.

Taylor, P., King, S., Isard, S., and Wright, H. (1998). Intonation and dialog context as constraints for speech recognition. *Language and Speech*, *41*(3-4), 489–508.

Traum, D. R. (2000). 20 questions for dialogue act taxonomies. *Journal of Semantics*, *17*(1).

Traum, D. R. and Hinkelman, E. A. (1992). Conversation acts in task-oriented spoken dialogue. *Computational Intelligence: Special Issue on Computational Approaches to Non-Literal Language*, *8*(3).

Traum, D. R. and Larsson, S. (2000). Information state and dialogue management in the trindi dialogue move engine toolkit. *Natural Language Engineering*, *6*(323-340), 97–114.

Traum, D. R. and Larsson, S. (2003). The information state approach to dialogue management. In van Kuppevelt, J. and Smith, R. (Eds.), *Current and New Directions in Discourse and Dialogue*. Kluwer.

VanLehn, K., Jordan, P. W., Rosé, C., Bhembe, D., Böttner, M., Gaydos, A., Makatchev, M., Pappuswamy, U., Ringenberg, M., Roque, A., Siler, S., Srivastava, R., and Wilson, R. (2002). The architecture of Why2-Atlas: A coach for qualitative physics essay writing. In *Proc. Intelligent Tutoring Systems*.

Wade, E., Shriberg, E., and Price, P. J. (1992). User behaviors affecting speech recognition. In *ICSLP-92*, pp. 995–998.

Waibel, A. (1988). *Prosody and Speech Recognition*. Morgan Kaufmann.

Walker, M. A., Fromer, J. C., and Narayanan, S. S. (1998). Learning optimal dialogue strategies: a case study of a spoken dialogue agent for email. In *COLING/ACL-98*, Montreal, Canada, pp. 1345–1351.

Walker, M. A., Kamm, C. A., and Litman, D. J. (2001). Towards developing general models of usability with PARADISE. *Natural Language Engineering: Special Issue on Best Practice in Spoken Dialogue Systems*, *6*(3).

Walker, M. A., Litman, D. J., Kamm, C. A., and Abella, A. (1997). PARADISE: A framework for evaluating spoken dialogue agents. In *ACL/EACL-97*, Madrid, Spain, pp. 271–280.

Walker, M. A., Maier, E., Allen, J., Carletta, J., Condon, S., Flammia, G., Hirschberg, J., Isard, S., Ishizaki, M., Levin, L., Luperfoy, S., Traum, D. R., and Whittaker, S. (1996). Penn multiparty standard coding scheme: Draft annotation manual. `www.cis.upenn.edu/˜ircs/discourse-tagging/newcoding.html`.

Walker, M. A., Passonneau, R., Rudnicky, A. I., Aberdeen, J., Boland, J., Bratt, E., Garofolo, J., Hirschman, L., Le, A., Lee, S., Narayanan, S. S., Papineni, K., Pellom, B., Polifroni, J., Potamianos, A., Prabhu, P., Rudnicky, A. I., Sanders, G., Seneff, S., Stallard, D., and Whittaker, S. (2002). Cross-site evaluation in DARPA Communicator: The June 2000 data collection. submitted.

Walker, M. A. and Rambow, O. (2002). Spoken language generation. *Computer Speech and Language, Special Issue on Spoken Language Generation*, *16*(3-4), 273–281.

Walker, M. A. and Whittaker, S. (1990). Mixed initiative in dialogue: An investigation into discourse segmentation. In *Proceedings of the 28th ACL*, Pittsburgh, PA, pp. 70–78.

Walker, M. A. *et al*. (2001). Cross-site evaluation in darpa communicator: The june 2000 data collection. Submitted ms.

Ward, N. and Tsukahara, W. (2000). Prosodic features which cue back-channel feedback in English and Japanese. *Journal of Pragmatics*, *32*, 1177–1207.

Ward, W. and Issar, S. (1994). Recent improvements in the cmu spoken language understanding system. In *ARPA Human Language Technologies Workshop*, Plainsboro, N.J.

Warnke, V., Kompe, R., Niemann, H., and Nöth, E. (1997). Integrated dialog act segmentation and classification using prosodic features and language models. In *EUROSPEECH-97*, Vol. 1, pp. 207–210.

Weinschenk, S. and Barker, D. T. (2000). *Designing effective speech interfaces*. Wiley.

Weng, F., Varges, S., Raghunathan, B., Ratiu, F., Pon-Barry, H., Lathrop, B., Zhang, Q., Scheideck, T., Bratt, H., Xu, K., Purver, M., Mishra, R., Raya, M., Peters, S., Meng, Y., Cavedon, L., and Shriberg, E. (2006). Chat: A conversational helper for automotive tasks. In *ICSLP-06*, pp. 1061–1064.

Wilensky, R. (1983). *Planning and Understanding: A Computational Approach to Human Reasoning*. Addison-Wesley.

Williams, J. D. and Young, S. J. (2000). Partially observable markov decision processes for spoken dialog systems. *Computer Speech and Language*, *21*(1), 393–422.

Williams, J. D. and Young, S. J. (2005). Scaling up pomdps for dialog management: The ”summary pomdp” method. In *IEEE ASRU-05*.

Wittgenstein, L. (1953). *Philosophical Investigations. (Translated by Anscombe, G.E.M.)*. Blackwell, Oxford.

Woszczyna, M. and Waibel, A. (1994). Inferring linguistic structure in spoken language. In *ICSLP-94*, Yokohama, Japan, pp. 847–850.

Xu, W. and Rudnicky, A. I. (2000). Task-based dialog management using an agenda. In *ANLP/NAACL Workshop on Conversational Systems*, Somerset, New Jersey, pp. 42–47.

Yankelovich, N., Levow, G.-A., and Marx, M. (1995). Designing SpeechActs: Issues in speech user interfaces. In *Human Factors in Computing Systems: CHI ’95 Conference Proceedings*, Denver, CO, pp. 369–376. ACM.

Yngve, V. H. (1970). On getting a word in edgewise. In *CLS-70*, pp. 567–577. University of Chicago.

Young, S. J. (2002). The statistical approach to the design of spoken dialogue systems. Tech. rep. CUED/F-INFENG/TR.433, Cambridge University Engineering Department, Cambridge, England.

Zue, V. W., Glass, J., Goodine, D., Leung, H., Phillips, M., Polifroni, J., and Seneff, S. (1989). Preliminary evaluation of the VOYAGER spoken language system. In *Proceedings DARPA Speech and Natural Language Workshop*, Cape Cod, MA, pp. 160–167. Morgan Kaufmann.

25 MACHINE TRANSLATION

The process of translating comprises in its essence the whole secret of human understanding and social communication.
Attributed to Hans-Georg Gadamer

What is translation? On a platter
A poet's pale and glaring head,
A parrot's screech, a monkey's chatter,
And profanation of the dead.
Nabokov, *On Translating Eugene Onegin*

Proper words in proper places
Jonathan Swift

MACHINE TRANSLATION
MT

This chapter introduces techniques for **machine translation** (**MT**), the use of computers to automate some or all of the process of translating from one language to another. Translation, in its full generality, is a difficult, fascinating, and intensely human endeavor, as rich as any other area of human creativity. Consider the following passage from the end of Chapter 45 of the 18th-century novel *The Story of the Stone*, also called *Dream of the Red Chamber*, by Cao Xue Qin (Cao, 1792), transcribed in the Mandarin dialect:

> dai yu zi zai chuang shang gan nian bao chai. . . you ting jian chuang wai zhu shao xiang ye zhe shang, yu sheng xi li, qing han tou mu, bu jue you di xia lei lai.

Fig. 25.1 shows the English translation of this passage by David Hawkes, in sentences labeled E_1-E_4. For ease of reading, instead of giving the Chinese, we have shown the English glosses of each Chinese word IN SMALL CAPS. Words in blue are Chinese words not translated into English, or English words not in the Chinese. We have shown **alignment** lines between words that roughly correspond in the two languages.

Consider some of the issues involved in this translation. First, the English and Chinese texts are very different structurally and lexically. The four English sentences

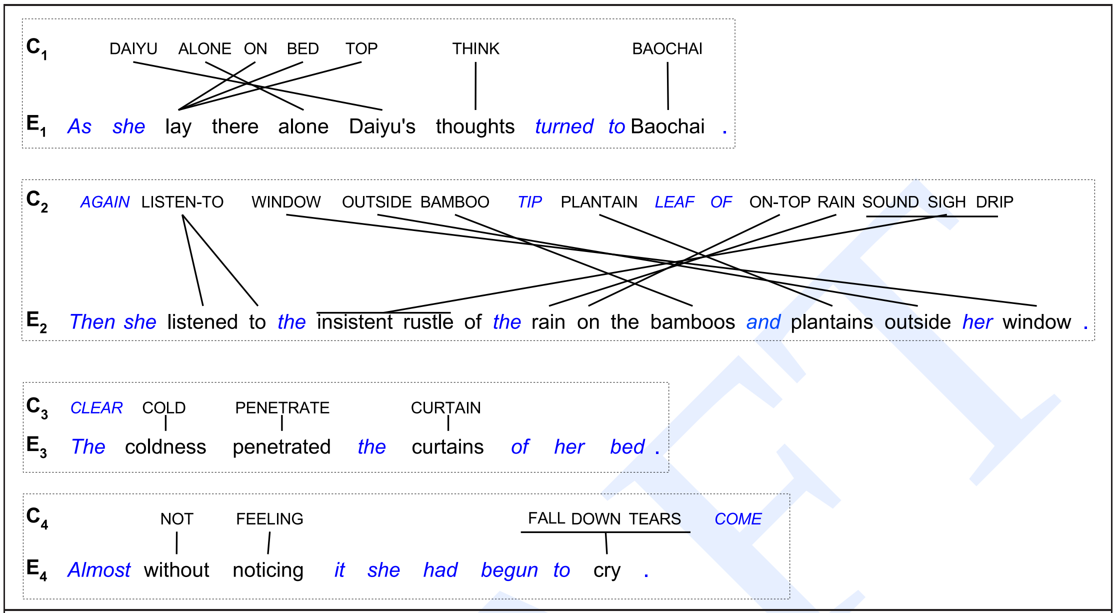

C1
DAIYU ALONE ON BED TOP THINK BAOCHAI
E1
As she lay there alone Daiyu's thoughts turned to Baochai .
C2
AGAIN LISTEN-TO WINDOW OUTSIDE BAMBOO TIP PLANTAIN LEAF OF ON-TOP RAIN SOUND SIGH DRIP
E2
Then she listened to the insistent rustle of the rain on the bamboos and plantains outside her window .
C3
CLEAR COLD PENETRATE CURTAIN
E3
The coldness penetrated the curtains of her bed .
C4
NOT FEELING FALL DOWN TEARS COME
E4
Almost without noticing it she had begun to cry .

Figure 25.1 A Chinese passage from *Dream of the Red Chamber*, with the Chinese words represented by English glosses IN SMALL CAPS. Alignment lines are drawn between 'Chinese' words and their English translations. Words in italics are Chinese words not translated into English, or English words not in the original Chinese.

(notice the periods in blue) correspond to one long Chinese sentence. The word order of the two texts is very different, as we can see by the many crossed alignment lines in Fig. 25.1. The English has many more words than the Chinese, as we can see by the large number of English words marked in blue. Many of these differences are caused by structural differences between the two languages. For example, because Chinese rarely marks verbal aspect or tense; the English translation has additional words like *as*, *turned to*, and *had begun*, and Hawkes had to decide to translate Chinese *tou* as *penetrated*, rather than say *was penetrating* or *had penetrated*. Chinese has less articles than English, explaining the large number of blue *the*s. Chinese also uses far fewer pronouns than English, so Hawkes had to insert *she* and *her* in many places into the English translation.

Stylistic and cultural differences are another source of difficulty for the translator. Unlike English names, Chinese names are made up of regular content words with meanings. Hawkes chose to use transliterations (*Daiyu*) for the names of the main characters but to translate names of servants by their meanings (Aroma, Skybright). To make the image clear for English readers unfamiliar with Chinese bed-curtains, Hawkes translated *ma* ('curtain') as *curtains of her bed*. The phrase *bamboo tip plantain leaf*, although elegant in Chinese, where such four-character phrases are a hallmark of literate prose, would be awkward if translated word-for-word into English, and so Hawkes used simply *bamboos and plantains*.

Translation of this sort clearly requires a deep and rich understanding of the source language and the input text, and a sophisticated, poetic, and creative command of the target language. The problem of automatically performing high-quality literary translation between languages as different as Chinese to English is thus far too hard to automate completely.

However, even non-literary translations between such similar languages as English and French can be difficult. Here is an English sentence from the Hansards corpus of Canadian parliamentary proceedings, with its French translation:

> **English**: Following a two-year transitional period, the new Foodstuffs Ordinance for Mineral Water came into effect on April 1, 1988. Specifically, it contains more stringent requirements regarding quality consistency and purity guarantees.
> **French:** La nouvelle ordonnance fèdèrale sur les denrées alimentaires concernant entre autres les eaux minérales, entrée en vigueur le ler avril 1988 aprés une période transitoire de deux ans. exige surtout une plus grande constance dans la qualité et une garantie de la pureté.
> **French gloss:** THE NEW ORDINANCE FEDERAL ON THE STUFF FOOD CONCERNING AMONG OTHERS THE WATERS MINERAL CAME INTO EFFECT THE 1ST APRIL 1988 AFTER A PERIOD TRANSITORY OF TWO YEARS REQUIRES ABOVE ALL A LARGER CONSISTENCY IN THE QUALITY AND A GUARANTEE OF THE PURITY.

Despite the strong structural and vocabulary overlaps between English and French, such translation, like literary translation, still has to deal with differences in word order (e.g., the location of the *following a two-year transitional period* phrase) and in structure (e.g., English uses the noun *requirements* while the French uses the verb *exige* 'REQUIRE').

Nonetheless, such translations are much easier, and a number of non-literary translation tasks can be addressed with current computational models of machine translation, including: (1) tasks for which a **rough translation** is adequate, (2) tasks where a human **post-editor** is used, and (3) tasks limited to small **sublanguage** domains in which **fully automatic high quality translation** (FAHQT) is still achievable.

FULLY AUTOMATIC HIGH QUALITY TRANSLATION

Information acquisition on the web is the kind of task where a rough translation may still be useful. Suppose you were at the market this morning and saw some lovely *plátanos* (plantains, a kind of banana) at the local Caribbean grocery store and you want to know how to cook them. You go to the web, and find the following recipe:

Platano en Naranja Para 6 personas
3 Plátanos maduros
1 taza de jugo (zumo) de naranja
1/8 cucharadita de nuez moscada en polvo
1 cucharada de canela en polvo (opcional)
2 cucharadas de mantequilla derretida
5 cucharadas de azúcar morena o blanc
1 cucharada de ralladura de naranja

Pelar los plátanos, cortarlos por la mitad y, luego, a lo largo. Engrasar una fuente o pirex con margarina. Colocar los plátanos y bañarlos con la mantequilla derretida. En un recipiente hondo, mezclar el jugo (zumo) de naranja con el azúcar, jengibre, nuez moscada y ralladura de naranja. Verter sobre los plátanos y hornear a 325 ° F. Los primeros 15 minutos, dejar los pátanos cubiertos, hornear 10 o 15 minutos más destapando los plátanos

An MT engine produces the following translation:

Platano in Orange For 6 people
3 Bananas mature 2 tablespoon melted butter
1 cup juice (juice) orange 5 tablespoons brown sugar or white
1/8 teaspoon nutmeg powder 1 tablespoon ralladura orange
1 tablespoon cinnamon powder (optional)
Peel bananas, cut in half and then along. Grease a source or pirex with margarine. Put bananas and showering them with the melted butter. In a deep bowl, mix the juice (juice) orange with the sugar, ginger, nutmeg and ralladura orange. Pour over bananas and bake to 350° F. The first 15 minutes, leave covered bananas, bake 10 to 15 minutes more uncovering bananas.

While there are still lots of confusions in this translation (is it for bananas or plantains? What exactly is the pot we should use? What is *ralladura*?) it's probably enough, perhaps after looking up one or two words, to get a basic idea of something to try in the kitchen with your new purchase!

POST-EDITING

COMPUTER-AIDED HUMAN TRANSLATION

LOCALIZATION

An MT system can also be used to speed-up the human translation process, by producing a draft translation that is fixed up in a **post-editing** phase by a human translator. Strictly speaking, systems used in this way are doing **computer-aided human translation** (CAHT or CAT) rather than (fully automatic) machine translation. This model of MT usage is effective especially for high volume jobs and those requiring quick turn-around, such as the translation of software manuals for **localization** to reach new markets.

SUBLANGUAGE

Weather forecasting is an example of a **sublanguage** domain that can be modeled completely enough to use raw MT output even without post-editing. Weather forecasts consist of phrases like *Cloudy with a chance of showers today and Thursday*, or *Outlook for Friday: Sunny*. This domain has a limited vocabulary and only a few basic phrase types. Ambiguity is rare, and the senses of ambiguous words are easily disambiguated based on local context, using word classes and semantic features such as WEEKDAY, PLACE, or TIME POINT. Other domains that are sublanguage-like include equipment maintenance manuals, air travel queries, appointment scheduling, and restaurant recommendations.

Applications for machine translation can also be characterized by the number and direction of the translations. Localization tasks like translations of computer manuals require one-to-many translation (from English into many languages). One-to-many translation is also needed for non-English speakers around the world to access web information in English. Conversely, many-to-one translation (into English) is relevant for anglophone readers who need the gist of web content written in other languages. Many-to-many translation is relevant for environments like the European Union, where 23 official languages (at the time of this writing) need to be intertranslated.

Before we turn to MT systems, we begin in section 25.1 by summarizing key differences among languages. The three classic models for doing MT are then presented in Sec. 25.2: the **direct**, **transfer**, and **interlingua** approaches. We then investigate in detail modern **statistical MT** in Secs. 25.3-25.8, finishing in Sec. 25.9 with a discussion of **evaluation**.

25.1 Why is Machine Translation So Hard?

TRANSLATION DIVERGENCES

We began this chapter with some of the issues that made it hard to translate *The Story of the Stone* from Chinese to English. In this section we look in more detail about what makes translation difficult. We'll discuss what makes languages similar or different, including **systematic** differences that we can model in a general way, as well as **idiosyncratic** and lexical differences that must be dealt with one by one. These differences between languages are referred to as **translation divergences** and an understanding of what causes them will help us in building models that overcome the differences (Dorr, 1994).

25.1.1 Typology

UNIVERSAL

When you accidentally pick up a radio program in some foreign language it seems like chaos, completely unlike the familiar languages of your everyday life. But there are patterns in this chaos, and indeed, some aspects of human language seem to be **universal**, holding true for every language. Many universals arise from the functional role of language as a communicative system by humans. Every language, for example, seems to have words for referring to people, for talking about women, men, and children, eating and drinking, for being polite or not. Other universals are more subtle; for example Ch. 5 mentioned that every language seems to have nouns and verbs.

TYPOLOGY

Even when languages differ, these differences often have systematic structure. The study of systematic cross-linguistic similarities and differences is called **typology** (Croft (1990), Comrie (1989)). This section sketches some typological facts about crosslinguistic similarity and difference.

ISOLATING

POLYSYNTHETIC

AGGLUTINATIVE

FUSION

Morphologically, languages are often characterized along two dimensions of variation. The first is the number of morphemes per word, ranging from **isolating** languages like Vietnamese and Cantonese, in which each word generally has one morpheme, to **polysynthetic** languages like Siberian Yupik ("Eskimo"), in which a single word may have very many morphemes, corresponding to a whole sentence in English. The second dimension is the degree to which morphemes are segmentable, ranging from **agglutinative** languages like Turkish (discussed in Ch. 3), in which morphemes have relatively clean boundaries, to **fusion** languages like Russian, in which a single affix may conflate multiple morphemes, like *-om* in the word *stolom*, (table-SG-INSTR-DECL1) which fuses the distinct morphological categories instrumental, singular, and first declension.

SVO

SOV

VSO

Syntactically, languages are perhaps most saliently different in the basic word order of verbs, subjects, and objects in simple declarative clauses. German, French, English, and Mandarin, for example, are all **SVO** (**Subject-Verb-Object**) languages, meaning that the verb tends to come between the subject and object. Hindi and Japanese, by contrast, are **SOV** languages, meaning that the verb tends to come at the end of basic clauses, while Irish, Arabic, and Biblical Hebrew are **VSO** languages. Two languages that share their basic word-order type often have other similarities. For example **SVO** languages generally have **prepositions** while **SOV** languages generally have **postpositions**.

For example in the following SVO English sentence, the verb *adores* is followed by its argument VP *listening to music*, the verb *listening* is followed by its argument PP *to music*, and the preposition *to* is followed by its argument *music*. By contrast, in the Japanese example which follows, each of these orderings is reversed; both verbs are *preceded* by their arguments, and the postposition follows its argument.

(25.1) English: *He adores listening to music*

Japanese:	*kare*	*ha*	*ongaku*	*wo*	*kiku*	*no*	*ga*	*daisuki*	*desu*
	he		music	to	listening			adores	

HEAD-MARKING

Another important dimension of typological variation has to do with **argument structure** and **linking** of predicates with their arguments, such as the difference between **head-marking** and **dependent-marking** languages (Nichols, 1986). Head-marking languages tend to mark the relation between the head and its dependents on the head. Dependent-marking languages tend to mark the relation on the non-head. Hungarian, for example, marks the possessive relation with an affix (A) on the head noun (H), where English marks it on the (non-head) possessor:

(25.2)				
	English:	*the*	*man-*^{A}*'s*	^{H}*house*
	Hungarian:	*az*	*ember*	^{H}*ház-*^{A}*a*
		the	man	house-his

Typological variation in linking can also relate to how the conceptual properties of an event are mapped onto specific words. Talmy (1985) and (1991) noted that languages can be characterized by whether direction of motion and manner of motion are marked on the verb or on the "satellites": particles, prepositional phrases, or adverbial phrases. For example a bottle floating out of a cave would be described in English with the direction marked on the particle *out*, while in Spanish the direction would be marked on the verb:

(25.3) English: *The bottle floated out.*

Spanish:	La	botella	salió	flotando.
	The	bottle	exited	floating.

VERB-FRAMED

SATELLITE-FRAMED

Verb-framed languages mark the direction of motion on the verb (leaving the satellites to mark the manner of motion), like Spanish *acercarse* 'approach', *alcanzar* 'reach', *entrar* 'enter', *salir* 'exit'. **Satellite-framed** languages mark the direction of motion on the satellite (leaving the verb to mark the manner of motion), like English *crawl out*, *float off*, *jump down*, *walk over to*, *run after*. Languages like Japanese, Tamil, and the many languages in the Romance, Semitic, and Mayan languages families, are verb-framed; Chinese as well as non-Romance Indo-European languages like English, Swedish, Russian, Hindi, and Farsi, are satellite-framed (Talmy, 1991; Slobin, 1996).

Finally, languages vary along a typological dimension related to the things they can omit. Many languages require that we use an explicit pronoun when talking about a referent that is given in the discourse. In other languages, however, we can sometimes omit pronouns altogether as the following examples from Spanish and Chinese show, using the $\emptyset$-notation introduced in Ch. 21:

(25.4) [El jefe]$_i$ dio con un libro. $\emptyset_i$ Mostró a un descifrador ambulante.
[The boss] came upon a book. [He] showed it to a wandering decoder.

(25.5) CHINESE EXAMPLE

PRO-DROP

REFERENTIAL DENSITY

COLD

HOT

Languages which can omit pronouns in these ways are called **pro-drop** languages. Even among the pro-drop languages, their are marked differences in frequencies of omission. Japanese and Chinese, for example, tend to omit far more than Spanish. We refer to this dimension as **referential density**; languages which tend to use more pronouns are more referentially dense than those that use more zeros. Referentially sparse languages, like Chinese or Japanese, that require the hearer to do more inferential work to recover antecedents are called **cold** languages. Languages that are more explicit and make it easier for the hearer are called **hot** languages. The terms *hot* and *cold* are borrowed from Marshall McLuhan's (1964) distinction between hot media like movies, which fill in many details for the viewer, versus cold media like comics, which require the reader to do more inferential work to fill out the representation (Bickel, 2003).

Each typological dimension can cause problems when translating between languages that differ along them. Obviously translating from SVO languages like English to SOV languages like Japanese requires huge structural reorderings, since all the constituents are at different places in the sentence. Translating from a satellite-framed to a verb-framed language, or from a head-marking to a dependent-marking language, requires changes to sentence structure and constraints on word choice. Languages with extensive pro-drop, like Chinese or Japanese, cause huge problems for translation into non-pro-drop languages like English, since each zero has to be identified and the anaphor recovered.

25.1.2 Other Structural Divergences

Many structural divergences between languages are based on typological differences. Others, however, are simply idiosyncratic differences that are characteristic of particular languages or language pairs. For example in English the unmarked order in a noun-phrase has adjectives precede nouns, but in French and Spanish adjectives generally follow nouns. [1]

(25.6)

Spanish	*bruja*	*verde*	**French**	*maison*	*bleue*
	witch	green		house	blue
English	"green witch"			"blue house"	

Chinese relative clauses are structured very differently than English relative clauses, making translation of long Chinese sentences very complex.

Language-specific constructions abound. English, for example, has an idiosyncratic syntactic construction involving the word *there* that is often used to introduce a new scene in a story, as in *there burst into the room three men with guns*. To give an idea of how trivial, yet crucial, these differences can be, think of dates. Dates not only appear in various formats — typically DD/MM/YY in British English, MM/DD/YY in American English, and YYMMDD in Japanese—but the calendars themselves may also differ. Dates in Japanese, for example, are often relative to the start of the current Emperor's reign rather than to the start of the Christian Era.

[1] As always, there are exceptions to this generalization, such as *galore* in English and *gros* in French; furthermore in French some adjectives can appear before the noun with a different meaning; *route mauvaise* 'bad road, badly-paved road' versus *mauvaise route* 'wrong road' (Waugh, 1976).

25.1.3 Lexical Divergences

Lexical divergences also cause huge difficulties in translation. We saw in Ch. 20, for example, that the English source language word *bass* could appear in Spanish as the fish *lubina* or the instrument *bajo*. Thus translation often requires solving the exact same problems as word sense disambiguation, and the two fields are closely linked.

In English the word *bass* is homonymous; the two senses of the word are not closely related semantically, and so it is natural that we would have to disambiguate in order to translate. Even in cases of polysemy, however, we often have to disambiguate if the target language doesn't have the exact same kind of polysemy. The English word *know*, for example, is polysemous; it can refer to knowing of a fact or proposition (*I know that snow is white*) or familiarity with a person or location (*I know Jon Stewart*). It turns out that translating these different senses requires using distinct French verbs, including the verbs *connaître*, and *savoir*. *Savoir* is generally used with sentential complements to indicate knowledge or mental representation of a fact or proposition, or verbal complements to indicate knowledge of how to do something (e.g., WordNet 3.0 senses #1, #2, #3). *Connaître* is generally used with NP complements to indicate familiarity or acquaintance with people, entities, or locations (e.g., WordNet 3.0 senses #4, #7). Similar distinctions occur in German, Chinese, and many other languages:

(25.7) **English:** I know he just bought a book.
(25.8) **French:** Je sais qu'il vient d'acheter un livre.

(25.9) **English:** I know John.
(25.10) **French:** Je connais Jean.

The *savoir/connaître* distinction corresponds to different groups of WordNet senses. Sometimes, however, a target language will make a distinction that is not even recognized in fine-grained dictionaries. German, for example, uses two distinct words for what in English would be called a *wall*: *Wand* for walls inside a building, and *Mauer* for walls outside a building. Similarly, where English uses the word *brother* for any male sibling, both Japanese and Chinese have distinct words for *older brother* and *younger brother* (Chinese *gege* and *didi*, respectively).

In addition to these distinctions, lexical divergences can be grammatical. For example, a word may translate best to a different part-of-speech in the target language. Many English sentences involving the verb *like* must be translated into German using the adverbial *gern*; thus *she likes to sing* maps to *sie singt gerne* (SHE SINGS LIKINGLY).

In translation, we can think of sense disambiguation as a kind of **specification**; we have to make a vague word like *know* or *bass* more specific in the target language. This kind of specification is also quite common with grammatical differences. Sometimes one language places more grammatical constraints on word choice than another. French and Spanish, for example, marks gender on adjectives, so an English translation into French requires specifying adjective gender. English distinguishes gender in pronouns where Mandarin does not; thus translating a third-person singular pronoun *tā* from Mandarin to English (*he*, *she*, or *it*) requires deciding who the original referent was. In Japanese, because there is no single word for *is*, the translator must choose between *iru* or *aru*, based on whether the subject is animate or not.

The way that languages differ in lexically dividing up conceptual space may be more complex than this one-to-many translation problem, leading to many-to-many mappings. For example Fig. 25.2 summarizes some of the complexities discussed by Hutchins and Somers (1992) in relating English *leg*, *foot*, and *paw*, to the French *jambe*, *pied*, *patte*, etc.

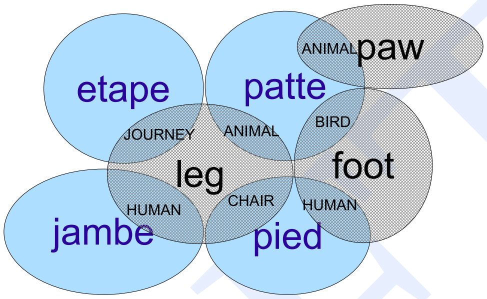


Figure 25.2 The complex overlap between English *leg*, *foot*, etc, and various French translations like *patte* discussed by Hutchins and Somers (1992) .

LEXICAL GAP

Further, one language may have a **lexical gap**, where no word or phrase, short of an explanatory footnote, can express the meaning of a word in the other language. For example, Japanese does not have a word for *privacy*, and English does not have a word for Japanese *oyakoko* or Chinese *xiáo* (we make do with the awkward phrase *filial piety* for both).

25.2 Classical MT & the Vauquois Triangle

The next few sections introduce the classical pre-statistical architectures for machine translation. Real systems tend to involve combinations of elements from these three architectures; thus each is best thought of as a point in an algorithmic design space rather than as an actual algorithm.

In **direct** translation, we proceed word-by-word through the source language text, translating each word as we go. Direct translation uses a large bilingual dictionary, each of whose entries is a small program with the job of translating one word. In **transfer** approaches, we first parse the input text, and then apply rules to transform the source language parse structure into a target language parse structure. We then generate the target language sentence from the parse structure. In **interlingua** approaches, we analyze the source language text into some abstract meaning representation, called an **interlingua**. We then generate into the target language from this interlingual representation.

VAUQUOIS TRIANGLE

A common way to visualize these three approaches is with **Vauquois triangle** shown in Fig. 25.3. The triangle shows the increasing depth of analysis required (on both the analysis and generation end) as we move from the direct approach through transfer approaches, to interlingual approaches. In addition, it shows the decreasing amount of transfer knowledge needed as we move up the triangle, from huge amounts of transfer at the direct level (almost all knowledge is transfer knowledge for each word) through transfer (transfer rules only for parse trees or thematic roles) through interlingua (no specific transfer knowledge).

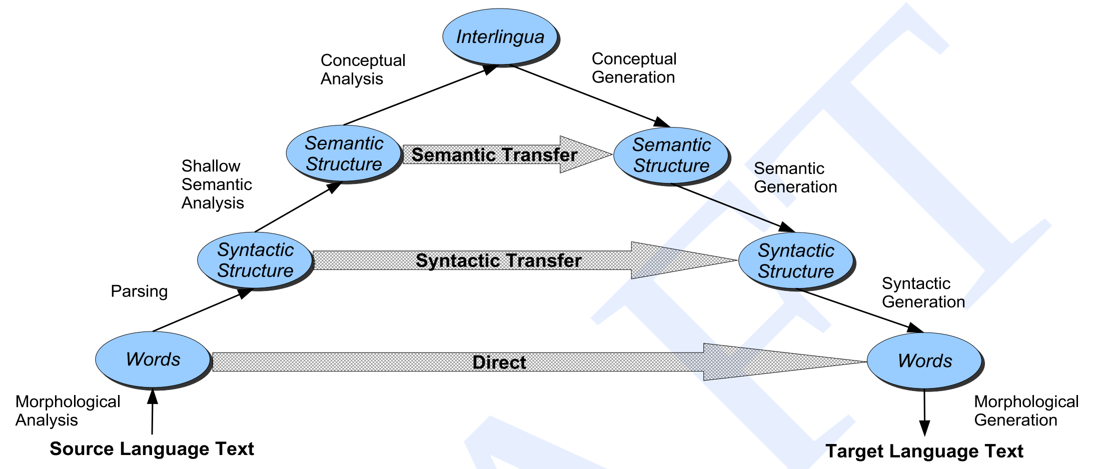


Figure 25.3 The Vauquois triangle.

In the next sections we'll see how these algorithms address some of the four translation examples shown in Fig. 25.4

English	Mary didn't slap the green witch								
⇒ Spanish	*Maria*	*no*	*dió*	*una*	*bofetada*	*a*	*la*	*bruja*	*verde*
	Mary	not	gave	a	slap	to	the	witch	green
English	The green witch is at home this week								
⇒ German	*Diese*	*Woche*	*ist*	*die*	*grüne*	*Hexe*	*zu*	*Hause.*	
	this	week	is	the	green	witch	at	house	
English	He adores listening to music								
⇒ Japanese	*kare*	*ha*	*ongaku*	*wo*	*kiku*	*no*	*ga*	*daisuki*	*desu*
	he		music	to	listening			adores	
Chinese	*cheng long*	*dao*	*xiang gang*	*qu*					
	Jackie Chan	to	Hong Kong	go					
⇒ English	Jackie Chan went to Hong Kong								

Figure 25.4 Example sentences used throughout the chapter.

25.2.1 Direct Translation

DIRECT TRANSLATION

In **direct translation**, we proceed word-by-word through the source language text, translating each word as we go. We make use of no intermediate structures, except for shallow morphological analysis; each source word is directly mapped onto some target word. Direct translation is thus based on a large bilingual dictionary; each entry in the dictionary can be viewed as a small program whose job is to translate one word. After the words are translated, simple reordering rules can apply, for example for moving adjectives after nouns when translating from English to French.

The guiding intuition of the direct approach is that we translate by incrementally **transforming** the source language text into a target language text. While the pure direct approach is no longer used, this transformational intuition underlies all modern systems, both statistical and non-statistical.

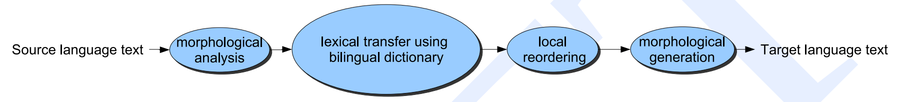


Figure 25.5 Direct machine translation. The major component, indicated by size here, is the bilingual dictionary.

Let's look at a simplified direct system on our first example, translating from English into Spanish:

(25.11) Mary didn't slap the green witch

Maria	*no*	*dió*	*una*	*bofetada*	*a*	*la*	*bruja*	*verde*
Mary	not	gave	a	slap	to	the	witch	green

The four steps outlined in Fig. 25.5 would proceed as shown in Fig. 25.6.

Step 2 presumes that the bilingual dictionary has the phrase *dar una bofetada a* as the Spanish translation of English *slap*. The local reordering step 3 would need to switch the adjective-noun ordering from *green witch* to *bruja verde*. And some combination of ordering rules and the dictionary would deal with the negation and past tense in English *didn't*. These dictionary entries can be quite complex; a sample dictionary entry from an early direct English-Russian system is shown in Fig. 25.7.

While the direct approach can deal with our simple Spanish example, and can handle single-word reorderings, it has no parsing component or indeed any knowledge about phrasing or grammatical structure in the source or target language. It thus cannot reliably handle longer-distance reorderings, or those involving phrases or larger structures. This can happen even in languages very similar to English, like German, where adverbs like *heute* ('today') occur in different places, and the subject (e.g., *die grüne Hexe*) can occur after the main verb, as shown in Fig. 25.8.

Input:	Mary didn't slap the green witch
After 1: Morphology	Mary DO-PAST not slap the green witch
After 2: Lexical Transfer	Maria PAST no dar una bofetada a la verde bruja
After 3: Local reordering	Maria no dar PAST una bofetada a la bruja verde
After 4: Morphology	Maria no dió una bofetada a la bruja verde

Figure 25.6 An example of processing in a direct system

```
function DIRECT_TRANSLATE_MUCH/MANY(word) returns Russian translation

if preceding word is how return skol'ko
else if preceding word is as return stol'ko zhe
else if word is much
   if preceding word is very return nil
   else if following word is a noun return mnogo
else /* word is many */
   if preceding word is a preposition and following word is a noun return mnogii
   else return mnogo
```

Figure 25.7 A procedure for translating *much* and *many* into Russian, adapted from Hutchins' (1986, pg. 133) discussion of Panov 1960. Note the similarity to decision list algorithms for word sense disambiguation.

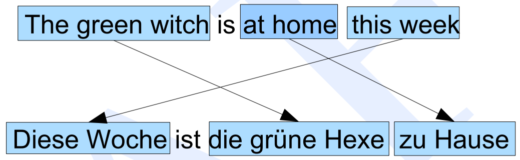


Figure 25.8 Complex reorderings necessary when translating from English to German. German often puts adverbs in initial position that English would more naturally put later. German tensed verbs often occur in second position in the sentence, causing the subject and verb to be inverted.

Similar kinds of reorderings happen between Chinese (where goal PPs often occur preverbally) and English (where goal PPs must occur postverbally), as shown in Fig. 25.9.

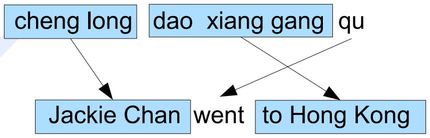


Figure 25.9 Chinese goal PPs often occur preverbally, unlike in English

.

Finally, even more complex reorderings occur when we translate from SVO to SOV languages, as we see in the English-Japanese example from Yamada and Knight (2002):

(25.12) He adores listening to music

kare	*ha*	*ongaku*	*wo*	*kiku*	*no*	*ga*	*daisuki*	*desu*
he		music	to	listening			adores	

These three examples suggest that the direct approach is too focused on individual words, and that in order to deal with real examples we'll need to add phrasal and structural knowledge into our MT models. We'll flesh out this intuition in the next section.

25.2.2 Transfer

CONTRASTIVE KNOWLEDGE

TRANSFER MODEL

As Sec. 25.1 illustrated, languages differ systematically in structural ways. One strategy for doing MT is to translate by a process of overcoming these differences, altering the structure of the input to make it conform to the rules of the target language. This can be done by applying **contrastive knowledge**, that is, knowledge about the differences between the two languages. Systems that use this strategy are said to be based on the **transfer model**.

The transfer model presupposes a parse of the source language, and is followed by a generation phase to actually create the output sentence. Thus, on this model, MT involves three phases: **analysis**, **transfer**, and **generation**, where transfer bridges the gap between the output of the source language parser and the input to the target language generator.

It is worth noting that a parse for MT may differ from parses required for other purposes. For example, suppose we need to translate *John saw the girl with the binoculars* into French. The parser does not need to bother to figure out where the prepositional phrase attaches, because both possibilities lead to the same French sentence.

Once we have parsed the source language, we'll need rules for **syntactic transfer** and **lexical transfer**. The syntactic transfer rules will tell us how to modify the source parse tree to resemble the target parse tree.

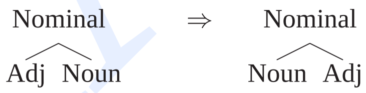


Figure 25.10 A simple transformation that reorders adjectives and nouns

SYNTACTIC TRANSFORMATIONS

Figure 25.10 gives an intuition for simple cases like adjective-noun reordering; we transform one parse tree, suitable for describing an English phrase, into another parse tree, suitable for describing a Spanish sentence. These **syntactic transformations** are operations that map from one tree structure to another.

The transfer approach and this rule can be applied to our example *Mary did not slap the green witch*. Besides this transformation rule, we'll need to assume that the morphological processing figures out that *didn't* is composed of *do-PAST* plus *not*, and that the parser attaches the PAST feature onto the VP. Lexical transfer, via lookup in the bilingual dictionary, will then remove *do*, change *not* to *no*, and turn *slap* into the phrase *dar una bofetada a*, with a slight rearrangement of the parse tree, as suggested in Fig. 25.11.

For translating from SVO languages like English to SOV languages like Japanese, we'll need even more complex transformations, for moving the verb to the end, changing prepositions into postpositions, and so on. An example of the result of such rules is shown in Fig. 25.12. An informal sketch of some transfer rules is shown in Fig. 25.13.

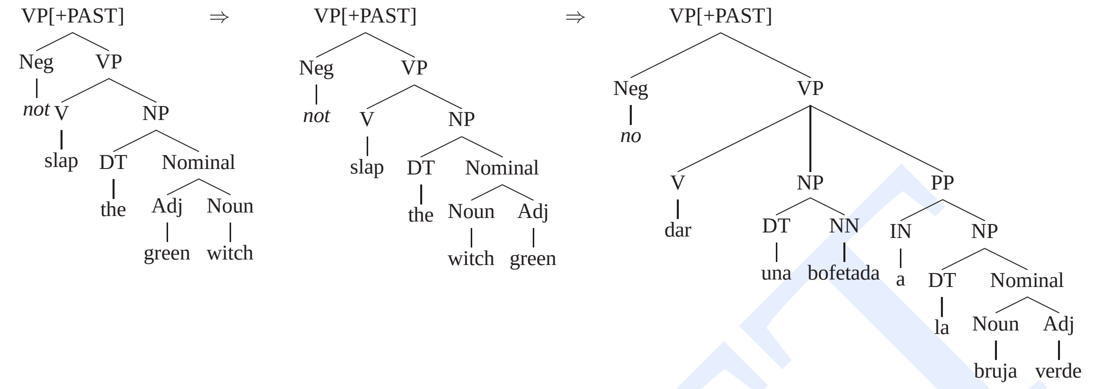


Figure 25.11 A further sketch of the transfer approach.

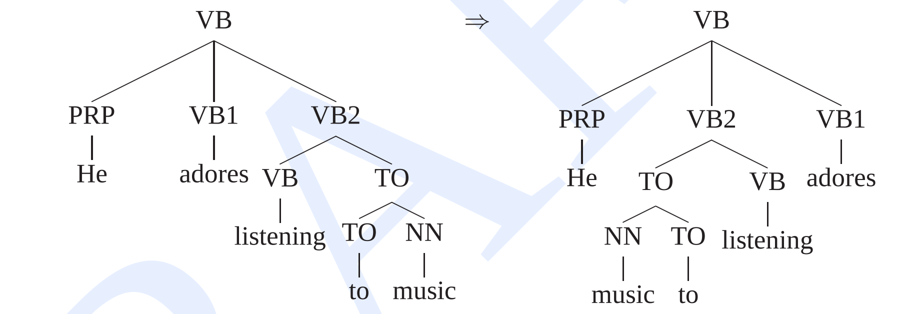


Figure 25.12 The result of syntactic transformations from English order (SVO) to Japanese order (SOV) for the sentence *He adores listening to music* (*kare ha ongaku wo kiku no ga daisuki desu*), after Yamada and Knight (2001). This transform would require rules for moving verbs after their NP and VP complements, and changing prepositions to postpositions.

English to Spanish:			
1.	NP → Adjective$_1$ Noun$_2$	⇒	NP → Noun$_2$ Adjective$_1$
Chinese to English:			
2.	VP → PP[+Goal] V	⇒	VP → V PP[+Goal]
English to Japanese:			
3.	VP → V NP	⇒	VP → NP V
4.	PP → P NP	⇒	PP → NP P
5.	NP → NP$_1$ Rel. Clause$_2$	⇒	NP → Rel. Clause$_2$ NP$_1$

Figure 25.13 An informal description of some transformations.

Transfer systems can be based on richer structures than just pure syntactic parses. For example a transfer based system for translating Chinese to English might have rules to deal with the fact shown in Fig. 25.9 that in Chinese PPs that fill the semantic role GOAL (like *to the store* in *I went to the store*) tend to appear before the verb, while in English these goal PPs must appear after the verb. In order to build a transformation to deal with this and related PP ordering differences, the parse of the Chinese must including thematic structure, so as to distinguish BENEFACTIVE PPs (which must occur before the verb) from DIRECTION and LOCATIVE PPs (which preferentially occur before the verb) from RECIPIENT PPs (which occur after) (Li and Thompson, 1981). We discussed how to do this kind of semantic role labeling in Ch. 20. Using semantic roles in this way is generally called **semantic transfer**; a simple such transformation is shown in Fig. 25.13.

SEMANTIC TRANSFER

In addition to syntactic transformations, transfer-based systems need to have lexical transfer rules. Lexical transfer is generally based on a bilingual dictionary, just as for direct MT. The dictionary itself can also be used to deal with problems of lexical ambiguity. For example the English word *home* has many possible translations in German, including *nach Hause* (in the sense of *going home*) *Heim* (in the sense of a *home game*), *Heimat* (in the sense of *homeland*, *home country*, or *spiritual home*), and *zu Hause* (in the sense of being *at home*). In this case, the phrase *at home* is very likely to be translated *zu Hause*, and so the bilingual dictionary can list this translation idiomatically.

Many cases of lexical transfer are too complex to deal with via a phrasal dictionary. In these cases transfer systems can do disambiguation during the source language analysis, by applying the sense disambiguation techniques of Ch. 20.

25.2.3 Combining direct and tranfer approaches in classic MT

Although the transfer metaphor offers the ability to deal with more complex source language phenomena than the direct approach, it turns out the simple SVO → SOV rules we've described above are not sufficient. In practice, we need messy rules which combine rich lexical knowledge of both languages with syntactic and semantic features. We briefly saw an example of such a rule for changing *slap* to *dar una bofetada a*.

For this reason, commercial MT systems tend to be combinations of the direct and transfer approaches, using rich bilingual dictionaries, but also using taggers and parsers. The Systran system, for example, as described in Hutchins and Somers (1992), Senellart et al. (2001), has three components. First is a shallow **analysis** stage, including:

- morphological analysis and part of speech tagging
- chunking of NPs, PPs, and larger phrases
- shallow dependency parsing (subjects, passives, head-modifiers)

Next is a **transfer** phase, including:

- translation of idioms,
- word sense disambiguation
- assigning prepositions based on governing verbs

Finally, in the **synthesis** stage, the system:

- applies a rich bilingual dictionary to do lexical translation

- deals with reorderings
- performs morphological generation

Thus like the direct system, the Systran system relies for much of its processing on the bilingual dictionary, which has lexical, syntactic, and semantic knowledge. Also like a direct system, Systran does reordering in a post-processing step. But like a transfer system, many of the steps are informed by syntactic and shallow semantic processing of the source language.

25.2.4 The Interlingua Idea: Using Meaning

One problem with the transfer model is that it requires a distinct set of transfer rules for each pair of languages. This is clearly suboptimal for translation systems employed in many-to-many multilingual environments like the European Union.

This suggests a different perspective on the nature of translation. Instead of directly transforming the words of the source language sentence into the target language, the interlingua intuition is to treat translation as a process of extracting the meaning of the input and then expressing that meaning in the target language. If this could be done, an MT system could do without contrastive knowledge, merely relying on the same syntactic and semantic rules used by a standard interpreter and generator for the language. The amount of knowledge needed would then be proportional to the number of languages the system handles, rather than to the square.

INTERLINGUA

This scheme presupposes the existence of a meaning representation, or **interlingua**, in a language-independent canonical form, like the semantic representations we saw in Ch. 17. The idea is for the interlingua to represent all sentences that mean the "same" thing in the same way, regardless of the language they happen to be in. Translation in this model proceeds by performing a deep semantic analysis on the input from language X into the interlingual representation and generating from the interlingua to language Y.

What kind of representation scheme can we use as an interlingua? The predicate calculus, or a variant such as minimal recursion semantics, is one possibility. Semantic decomposition into some kind of atomic semantic primitives is another. We will illustrate a third common approach, a simple event-based representation, in which events are linked to their arguments via a small fixed set of thematic roles. Whether we use logics or other representations of events, we'll need to specify temporal and aspectual properties of the events, and we'll also need to represent non-eventive relationships between entities, such as the *has-color* relation between *green* and *witch*. Fig. 25.14 shows a possible interlingual representation for *Mary did not slap the green witch* as a unification-style feature structure.

We can create these interlingual representation from the source language text using the **semantic analyzer** techniques of Ch. 18 and Ch. 20; using a semantic role labeler to discover the AGENT relation between *Mary* and the *slap* event, or the THEME relation between the *witch* and the *slap* event. We would also need to do disambiguation of the noun-modifier relation to recognize that the relationship between *green* and *witch* is the *has-color* relation, and we'll need to discover that this event has negative polarity (from the word *didn't*). The interlingua thus requires more analysis work than the

$$\begin{bmatrix} \text{EVENT} & \text{SLAPPING} \\ \text{AGENT} & \text{MARY} \\ \text{TENSE} & \text{PAST} \\ \text{POLARITY} & \text{NEGATIVE} \\ \text{THEME} & \begin{bmatrix} \text{WITCH} & \\ \text{DEFINITENESS} & \text{DEF} \\ \text{ATTRIBUTES} & \begin{bmatrix} \text{HAS-COLOR} & \text{GREEN} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 25.14 Interlingual representation of *Mary did not slap the green witch.*

transfer model, which only required syntactic parsing (or at most shallow thematic role labeling). But generation can now proceed directly from the interlingua with no need for syntactic transformations.

In addition to doing without syntactic transformations, the interlingual system does without lexical transfer rules. Recall our earlier problem of whether to translate *know* into French as *savoir* or *connaître*. Most of the processing involved in making this decision is not specific to the goal of translating into French; German, Spanish, and Chinese all make similar distinctions, and furthermore the disambiguation of *know* into concepts such as HAVE-A-PROPOSITION-IN-MEMORY and BE-ACQUAINTED-WITH-ENTITY is also important for other NLU applications that require word-senses. Thus by using such concepts in an interlingua, a larger part of the translation process can be done with general language processing techniques and modules, and the processing specific to the English-to-French translation task can be eliminated or at least reduced, as suggested in Fig. 25.3.

The interlingual model has its own problems. For example, in order to translate from Japanese to Chinese the universal interlingua must include concepts such as ELDER-BROTHER and YOUNGER-BROTHER. Using these same concepts translating from German-to-English would then require large amounts of unnecessary disambiguation. Furthermore, doing the extra work involved by the interlingua commitment requires exhaustive analysis of the semantics of the domain and formalization into an ontology. Generally this is only possible in relatively simple domains based on a database model, as in the air travel, hotel reservation, or restaurant recommendation domains, where the database definition determines the possible entities and relations. For these reasons, interlingual systems are generally only used in sublanguage domains.

25.3 Statistical MT

The three classic architectures for MT (direct, transfer, and interlingua) all provide answers to the questions of what representations to use and what steps to perform to translate. But there is another way to approach the problem of translation: to focus on the result, not the process. Taking this perspective, let's consider what it means for a sentence to be a translation of some other sentence.

This is an issue to which philosophers of translation have given a lot of thought. The consensus seems to be, sadly, that it is impossible for a sentence in one language to be a translation of a sentence in other, strictly speaking. For example, one cannot really translate Hebrew *adonai roi* ('the Lord is my shepherd') into the language of a culture that has no sheep. On the one hand, we can write something that is clear in the target language, at some cost in fidelity to the original, something like *the Lord will look after me*. On the other hand, we can be faithful to the original, at the cost of producing something obscure to the target language readers, perhaps like *the Lord is for me like somebody who looks after animals with cotton-like hair*. As another example, if we translate the Japanese phrase *fukaku hansei shite orimasu*, as *we apologize*, we are not being faithful to the meaning of the original, but if we produce *we are deeply reflecting (on our past behavior, and what we did wrong, and how to avoid the problem next time)*, then our output is unclear or awkward. Problems such as these arise not only for culture-specific concepts, but whenever one language uses a metaphor, a construction, a word, or a tense without an exact parallel in the other language.

So, true translation, which is both faithful to the source language and natural as an utterance in the target language, is sometimes impossible. If you are going to go ahead and produce a translation anyway, you have to compromise. This is exactly what translators do in practice: they produce translations that do tolerably well on both criteria.

This provides us with a hint for how to do MT. We can model the goal of translation as the production of an output that maximizes some value function that represents the importance of both faithfulness and fluency. Statistical MT is the name for a class of approaches that do just this, by building probabilistic models of faithfulness and fluency, and then combining these models to choose the most probable translation. If we chose the product of faithfulness and fluency as our quality metric, we could model the translation from a source language sentence S to a target language sentence $\hat{T}$ as:

$$\text{best-translation } \hat{T} = \text{argmax}_T \text{ faithfulness(T,S) fluency(T)}$$

This intuitive equation clearly resembles the Bayesian **noisy channel model** we've seen in Ch. 5 for spelling and Ch. 9 for speech. Let's make the analogy perfect and formalize the noisy channel model for statistical machine translation.

First of all, for the rest of this chapter, we'll assume we are translating from a foreign language sentence $F = f_1, f_2, ..., f_m$ to English. For some examples we'll use French as the foreign language, and for others Spanish. But in each case we are translating **into English** (although of course the statistical model also works for translating out of English). In a probabilistic model, the best English sentence $\hat{E} = e_1, e_2, ..., e_l$ is the one whose probability $P(E|F)$ is the highest. As is usual in the noisy channel model, we can rewrite this via Bayes rule:

$$\begin{aligned}\hat{E} &= argmax_E P(E|F) \\ &= argmax_E \frac{P(F|E)P(E)}{P(F)} \\ &= argmax_E P(F|E)P(E) \qquad (25.13)\end{aligned}$$

We can ignore the denominator $P(F)$ inside the argmax since we are choosing the best English sentence for a fixed foreign sentence F, and hence $P(F)$ is a constant. The

TRANSLATION MODEL
LANGUAGE MODEL

resulting noisy channel equation shows that we need two components: a **translation model** $P(F|E)$, and a **language model** $P(E)$.

$$\hat{E} = \operatorname*{argmax}_{E\in\text{English}} \overbrace{P(F|E)}^{\text{translation model}} \quad \overbrace{P(E)}^{\text{language model}} \tag{25.14}$$

Notice that applying the noisy channel model to machine translation requires that we think of things backwards, as shown in Fig. 25.15. We pretend that the foreign (source language) input F we must translate is a corrupted version of some English (target language) sentence E, and that our task is to discover the hidden (target language) sentence E that generated our observation sentence F.

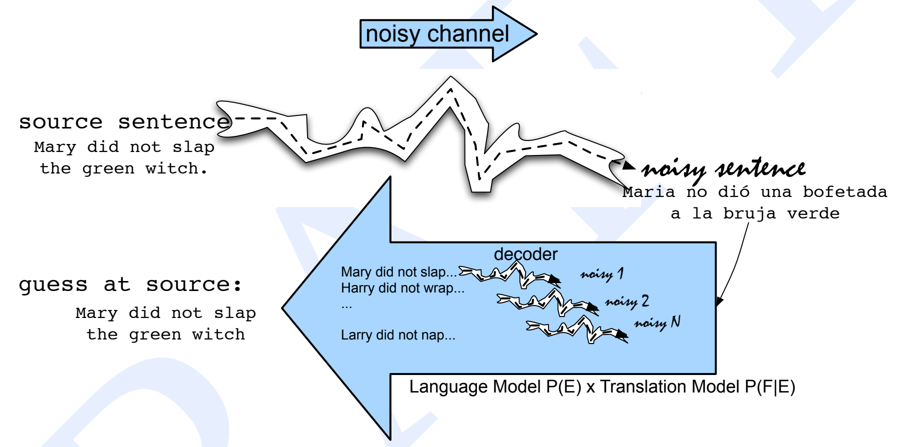


Figure 25.15 The noisy channel model of statistical MT. If we are translating a source language French to a target language English, we have to think of ’sources’ and ’targets’ backwards. We build a model of the generation process from an English sentence through a channel to a French sentence. Now given a French sentence to translate, we pretend it is the output of an English sentence going through the noisy channel, and search for the best possible ‘source’ English sentence.

The noisy channel model of statistical MT thus requires three components to translate from a French sentence F to an English sentence E:

- A **language model** to compute $P(E)$
- A **translation model** to compute $P(F|E)$
- A **decoder**, which is given F and produces the most probable E

Of these three components, we have already introduced the language model $P(E)$ in Ch. 4. Statistical MT systems are based on the same N-gram language models as speech recognition and other applications. The language model component is monolingual, and so acquiring training data is relatively easy.

The next few sections will therefore concentrate on the other two components, the translation model and the decoding algorithm.

25.4 $P(F|E)$: THE PHRASE-BASED TRANSLATION MODEL

PHRASE-BASED

The job of the translation model, given an English sentence E and a foreign sentence F, is to assign a probability that E generates F. While we can estimate these probabilities by thinking about how each individual word is translated, modern statistical MT is based on the intuition that a better way to compute these probabilities is by considering the behavior of **phrases**. As we see in Fig. 25.16, repeated from page 12, entire phrases often need to be translated and moved as a unit. The intuition of **phrase-based** statistical MT is to use phrases (sequences of words) as well as single words as the fundamental units of translation.

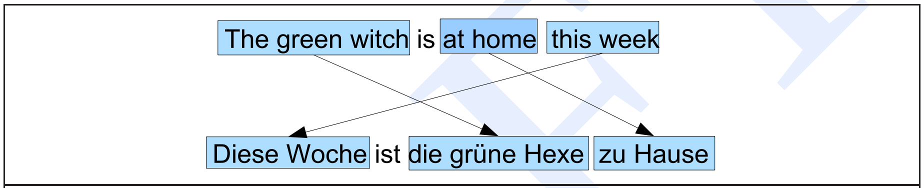


Figure 25.16 Phrasal reorderings necessary when generating German from English; repeated from Fig. 25.8.

There are a wide variety of phrase-based models; in this section we will sketch the model of Koehn et al. (2003). We'll use a Spanish example, seeing how the phrase-based model computes the probability P(*Maria no dió una bofetada a la bruja verde*|*Mary did not slap the green witch*).

The generative story of phrase-based translation has three steps. First we group the English source words into phrases $\bar{e}_1, \bar{e}_2 ... \bar{e}_I$. Next we translate each English phrase $\bar{e}_i$ into a Spanish phrase $\bar{f}_j$. Finally each of the Spanish phrases is (optionally) reordered.

DISTORTION

The probability model for phrase-based translation relies on a **translation probability** and a **distortion probability**. The factor $\phi(\bar{f}_j|\bar{e}_i)$ is the translation probability of generating Spanish phrase $\bar{f}_j$ from English phrase $\bar{e}_i$. The reordering of the Spanish phrases is done by the **distortion** probability d. Distortion in statistical machine translation refers to a word having a different ('distorted') position in the Spanish sentence than it had in the English sentence; it is thus a measure of the **distance** between the positions of a phrase in the two languages. The distortion probability in phrase-based MT means the probability of two consecutive English phrases being separated in Spanish by a span (of Spanish words) of a particular length. More formally, the distortion is parameterized by $d(a_i - b_{i-1})$, where a_i is the start position of the foreign (Spanish) phrase generated by the ith English phrase $\bar{e}_i$, and b_{i-1} is the end position of the foreign (Spanish) phrase generated by the $i-1$th English phrase $\bar{e}_{i-1}$. We can use a very simple distortion probability, in which we simply raise some small constant α to the distortion. $d(a_i - b_{i-1}) = \alpha^{|a_i - b_{i-1} - 1|}$. This distortion model penalizes large distortions by giving lower and lower probability the larger the distortion.

The final translation model for phrase-based MT is:

$$P(F|E) = \prod_{i=1}^{I} \phi(\bar{f_i}, \bar{e_i}) d(a_i - b_{i-1}) \tag{25.15}$$

Let's consider the following particular set of phrases for our example sentences:[2]

Position	**1**	**2**	**3**	**4**	**5**
English	Mary	did not	slap	the	green witch
Spanish	Maria	no	dió una bofetada	a la	bruja verde

Since each phrase follows directly in order (nothing moves around in this example, unlike the German example in (25.16)) the distortions are all 1, and the probability $P(F|E)$ can be computed as:

$$\begin{aligned} P(F|E) = & P(\text{Maria}, \text{Mary}) \times d(1) \times P(\text{no}|\text{did not}) \times d(1) \times \\ & P(\text{dió una bofetada}|\text{slap}) \times d(1) \times P(\text{a la}|\text{the}) \times d(1) \times \\ & P(\text{bruja verde}|\text{green witch}) \times d(1) \end{aligned} \tag{25.16}$$

In order to use the phrase-based model, we need two more things. We need a model of **decoding**, so we can go from a surface Spanish string to a hidden English string. And we need a model of **training**, so we can learn parameters. We'll introduce the decoding algorithm in Sec. 25.8. Let's turn first to training.

How do we learn the simple phrase-based translation probability model in (25.15)? The main set of parameters that needs to be trained is the set of phrase translation probabilities $\phi(\bar{f_i}, \bar{e_i})$.

These parameters, as well as the distortion constant α, could be set if only we had a large bilingual training set, in which each Spanish sentence was paired with an English sentence, and if furthermore we knew exactly which phrase in the Spanish sentence was translated by which phrase in the English sentence. We call such a mapping a **phrase alignment**.

PHRASE ALIGNMENT

The table of phrases above showed an implicit alignment of the phrases for this sentence, for example *green witch* aligned with *bruja verde*. If we had a large training set with each pair of sentences labeled with such a phrase alignment, we could just count the number of times each phrase-pair occurred, and normalize to get probabilities:

$$\phi(\bar{f}, \bar{e}) = \frac{\text{count}(\bar{f}, \bar{e})}{\sum_{\bar{f}} \text{count}(\bar{f}, \bar{e})} \tag{25.17}$$

We could store each phrase pair $(\bar{f}, \bar{e})$, together with its probability $\phi(\bar{f}, \bar{e})$, in a large **phrase translation table**.

PHRASE TRANSLATION TABLE

Alas, we don't have large hand-labeled phrase-aligned training sets. But it turns that we can extract phrases from another kind of alignment called a **word alignment**. A word alignment is different than a phrase alignment, because it shows exactly which

WORD ALIGNMENT

[2] Exactly which phrases we use depends on which phrases are discovered in the training process, as described in Sec. 25.7; thus for example if we don't see the phrase *green witch* in our training data, we would have to translate *green* and *witch* independently.

Spanish word aligns to which English word inside each phrase. We can visualize a word alignment in various ways. Fig. 25.17 and Fig. 25.18 show a graphical model and an alignment matrix, respectively, for a word alignment.

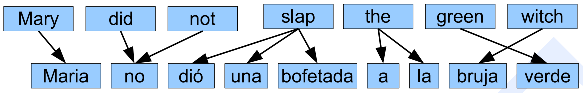
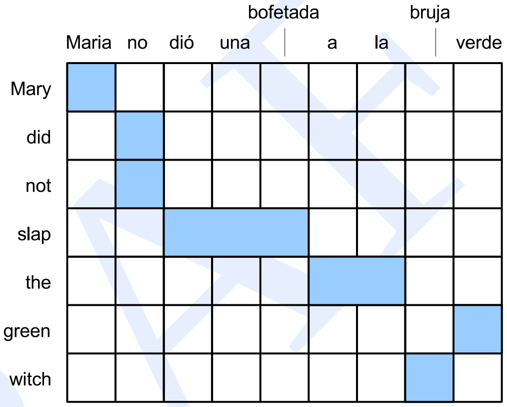


Figure 25.17 A graphical model representation of a word alignment between the English and Spanish sentences. We will see later how to extract phrases.



	Maria	no	dió	una	bofetada	a	la	bruja	verde
Mary	■								
did		■							
not		■							
slap			■	■	■				
the						■	■		
green									■
witch								■	

Figure 25.18 An alignment matrix representation of a word alignment between the English and Spanish sentences. We will see later how to extract phrases.

The next section introduces a few algorithms for deriving word alignments. We then show in Sec. 25.7 how we can extract a phrase table from word alignments, and finally in Sec. 25.8 how the phrase table can be used in decoding.

25.5 ALIGNMENT IN MT

WORD ALIGNMENT

All statistical translation models are based on the idea of a **word alignment**. A word alignment is a mapping between the source words and the target words in a set of parallel sentences.

Fig. 25.19 shows a visualization of an alignment between the English sentence *And the program has been implemented* and the French sentence *Le programme a été mis en application*. For now, we assume that we already know which sentences in the English text aligns with which sentences in the French text.

In principle, we can have arbitrary alignment relationships between the English and French word. But the word alignment models we will present (IBM Models 1

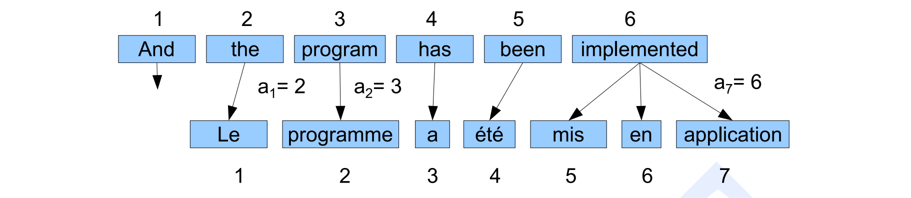


Figure 25.19 An alignment between an English and a French sentence, after Brown et al. (1993). Each French word aligns to a single English word.

and 3 and the HMM model) make a more stringent requirement, which is that each French word comes from exactly one English word; this is consistent with Fig. 25.19. One advantage of this assumption is that we can represent an alignment by giving the index number of the English word that the French word comes from. We can thus represent the alignment shown in Fig. 25.19 as $A = 2,3,4,5,6,6,6$. This is a very likely alignment. A very unlikely alignment, by contrast, might be $A = 3,3,3,3,3,3,3$.

We will make one addition to this basic alignment idea, which is to allow words to appear in the foreign sentence that don't align to any word in the English sentence. We model these words by assuming the existence of a NULL English word e_0 at position 0. Words in the foreign sentence that are not in the English sentence, called **spurious words**, may be generated by e_0. Fig. 25.20 shows the alignment of spurious Spanish *a* to English NULL.[3]

SPURIOUS WORDS

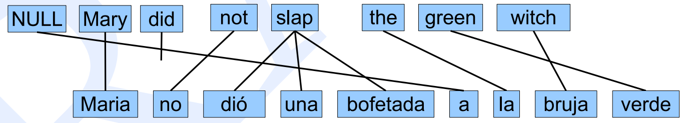


Figure 25.20 The alignment of the **spurious** Spanish word *a* to the English NULL word e_0.

While the simplified model of alignment above disallows many-to-one or many-to-many alignments, we will discuss more powerful translation models that allow such alignments. Here are two such sample alignments; in Fig. 25.21 we see an alignment which is many-to-one; each French word does not align to a single English word, although each English word does align to a single French word.

Fig. 25.22 shows an even more complex example, in which multiple English words *don't have any money* jointly align to the French words *sont démunis*. Such **phrasal alignments** will be necessary for phrasal MT, but it turns out they can't be directly generated by the IBM Model 1, Model 3, or HMM word alignment algorithms.

[3] While this particular *a* might instead be aligned to English *slap*, there are many cases of spurious words which have no other possible alignment site.

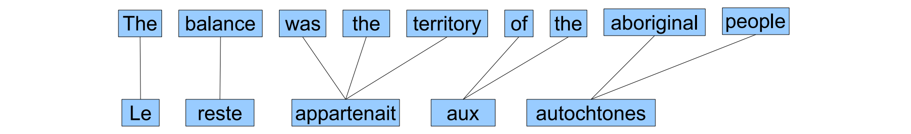


Figure 25.21 An alignment between an English and a French sentence, in which each French word does not align to a single English word, but each English word aligns to one French word. Adapted from Brown et al. (1993).

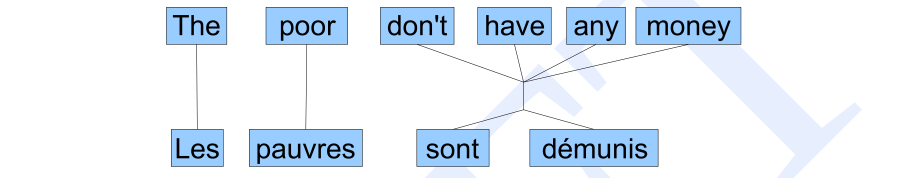


Figure 25.22 An alignment between an English and a French sentence, in which there is a many-to-many alignment between English and French words. Adapted from Brown et al. (1993).

25.5.1 IBM Model 1

We'll describe two alignment models in this section: IBM Model 1 and the HMM model (we'll also sketch the fertility-based IBM Model 3 in the advanced section). Both are **statistical alignment** algorithms. For phrase-based statistical MT, we use the alignment algorithms just to find the best alignment for a sentence pair (F,E), in order to help extract a set of phrases. But it is also possible to use these word alignment algorithms as a translation model $P(F,E)$ as well. As we will see, the relationship between alignment and translation can be expressed as follows:

$$P(F|E) \;=\; \sum_A P(F,A|E)$$

We'll start with IBM Model 1, so-called because it is the first and simplest of five models proposed by IBM researchers in a seminal paper (Brown et al., 1993).

Here's the general IBM Model 1 generative story for how we generate a Spanish sentence from an English sentence $E = e_1, e_2, ..., e_I$ of length I:

1. Choose a length K for the Spanish sentence, henceforth $F = f_1, f_2, ..., f_K$.
2. Now choose an alignment $A = a_1, a_2, ..., a_J$ between the English and Spanish sentences.
3. Now for each position j in the Spanish sentence, chose a Spanish word f_j by translating the English word that is aligned to it.

Fig. 25.23 shows a visualization of this generative process.

Let's see how this generative story assigns a probability $P(F|E)$ of generating the Spanish sentence F from the English sentence E. We'll use this terminology:

- e_{a_j} is the English word that is aligned to the Spanish word f_j.

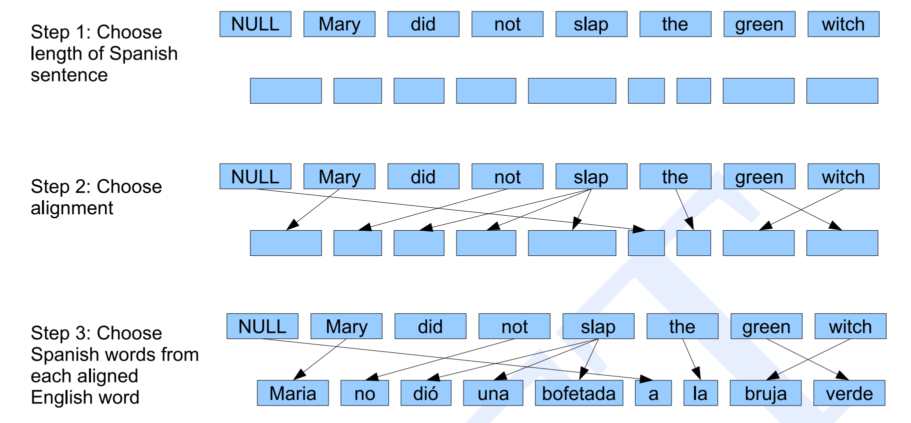


Figure 25.23 The three steps of IBM Model 1 generating a Spanish sentence and alignment from an English sentence.

- $t(f_x, e_y)$ is the probability of translating e_y by f_x (i.e. $P(f_x|e_y)$

We'll work our way backwards from step 3. So suppose we already knew the length J and the alignment A, as well as the English source E. The probability of the Spanish sentence would be:

(25.18)
$$P(F|E,A) = \prod_{j=1}^{J} t(f_j|e_{a_j})$$

Now let's formalize steps 1 and 2 of the generative story. This is the probability $P(A|E)$ of an alignment A (of length J) given the English sentence E. IBM Model 1 makes the (very) simplifying assumption that each alignment is equally likely. How many possible alignments are there between an English sentence of length I and a Spanish sentence of length J? Again assuming that each Spanish word must come from one of the I English words (or the 1 NULL word), there are $(I+1)^J$ possible alignments. Model 1 also assumes that the probability of choosing length J is some small constant ε. The combined probability of choosing a length J and then choosing any particular one of the $(I+1)^J$ possible alignments is:

(25.19)
$$P(A|E) = \frac{\varepsilon}{(I+1)^J}$$

We can combine these probabilities as follows:

(25.20)
$$\begin{aligned} P(F,A|E) &= P(F|E,A) \times P(A|E) \\ &= \frac{\varepsilon}{(I+1)^J} \prod_{j=1}^{J} t(f_j|e_{a_j}) \end{aligned}$$

This probability, $P(F,A|E)$, is the probability of generating a Spanish sentence F via a particular alignment. In order to compute the total probability $P(F|E)$ of generating F , we just sum over all possible alignments:

$$
\begin{aligned}
P(F|E) &= \sum_{A} P(F,A|E) \\
&= \sum_{A} \frac{\varepsilon}{(I+1)^J} \prod_{j=1}^{J} t(f_j|e_{a_j}) \qquad (25.21)
\end{aligned}
$$

Equation (25.21) shows the generative probability model for Model 1, as it assigns a probability to each possible Spanish sentence.

In order to find the best alignment between a pair of sentences F and E, we need a way to **decode** using this probabilistic model. It turns out there is a very simple polynomial algorithm for computing the best (Viterbi) alignment with Model 1, because the best alignment for each word is independent of the decision about best alignments of the surrounding words:

$$
\begin{aligned}
\hat{A} &= \underset{A}{\operatorname{argmax}} P(F,A|E) \\
&= \underset{A}{\operatorname{argmax}} \frac{\varepsilon}{(I+1)^J} \prod_{j=1}^{J} t(f_j|e_{a_j}) \\
&= \underset{a_j}{\operatorname{argmax}} t(f_j|e_{a_j}) \qquad 1 < j < J \qquad (25.22)
\end{aligned}
$$

Training for Model 1 is done by the EM algorithm, which we will cover in Sec. 25.6.

25.5.2 HMM Alignment

Now that we've seen Model 1, it should be clear that it makes some really appalling simplifying assumptions. One of the most egregious is the assumption that all alignments are equally likely. One way in which this is a bad assumption is that alignments tend to preserve **locality**; neighboring words in English are often aligned with neighboring words in Spanish. If we look back at the Spanish/English alignment in Fig. 25.17, for example, we can see that this locality in the neighboring alignments. The HMM alignment model captures this kind of locality by conditioning each alignment decision on previous decisions. Let's see how this works.

The HMM alignment model is based on the familiar HMM model we've now seen in many chapters. As with IBM Model 1, we are trying to compute $P(F,A|E)$. The HMM model is based on a restructuring of this probability using the chain rule as follows:

$$
\begin{aligned}
P(f_1^J, a_1^J | e_1^I) &= P(J|e_1^I) \times \prod_{j=1}^{J} P(f_j, a_j | f_1^{j-1}, a_1^{j-1}, e_1^I) \\
&= P(J|e_1^I) \times \prod_{j-1}^{J} P(a_j | f_1^{j-1}, a_1^{j-1}, e_1^I) \times P(f_j | f_1^{j-1}, a_1^j, e_1^I) \qquad (25.23)
\end{aligned}
$$

Via this restructuring, we can think of $P(F,A|E)$ as being computable from probabilities of three types: a length probability $P(J|e_1^I)$, an alignment probability $P(a_j|f_1^{j-1},a_1^{j-1},e_1^I)$, and a lexicon probability $P(f_j|f_1^{j-1},a_1^j,e_1^I)$.

We next make some standard Markov simplifying assumptions. We'll assume that the probability of a particular alignment a_j for Spanish word j is only dependent on the previous aligned position a_{j-1}. We'll also assume that the probability of a Spanish word f_j is dependent only on the aligned English word e_{a_j}at position a_j:

$$P(a_j|f_1^{j-1},a_1^{j-1},e_1^I) = P(a_j|a_{j-1},I) \tag{25.24}$$

$$P(f_j|f_1^{j-1},a_1^j,e_1^I) = P(f_j|e_{a_j}) \tag{25.25}$$

Finally, we'll assume that the length probability can be approximated just as $P(J|I)$.

Thus the probabilistic model for HMM alignment is:

$$P(f_1^J,a_1^J|e_1^I) = P(J|I)\times\prod_{j=1}^{J}P(a_j|a_{j-1},I)P(f_j|e_{a_j}) \tag{25.26}$$

To get the total probability of the Spanish sentence $P(f_1^J|e_1^I)$ we need to sum over all alignments:

$$P(f_1^J|e_1^I) = P(J|I)\times\sum_A\prod_{j=1}^{J}P(a_j|a_{j-1},I)P(f_j|e_{a_j}) \tag{25.27}$$

As we suggested at the beginning of the section, we've conditioned the alignment probability $P(a_j|a_{j-1},I)$ on the previous aligned word, to capture the locality of alignments. Let's rephrase this probability for a moment as $P(i|i',I)$, where i will stand for the absolute positions in the English sentence of consecutive aligned states in the Spanish sentence. We'd like to make these probabilities dependent not on the absolute word positions i and i', but rather on the **jump width** between words; the jump width is the distance between their positions $i'-i$. This is because our goal is to capture the fact that *'the English words that generate neighboring Spanish words are likely to be nearby'*. We thus don't want to be keeping separate probabilities for each absolute word position like $P(7|6,15)$ and $P(8|7,15)$. Instead, we compute alignment probabilities by using a non-negative function of the jump width:

JUMP WIDTH

$$P(i|i',I) = \frac{c(i-i')}{\sum_{i''=1}^{I}c(i''-i')} \tag{25.28}$$

Let's see how this HMM model gives the probability of a particular alignment of our English-Spanish sentences; we've simplified the sentence slightly.

Thus the probability $P(F,A|E)$ for this particular alignment of our simplified sentence *Maria dió una bofetada a la bruja verde* is the product of:

$$\begin{aligned} P(F,A|E) &= P(J|I)\times P(Maria|Mary)\times P(2|1,5)\times \\ &\quad t(dió|slapped)\times P(2|2,5)\times T(una|slapped)\times P(2|2,5)\times\ldots \end{aligned} \tag{25.29}$$

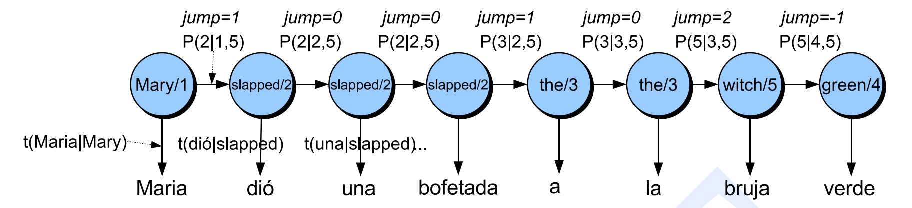


Figure 25.24 The HMM alignment model generating from *Mary slappped the green witch*, showing the alignment and lexicon components of the probability $P(F,A|E)$ for this particular alignment.

There are also more sophisticated augmentations to the basic HMM alignment model. These include adding NULL words in the English source which can be used to align with Spanish words that don't align with English words, or conditioning the alignment on $C(e_{a_{j-1}})$, the word class of the preceding target word: $P(a_j|a_{j-1},I,C(e_{a_{j-1}}))$ (Och and Ney, 2003; Toutanova et al., 2002).

The main advantage of the HMM alignment model is that there are well-understood algorithms both for decoding and for training. For decoding, we can use the Viterbi algorithm introduced in Ch. 5 and Ch. 6 to find the best (Viterbi) alignment for a sentence pair (F,E). For training, we can use the Baum-Welch algorithm, as summarized in the next section.

25.6 Training Alignment Models

PARALLEL CORPUS
BITEXT
HANSARDS
HONG KONG HANSARDS

All statistical translation models are trained using a large **parallel corpus**. A **parallel corpus**, **parallel text**, or **bitext** is a text that is available in two languages. For example, the proceedings of the Canadian parliament are kept in both French and English. Each sentence spoken in parliament is translated, producing a volume with running text in both languages. These volumes are called **Hansards**, after the publisher of the British parliamentary proceedings. Similarly, the **Hong Kong Hansards** corpus contains the proceedings of the Hong Kong SAR Legislative Council in both English and Chinese. Both of these corpora contain tens to hundreds of millions of words. Other parallel corpora have been made available by the United Nations. It is possible to make parallel corpora out of literary translations, but this is less common for MT purposes, partly because it is difficult to acquire the legal rights to fiction, but mainly because, as we saw at the beginning of the chapter, translating fiction is very difficult and translations are not very literal. Thus statistical systems tend to be trained on very literal translations such as Hansards.

SENTENCE SEGMENTATION
SENTENCE ALIGNMENT

The first step in training is to segment the corpus into sentences. This task is called **sentence segmentation** or **sentence alignment**. The simplest methods align sentences based purely on their length in words or characters, without looking at the contents of the words in the sentences. The intuition is that if we see a long sentence in roughly the same position in each language of the parallel text, we might suspect these sentences are translations. This intuition can be implemented by a dynamic programming algorithm. More sophisticated algorithms also make use of information about word alignments. Sentence alignment algorithms are run on a parallel corpus before training MT models. Sentences which don't align to anything are thrown out, and the remaining aligned sentences can be used as a training set. See the end of the chapter for pointers to more details on sentence segmentation.

Once we have done sentence alignment, the input to our training algorithm is a corpus consisting of S sentence pairs $\{(F_s,E_s) : s = 1 \ldots S\}$. For each sentence pair (F_s,E_s) the goal is to learn an alignment $A = a_1^J$ and the component probabilities (t for Model 1, and the lexicon and alignment probabilities for the HMM model).

25.6.1 EM for Training Alignment Models

If each sentence pair (F_s,E_s) was already hand-labeled with a perfect alignment, learning the Model 1 or HMM parameters would be trivial. For example, to get a maximum likelihood estimates in Model 1 for the translation probability *t*(*verde*,*green*), we would just count the number of times *green* is aligned to *verde*, and normalize by the total count of *green*.

But of course we don't know the alignments in advance; all we have are the **probabilities** of each alignment. Recall that Eq˙ 25.20 showed that if we already had good estimates for the Model 1 t parameter, we could use this to compute probabilities $P(F,A|E)$ for alignments. Given $P(F,A|E)$, we can generate the probability of an alignment just by normalizing:

$$P(A|E,F) = \frac{P(A,F|E)}{\sum_A P(A,F|E)}$$

So, if we had a rough estimate of the Model 1 t parameters, we could compute the probability for each alignment. Then instead of estimating the t probabilities from the (unknown) perfect alignment, we would estimate them from each possible alignment, and combine these estimates weighted by the probability of each alignment. For example if there were two possible alignments, one of probability .9 and one of probability .1, we would estimate the t parameters separately from the two alignments and mix these two estimates with weights of .9 and .1.

Thus if we had model 1 parameters already, we could **re-estimate** the parameters, by using the parameters to compute the probability of each possible alignment, and then using the weighted sum of alignments to re-estimate the model 1 parameters. This idea of iteratively improving our estimates of probabilities is a special case of the **EM algorithm** that we introduced in Ch. 6, and that we saw again for speech recognition in Ch. 9. Recall that we use the EM algorithm when we have a variable that we can't optimize directly because it is **hidden**. In this case the hidden variable is the alignment. But we can use the EM algorithm to estimate the parameters, compute alignments from these estimates, use the alignments to re-estimate the parameters, and so on!

Let's walk through an example inspired by Knight (1999b), using a simplified version of Model 1, in which we ignore the NULL word, and we only consider a subset of the alignments (ignoring alignments for which an English word aligns with no Spanish word). Hence we compute the simplified probability $P(A,F|E)$ as follows:

$$P(A,F|E) = \prod_{j=1}^{J} t(f_j|e_{a_j}) \tag{25.30}$$

The goal of this example is just to give an intuition of EM applied to this task; the actual details of Model 1 training would be somewhat different.

The intuition of EM training is that in the E-step, we compute **expected counts** for the t parameter based on summing over the hidden variable (the alignment), while in the M-step, we compute the maximum likelihood estimate of the t probability from these counts.

Let's see a few stages of EM training of this parameter on a a corpus of two sentences:

```
green house        the house
casa  verde        la  casa
```

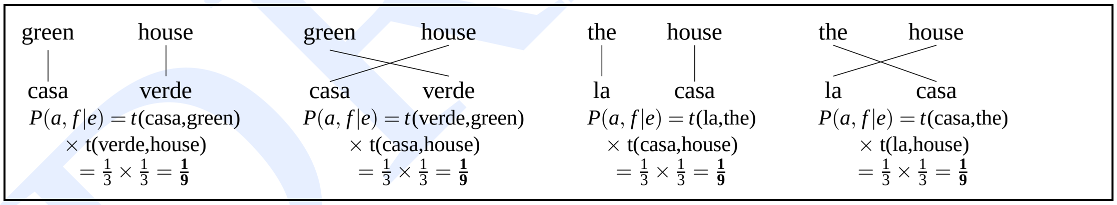
The vocabularies for the two languages are $E = \{$green,house,the$\}$ and $S = \{$casa,la,verde$\}$. We'll start with uniform probabilities:

t(casa|green) $= \frac{1}{3}$	t(verde|green) $= \frac{1}{3}$	t(la|green) $= \frac{1}{3}$
t(casa|house) $= \frac{1}{3}$	t(verde|house) $= \frac{1}{3}$	t(la|house) $= \frac{1}{3}$
t(casa|the) $= \frac{1}{3}$	t(verde|the) $= \frac{1}{3}$	t(la|the) $= \frac{1}{3}$

Now let's walk through the steps of EM:

E-step 1: Compute the expected counts $E[\text{count}(t(f,e))]$ for all word pairs (f_j, e_{a_j})

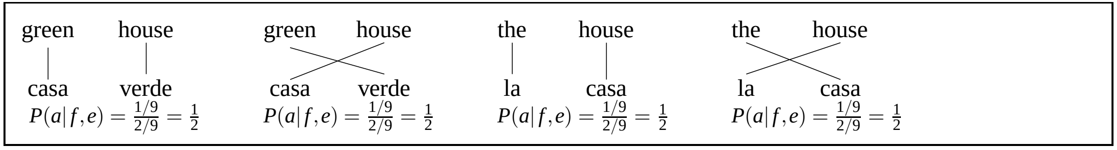
E-step 1a: We first need to compute $P(a,f|e)$, by multiplying all the t probabilities, following Eq. 25.30


green house
casa verde
$P(a,f|e) = t(\text{casa,green})$
$\times$ t(verde,house)
$= \frac{1}{3} \times \frac{1}{3} = \mathbf{\frac{1}{9}}$
green house
casa verde
$P(a,f|e) = t(\text{verde,green})$
$\times$ t(casa,house)
$= \frac{1}{3} \times \frac{1}{3} = \mathbf{\frac{1}{9}}$
the house
la casa
$P(a,f|e) = t(\text{la,the})$
$\times$ t(casa,house)
$= \frac{1}{3} \times \frac{1}{3} = \mathbf{\frac{1}{9}}$
the house
la casa
$P(a,f|e) = t(\text{casa,the})$
$\times$ t(la,house)
$= \frac{1}{3} \times \frac{1}{3} = \mathbf{\frac{1}{9}}$

E-step 1b: Normalize $P(a,f|e)$ to get $P(a|e,f)$, using the following:

$$P(a|e,f) = \frac{P(a,f|e)}{\sum_a P(a,f|e)}$$

The resulting values of $P(a|f,e)$ for each alignment are as follows:


green house
casa verde
$P(a|f,e) = \frac{1/9}{2/9} = \frac{1}{2}$
green house
casa verde
$P(a|f,e) = \frac{1/9}{2/9} = \frac{1}{2}$
the house
la casa
$P(a|f,e) = \frac{1/9}{2/9} = \frac{1}{2}$
the house
la casa
$P(a|f,e) = \frac{1/9}{2/9} = \frac{1}{2}$

E-step 1c: Compute expected (fractional) counts, by weighting each count by $P(a|e,f)$

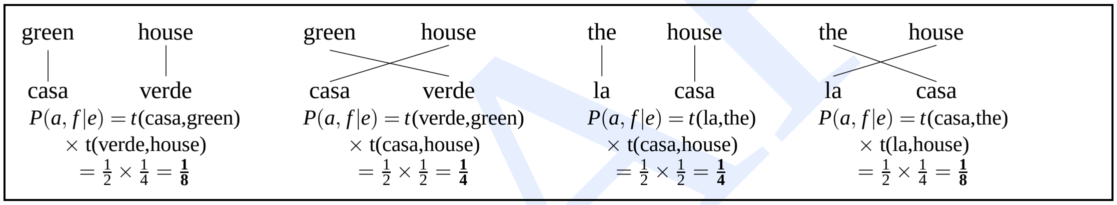
tcount(casa|green) $= \frac{1}{2}$	tcount(verde|green) $= \frac{1}{2}$	tcount(la|green) $= 0$	total(green) $= 1$
tcount(casa|house) $= \frac{1}{2} + \frac{1}{2}$	tcount(verde|house) $= \frac{1}{2}$	tcount(la|house) $= \frac{1}{2}$	total(house) $= 2$
tcount(casa|the) $= \frac{1}{2}$	tcount(verde|the) $= 0$	tcount(la|the) $= \frac{1}{2}$	total(the) $= 1$

M-step 1: Compute the MLE probability parameters by normalizing the tcounts to sum to one.

t(casa|green) $= \frac{1/2}{1} = \frac{1}{2}$	t(verde|green) $= \frac{1/2}{1} = \frac{1}{2}$	t(la|green) $= \frac{0}{1} = 0$
t(casa|house) $= \frac{1}{2} = \frac{1}{2}$	t(verde|house) $= \frac{1/2}{2} = \frac{1}{4}$	t(la|house) $= \frac{1/2}{2} = \frac{1}{4}$
t(casa|the) $= \frac{1/2}{1} = \frac{1}{2}$	t(verde|the) $= \frac{0}{1} = 0$	t(la|the) $= \frac{1/2}{1} = \frac{1}{2}$

Note that each of the correct translations have increased in probability from the initial assignment; for example the translation *casa* for *house* has increased in probability from $\frac{1}{3}$ to $\frac{1}{2}$.

E-step 2a: We re-compute $P(a, f|e)$, again by multiplying all the t probabilities, following Eq. 25.30



$P(a, f|e) = t(\text{casa,green}) \times \text{t(verde,house)} = \frac{1}{2} \times \frac{1}{4} = \mathbf{\frac{1}{8}}$

$P(a, f|e) = t(\text{verde,green}) \times \text{t(casa,house)} = \frac{1}{2} \times \frac{1}{2} = \mathbf{\frac{1}{4}}$

$P(a, f|e) = t(\text{la,the}) \times \text{t(casa,house)} = \frac{1}{2} \times \frac{1}{2} = \mathbf{\frac{1}{4}}$

$P(a, f|e) = t(\text{casa,the}) \times \text{t(la,house)} = \frac{1}{2} \times \frac{1}{4} = \mathbf{\frac{1}{8}}$

Note that the two correct alignments are now higher in probability than the two incorrect alignments. Performing the second and further round of E-steps and M-steps is left as Exercise 25.6 for the reader.

We have shown that EM can be used to learn the parameters for a simplified version of Model 1. Our intuitive algorithm, however, requires that we enumerate all possible alignments. For a long sentence, enumerating every possible alignment would be very inefficient. Luckily in practice there is a very efficient version of EM for Model 1 that efficiently and implicitly sums over all alignments.

We also use EM, in the form of the Baum-Welch algorithm, for learning the parameters of the HMM model.

25.7 SYMMETRIZING ALIGNMENTS FOR PHRASE-BASED MT

The reason why we needed Model 1 or HMM alignments was to build word alignments on the training set, so that we could extract aligned pairs of phrases.

Unfortunately, HMM (or Model 1) alignments are insufficient for extracting pairings of Spanish phrases with English phrases. This is because in the HMM model, each Spanish word must be generated from a single English word; we cannot generate a Spanish phrase from multiple English words. The HMM model thus cannot align a multiword phrase in the source language with a multiword phrase in the target language.

SYMMETRIZING

We can, however, extend the HMM model to produce phrase-to-phrase alignments for a pair of sentences (F,E), via a method that's often called **symmetrizing**. First, we train two separate HMM aligners, an English-to-Spanish aligner and a Spanish-to-English aligner. We then align (F,E) using both aligners. We can then combine these alignments in clever ways to get an alignment that maps phrases to phrases.

INTERSECTION

To combine the alignments, we start by taking the **intersection** of the two alignments, as shown in Fig. 25.25. The intersection will contain only places where the two alignments agree, hence the high-precision aligned words. We can also separately compute the **union** of these two alignments. The union will have lots of less accurately aligned words. We can then build a classifier to select words from the union, which we incrementally add back in to this minimal intersective alignment.

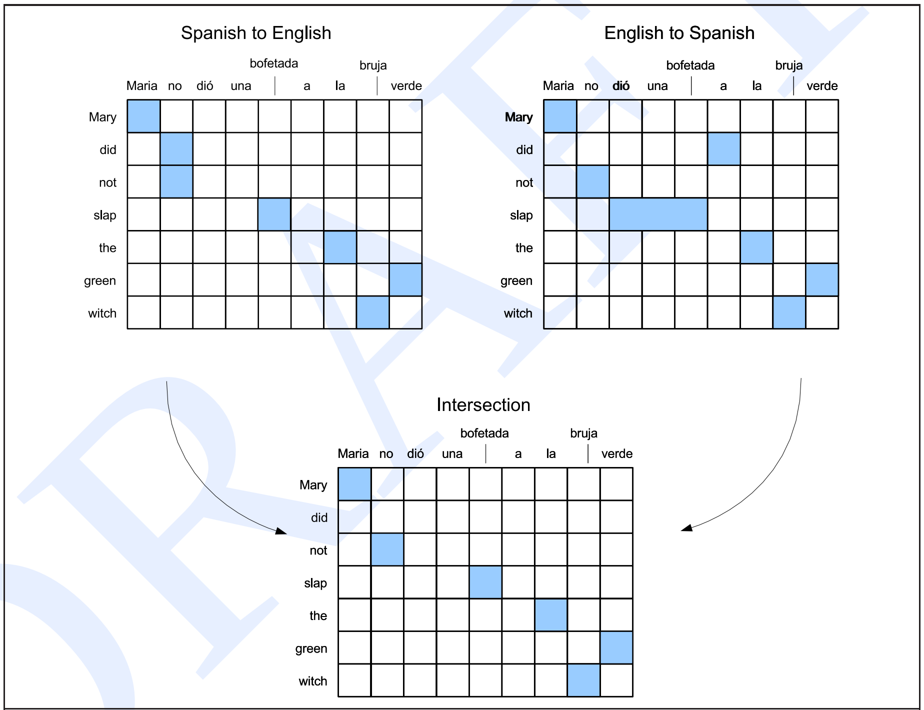
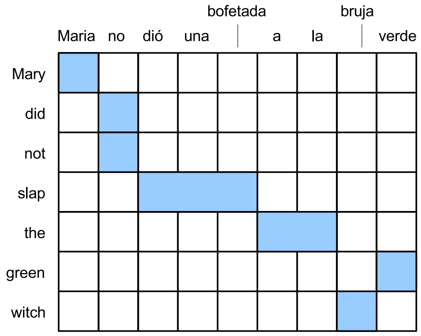


Figure 25.25 Intersection of English-to-Spanish and Spanish-to-English alignments to produce a high-precision alignment. Alignment can then be expanded with points from both alignments to produce an alignment like that shown in Fig. 25.26. After Koehn (2003b).

Fig. 25.26 shows an example of the resulting word alignment. Note that it does allow many-to-one alignments in both directions. We can now harvest all phrase pairs that are consistent with this word alignment. A consistent phrase pair is one in which all the words are aligned only with each other, and not to any external words. Fig. 25.26 also shows some phrases consistent with the alignment.

Once we collect all the aligned phrases pairs from the entire training corpus, we



(Maria, Mary), (no, did not),
(slap, dió una bofetada), (verde, green),
(a la, the), (bruja, witch),
(Maria no, Mary did not),
(no dió una bofetada, did not slap),
(dió una bofetada a la, slap the),
(bruja verde, green witch),
(a la bruja verde, the green witch),. . .

Figure 25.26 A better phrasal alignment for the *green witch* sentence, computed by starting with the intersection alignment in Fig. 25.25 and adding points from the union alignment, using the algorithm of Och and Ney (2003). On the right, some of the phrases consistent with this alignment, after Koehn (2003b).

can compute the maximum likelihood estimate for the phrase translation probability of a particular pair as follows:

$$\phi(\bar{f},\bar{e}) = \frac{\text{count}(\bar{f},\bar{e})}{\sum_{\bar{f}}\text{count}(\bar{f},\bar{e})} \qquad (25.31)$$

PHRASE TRANSLATION TABLE

We can now store each phrase $(\bar{f},\bar{e})$, together with its probability $\phi(\bar{f},\bar{e})$, in a large **phrase translation table**. The decoding algorithm discussed in the next section can use this phrase translation table to compute the translation probability.

25.8 Decoding for Phrase-Based Statistical MT

The remaining component of a statistical MT system is the decoder. Recall that the job of the decoder is to take a foreign (Spanish) source sentence F and produce the best (English) translation E according to the product of the translation and language models:

$$\hat{E} = \underset{E\in\text{English}}{\text{argmax}} \; \overbrace{P(F|E)}^{\text{translation model}} \; \overbrace{P(E)}^{\text{language model}} \qquad (25.32)$$

Finding the sentence which maximizes the translation and language model probabilities is a **search** problem, and decoding is thus a kind of search. Decoders in MT are based on **best-first search**, a kind of **heuristic** or **informed search**; these are search algorithms that are informed by knowledge from the problem domain. Best-first search algorithms select a node n in the search space to explore based on an evaluation function $f(n)$. MT decoders are variants of a specific kind of best-first search called $\mathbf{A}^*$ search. A^* search was first implemented for machine translation by IBM (Brown et al., 1995), based on IBM's earlier work on A* search for speech recognition (Jelinek, 1969). As we discussed in Sec. **??**, for historical reasons A* search and its variants are commonly called **stack decoding** in speech recognition and sometimes also in machine translation.

STACK DECODING

Let's begin in Fig. 25.27 with a generic version of stack decoding for machine translation. The basic intuition is to maintain a **priority queue** (traditionally referred to as a **stack**) with all the partial translation hypotheses, together with their scores.

```
function STACK DECODING(source sentence) returns target sentence

initialize stack with a null hypothesis
loop do
  pop best hypothesis $h$ off of stack
  if $h$ is a complete sentence, return $h$
  for each possible expansion $h'$ of $h$
      assign a score to $h'$
      push $h'$ onto stack
```

Figure 25.27 Generic version of stack or A* decoding for machine translation. A hypothesis is expanded by choosing a single word or phrase to translate. We'll see a more fleshed-out version of the algorithm in Fig. 25.30.

Let's now describe stack decoding in more detail. While the original IBM statistical decoding algorithms were for word-based statistical MT, we will describe the application to phrase-based decoding in the publicly available MT decoder **Pharaoh** (Koehn, 2004).

In order to limit the search space in decoding, we don't want to search through the space of all English sentences; we only want to consider the ones that are possible translations for F. To help reduce the search space, we only want to consider sentences that include words or phrases which are possible translations of words or phrases in the Spanish sentence F. We do this by searching the **phrase translation table** described in the previous section, for all possible English translations for all possible phrases in F.

A sample lattice of possible translation options is shown in Fig. 25.28 drawn from Koehn (2003a, 2004). Each of these options consists of a Spanish word or phrase, the English translation, and the phrase translation probability ϕ. We'll need to search through combinations of these to find the best translation string.

Now let's walk informally through the stack decoding example in Fig. 25.29, producing an English translation of *Mary dió una bofetada a la bruja verde* left to right. For the moment we'll make the simplifying assumption that there is a single stack, and that there is no pruning.

We start with the null hypothesis as the initial **search state**, in which we have selected no Spanish words and produced no English translation words. We now **expand** this hypothesis by choosing each possible source word or phrase which could generate an English sentence-initial phrase. Fig. 25.29a shows this first ply of the search. For

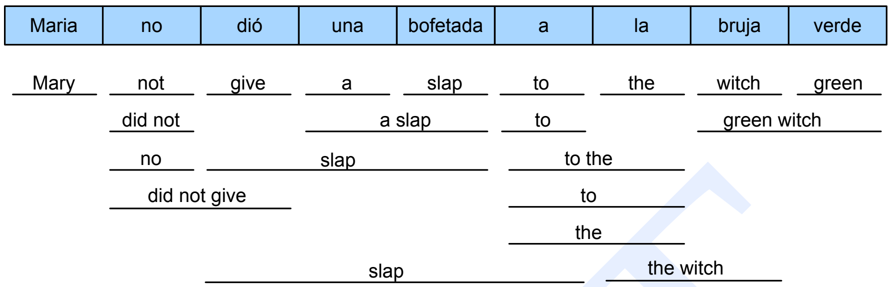


Figure 25.28 The lattice of possible English translations for words and phrases in a particular sentence F, taken from the entire aligned training set. After Koehn (2003a)

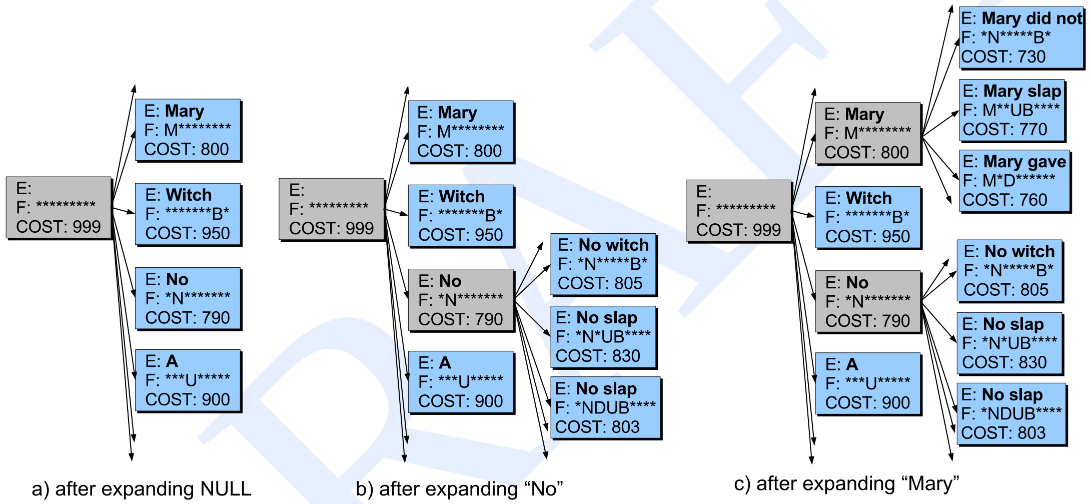


Figure 25.29 Three stages in stack decoding of *Maria no dió una bofetada a la bruja verde* (simplified by assuming a single stack and no pruning). The nodes in blue, on the fringe of the search space, are all on the stack, and are **open** nodes still involved in the search. Nodes in gray are **closed** nodes which have been popped off the stack.

example the top state represents the hypothesis that the English sentence starts with *Mary*, and the Spanish word *Maria* has been covered (the asterisk for the first word is marked with an M). Each state is also associated with a cost, discussed below. Another state at this ply represents the hypothesis that the English translation starts with the word *No*, and that Spanish *no* has been covered. This turns out to be the lowest-cost node on the queue, so we pop it off the queue and push all its expansions back on the queue. Now the state *Mary* is the lowest cost, so we expand it; *Mary did not* is now the lowest cost translation so far, so will be the next to be expanded. We can then continue to expand the search space until we have states (hypotheses) that cover the entire Spanish sentence, and we can just read off an English translation from this state.

We mentioned that each state is associated with a cost which, as we'll see below, is used to guide the search, The cost combines the **current cost** with an estimate of the **future cost**. The **current cost** is the total probability of the phrases that have been translated so far in the hypothesis, i.e. the product of the translation, distortion, and language model probabilities. For the set of partially translated phrases $S = (F, E)$, this probability would be:

$$\text{cost}(E,F) = \prod_{i \in S} \phi(\bar{f_i}, \bar{e_i}) d(a_i - b_{i-1}) P(E) \tag{25.33}$$

The **future cost** is our estimate of the cost of translating the *remaining* words in the Spanish sentence. By combining these two factors, the state cost gives an estimate of the total probability of the search path for the eventual complete translation sentence E passing through the current node. A search algorithm based only on the current cost would tend to select translations that had a few high-probability words at the beginning, at the expense of translations with a higher overall probability. [4] For the future cost, it turns out to be far too expensive to compute the true minimum probability for all possible translations. Instead, we approximate this cost by ignoring the distortion cost and just finding the sequence of English phrases which has the minimum product of the language model and translation model costs, which can be easily computed by the Viterbi algorithm.

This sketch of the decoding process suggests that we search the entire state space of possible English translations. But we can't possibly afford to expand the entire search space, because there are far too many states; unlike in speech recognition, the need for distortion in MT means there is (at least) a distinct hypothesis for every possible ordering of the English words![5]

BEAM-SEARCH PRUNING

For this reason MT decoders, like decoders for speech recognition, all require some sort of pruning. Pharaoh and similar decoders use a version of **beam-search pruning**, just as we saw in decoding for speech recognition and probabilistic parsing. Recall that in beam-search pruning, at every iteration we keep only the most promising states, and prune away unlikely (high-cost) states (those 'outside the search beam'). We could modify the search sequence depicted in Fig. 25.29, by pruning away all bad (high-cost) states at every ply of the search, and expanding only the best state. In fact, in Pharaoh, instead of expanding only the best state, we expand all states within the beam; thus Pharaoh is technically **beam search** rather than **best-first search** or A* search.

More formally, at each ply of the search we keep around a stack (priority queue) of states. The stack only fits n entries. At every ply of the search, we expand all the states on the stack, push them onto the stack, order them by cost, keep the best n entries and delete the rest.

We'll need one final modification. While in speech we just used one stack for stack decoding, in MT we'll use multiple stacks, because we can't easily compare the cost of hypotheses that translate different numbers of foreign words. So we'll use m stacks, where stack s_m includes all hypotheses that cover m foreign words. When we expand a

[4] We saw this same kind of cost function for A* search in speech recognition, where we used the A* evaluation function: $f^*(p) = g(p) + h^*(p)$.

[5] Indeed, as Knight (1999a) shows, decoding even in IBM Model 1 with a bigram language model is equivalent to the difficult class of problems known as **NP-complete**.

hypothesis by choosing a phrase to translate, we'll insert the new state into the correct stack for the number of foreign words covered. Then we'll use beam-search inside each of these stacks, keep only *n* hypotheses for each of the *m* stacks. The final multi-stack version of beam search stack decoding is shown in Fig. 25.30.

```
function BEAM SEARCH STACK DECODER(source sentence) returns target sentence

initialize hypothesisStack[0..nf]
push initial null hypothesis on hypothesisStack[0]
for $i \leftarrow 0$ to nf-1
  for each hyp in hypothesisStack[i]
    for each new_hyp that can be derived from hyp
      nf[new_hyp] ← number of foreign words covered by new_hyp
      add new_hyp to hypothesisStack[nf[new_hyp]]
      prune hypothesisStack[nf[new_hyp]]
find best hypothesis best_hyp in hypothesisStack[nf]
return best path that leads to best_hyp via backtrace
```

Figure 25.30 Pharaoh beam search multi-stack decoding algorithm, adapted from (Koehn, 2003a, 2004). For efficiency, most decoders don't store the entire foreign and English sentence in each state, requiring that we backtrace to find the state path from the initial to the final state so we can generate the entire English target sentence.

RECOMBINING HYPOTHESES

There are a number of additional issues in decoding that must be dealt with. All decoders attempt to limit somewhat the exponential explosion in the search space by **recombining hypotheses**. . We saw hypothesis recombination in the **Exact *N*-Best** algorithm of Sec. **??**. In MT, we can merge any two hypotheses that are sufficiently similar (cover the same foreign words, have the same last-two English words, and have the same end of the last foreign phrase covered).

In addition, it turns out that decoders for phrasal MT optimize a slightly different function than the one we presented in Eq. 25.32. In practice, it turns out that we need to add another factor, which serves to penalize sentences which are too short. Thus the decoder is actually choosing the sentence which maximizes:

$$\hat{E} = \operatorname*{argmax}_{E\in\text{English}} \overbrace{P(F|E)}^{\text{translation model}} \overbrace{P(E)}^{\text{language model}} \overbrace{\omega^{\text{length}(E)}}^{\text{short sentence penalty}} \tag{25.34}$$

This final equation is extremely similar to the use of the word insertion penalty in speech recognition in Eq. **??**.

25.9 MT Evaluation

Evaluating the quality of a translation is an extremely subjective task, and disagreements about evaluation methodology are rampant. Nevertheless, evaluation is essential, and research on evaluation methodology has played an important role from the earliest days of MT (Miller and Beebe-Center, 1958) to the present. Broadly speaking, we attempt to evaluate translations along two dimensions, corresponding to the **fidelity** and **fluency** discussed in Sec. 25.3.

25.9.1 Using Human Raters

The most accurate evaluations use human raters to evaluate each translation along each dimension. For example, along the dimension of **fluency**, we can ask how intelligible, how clear, how readable, or how natural is the MT output (the target translated text). There are two broad ways to use human raters to answer these questions. One method is to give the raters a scale, for example from 1 (totally unintelligible) to 5 (totally intelligible), and ask them to rate each sentence or paragraph of the MT output. We can use distinct scales for any of the aspects of fluency, such as **clarity**, **naturalness**, or **style**. The second class of methods relies less on the conscious decisions of the participants. For example, we can measure the time it takes for the raters to read each output sentence or paragraph. Clearer or more fluent sentences should be faster or easier to read. We can also measure fluency with the **cloze** task (Taylor, 1953, 1957). The cloze task is a metric used often in psychological studies of reading. The rater sees an output sentence with a word replaced by a space (for example, every 8th word might be deleted). Raters have to guess the identity of the missing word. Accuracy at the cloze task, i.e. average success of raters at guessing the missing words, generally correlates with how intelligible or natural the MT output is.

CLOZE

A similar variety of metrics can be used to judge the second dimension, **fidelity**. Two common aspects of fidelity which are measured are **adequacy** and **informativeness**. The **adequacy** of a translation is whether it contains the information that existed in the original. We measure adequacy by using raters to assign scores on a scale. If we have bilingual raters, we can give them the source sentence and a proposed target sentence, and rate, perhaps on a 5-point scale, how much of the information in the source was preserved in the target. If we only have monolingual raters, but we have a good human translation of the source text, we can give the monolingual raters the human reference translation and a target machine translation, and again rate how much information is preserved. The **informativeness** of a translation is a task-based evaluation of whether there is sufficient information in the MT output to perform some task. For example we can give raters multiple-choice questions about the content of the material in the source sentence or text. The raters answer these questions based only on the MT output. The percentage of correct answers is an informativeness score.

ADEQUACY

INFORMATIVENESS

Another set of metrics attempt to judge the overall quality of a translation, combining fluency and fidelity. For example, the typical evaluation metric for MT output to be post-edited is the **edit cost** of **post-editing** the MT output into a good translation. For example, we can measure the number of words, the amount of time, or the number of

EDIT COST

POST-EDITING

keystrokes required for a human to correct the output to an acceptable level.

25.9.2 Automatic Evaluation: Bleu

While humans produce the best evaluations of machine translation output, running a human evaluation can be very time-consuming, taking days or even weeks. It is useful to have an automatic metric that can be run relatively frequently to quickly evaluate potential system improvements. In order to have such convenience, we would be willing for the metric to be much worse than human evaluation, as long as there was some correlation with human judgments.

In fact there are a number of such heuristic methods, such as **Bleu**, **NIST**, **TER**, **Precision and Recall**, and **METEOR** (see references at the end of the chapter). The intuition of these automatic metrics derives from Miller and Beebe-Center (1958), who pointed out that a good MT output is one which is very similar to a human translation. For each of these metrics, we assume that we already have one or more human translations of the relevant sentences. Now given an MT output sentence, we compute the translation closeness between the MT output and the human sentences. An MT output is ranked as better if on average it is closer to the human translations. The metrics differ on what counts as 'translation closeness'.

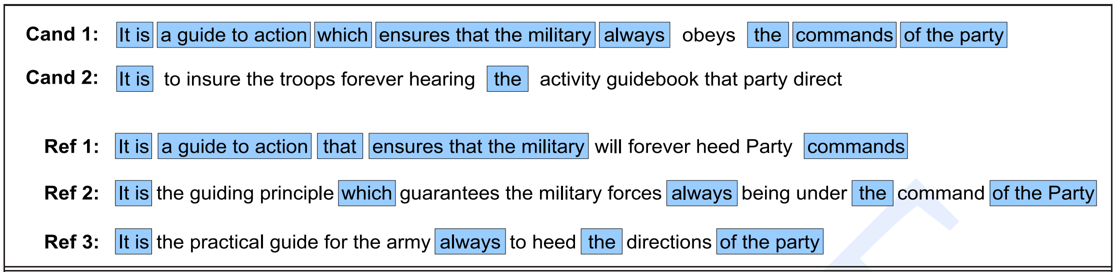
In the field of automatic speech recognition, the metric for 'transcription closeness' is word error rate, which is the minimum edit distance to a human transcript. But in translation, we can't use the same word error rate metric, because there are many possible translations of a source sentence; a very good MT output might look like one human translation, but very unlike another one. For this reason, most of the metrics judge an MT output by comparing it to multiple human translations.

Each of these metrics thus require that we get human translations in advance for a number of test sentences. This may seem time-consuming, but the hope is that we can reuse this translated test set over and over again to evaluate new ideas.

For the rest of this section, let's walk through one of these metrics, the **Bleu** metric, following closely the original presentation in Papineni et al. (2002). In Bleu we rank each MT output by a weighted average of the number of N-gram overlaps with the human translations.

Fig. 25.31 shows an intuition, from two candidate translations of a Chinese source sentence (Papineni et al., 2002), shown with three reference human translations of the source sentence. Note that Candidate 1 shares many more words (shown in blue) with the reference translations than Candidate 2.

Let's look at how the Bleu score is computed, starting with just unigrams. Bleu is based on precision. A basic unigram precision metric would be to count the number of words in the candidate translation (MT output) that occur in some reference translation, and divide by the total number of words in the candidate translation. If a candidate translation had 10 words, and 6 of them occurred in at least one of the reference translations, we would have a precision of $6/10 = 0.6$. Alas, there is a flaw in using simple precision: it rewards candidates that have extra repeated words. Fig. 25.32 shows an example of a pathological candidate sentence composed of multiple instances of the single word *the*. Since each of the 7 (identical) words in the candidate occur in one of the reference translations, the unigram precision would be 7/7!



Cand 1: It is a guide to action which ensures that the military always obeys the commands of the party

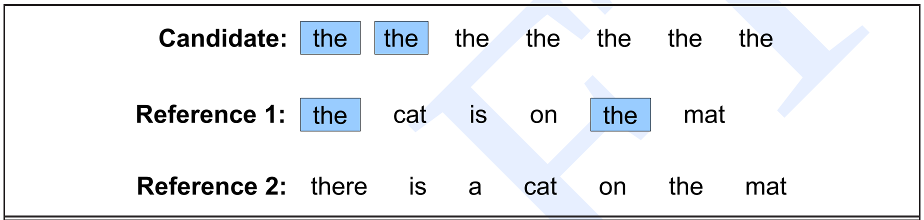
Cand 2: It is to insure the troops forever hearing the activity guidebook that party direct

Ref 1: It is a guide to action that ensures that the military will forever heed Party commands

Ref 2: It is the guiding principle which guarantees the military forces always being under the command of the Party

Ref 3: It is the practical guide for the army always to heed the directions of the party

Figure 25.31 Intuition for Bleu: one of two candidate translations of a Chinese source sentence shares more words with the reference human translations.



Candidate: the the the the the the the

Reference 1: the cat is on the mat

Reference 2: there is a cat on the mat

Figure 25.32 A pathological example showing why Bleu uses a modified precision metric. Unigram precision would be unreasonably high (7/7). Modified unigram precision is appropriately low (2/7).

MODIFIED N-GRAM PRECISION

In order to avoid this problem, Bleu uses a **modified N-gram precision** metric. We first count the maximum number of times a word is used in any single reference translation. The count of each *candidate* word is then clipped by this maximum *reference* count. Thus the modified unigram precision in the example in Fig. 25.32 would be 2/7, since Reference 1 has a maximum of 2 *the*s. Going back to Chinese example in Fig. 25.32, Candidate 1 has a modified unigram precision of 17/18, while Candidate 2 has one of 8/14.

We compute the modified precision similarly for higher order N-grams as well. The modified bigram precision for Candidate 1 is 10/17, and for Candidate 2 is 1/13. The reader should check these numbers for themselves on Fig. 25.31.

To compute a score over the whole testset, Bleu first computes the N-gram matches for each sentence, and add together the clipped counts over all the candidates sentences, and divide by the total number of candidate N-grams in the testset. The modified precision score is thus:

$$p_n = \frac{\sum_{C\in\{Candidates\}} \sum_{n\text{-}gram\in C} \text{Count}_{\text{clip}}(n\text{-}gram)}{\sum_{C'\in\{Candidates\}} \sum_{n\text{-}gram'\in C'} \text{Count}(n\text{-}gram')} \quad (25.35)$$

Bleu uses unigram, bigrams, trigrams, and often quadrigrams; it combines these modified N-gram precisions together by taking their geometric mean.

In addition, Bleu adds a further penalty to penalize candidate translations that are too short. Consider the candidate translation *of the*, compared with References 1-3 in Fig. 25.31 above. Because this candidate is so short, and all its words appear in some translation, its modified unigram precision is inflated to 2/2. Normally we deal with these problems by combining precision with *recall*. But as we discussed above, we can't use recall over multiple human translations, since recall would require (incorrectly) that a good translation must contain contains lots of N-grams from *every* translation. Instead, Bleu includes a brevity penalty over the whole corpus. Let c be the total length of the candidate translation corpus. We compute the **effective reference length** r for that corpus by summing, for each candidate sentence, the lengths of the best matches. The brevity penalty is then an exponential in r/c. In summary:

$$\begin{aligned} BP &= \begin{cases} 1 & \text{if} \quad c > r \\ e^{(1-r/c)} & \text{if} \quad c \leq r \end{cases} \\ \text{Bleu} &= \text{BP} \times \exp\left(\frac{1}{N}\sum_{n=1}^{N} \log p_n\right) \end{aligned} \tag{25.36}$$

While automatic metrics like Bleu (or NIST, METEOR, etc) have been very useful in quickly evaluating potential system improvements, and match human judgments in many cases, they have certain limitations that are important to consider. First, many of them focus on very local information. Consider slightly moving a phrase in Fig. 25.31 slightly to produce a candidate like: *Ensures that the military it is a guide to action which always obeys the commands of the party*. This sentence would have an identical Bleu score to Candidate 1, although a human rater would give it a lower score.

Furthermore, the automatic metrics probably do poorly at comparing systems that have radically different architectures. Thus Bleu, for example, is known to perform poorly (i.e. not agree with human judgments of translation quality) when evaluating the output of commercial systems like Systran against N-gram-based statistical systems, or even when evaluating human-aided translation against machine translation (Callison-Burch et al., 2006).

We can conclude that automatic metrics are most appropriate when evaluating incremental changes to a single system, or comparing systems with very similar architectures.

25.10 Advanced: Syntactic Models for MT

The earliest statistical MT systems (like IBM Models 1, 2 and 3) were based on words as the elementary units. The phrase-based systems that we described in earlier sections improved on these word-based systems by using larger units, thus capturing larger contexts and providing a more natural unit for representing language divergences.

Recent work in MT has focused on ways to move even further up the Vauquois hierarchy, from simple phrases to larger and hierarchical syntactic structures.

It turns out that it doesn't work just to constrain each phrase to match the syntactic boundaries assigned by traditional parsers (Yamada and Knight, 2001). Instead, modern approaches attempt to assign a parallel syntactic tree structure to a pair of sentences in different languages, with the goal of translating the sentences by applying reordering operations on the trees. The mathematical model for these parallel structures is known as a **transduction grammar**. These transduction grammars can be viewed as an explicit implementation of the **syntactic transfer** systems that we introduced on page 14, but based on a modern statistical foundation.

TRANSDUCTION GRAMMAR

SYNCHRONOUS GRAMMAR

A transduction grammar (also called a **synchronous grammar**) describes a structurally correlated pair of languages. From a generative perspective, we can view a transduction grammar as generating pairs of aligned sentences in two languages. Formally, a transduction grammar is a generalization of the finite-state transducers we saw in Ch. 3. There are a number of transduction grammars and formalisms used for MT, most of which are generalizations of context-free grammars to the two-language situation. Let's consider one of the most widely used such models for MT, the **inversion transduction grammar** (ITG).

INVERSION TRANSDUCTION GRAMMAR

In an ITG grammar, each non-terminal generates two separate strings. There are three types of these rules. A lexical rule like the following:

$$N \rightarrow witch/bruja$$

generates the word *witch* on one stream, and *bruja* on the second stream. A nonterminal rule in square brackets like:

$$S \rightarrow [NP\ VP]$$

generates two separate streams, each of *NP VP*. A non-terminal in angle brackets, like

$$Nominal \rightarrow \langle Adj\ N \rangle$$

generates two separate streams, with *different orderings*: *Adj N* in one stream, and *N Adj* in the other stream.

Fig. 25.33 shows a sample grammar with some simple rules. Note that each lexical rule derives distinct English and Spanish word strings, that rules in square brackets ([]) generate two identical non-terminal right-hand sides, and that the one rule in angle brackets (⟨⟩) generates different orderings in Spanish from English.

Thus an ITG parse tree is a single joint structure which spans over the two observed sentences:

(25.37)
(a) [$_{S}$ [$_{NP}$ Mary] [$_{VP}$ didn't [$_{VP}$ slap [$_{PP}$ [$_{NP}$ the [$_{Nom}$ green witch]]]]]]
(b) [$_{S}$ [$_{NP}$ María] [$_{VP}$ no [$_{VP}$ dió una bofetada [$_{PP}$ a [$_{NP}$ la [$_{Nom}$ bruja verde]]]]]]

Each non-terminal in the parse derives two strings, one for each language. Thus we could visualize the two sentences in a single parse, where the angle brackets mean that the order of the *Adj N* constituents *green witch* and *bruja verde* are generated in opposite order in the two languages:

[$_{S}$ [$_{NP}$ Mary/María] [$_{VP}$ didn't/no [$_{VP}$ slap/dió una bofetada [$_{PP}$ ε/a [$_{NP}$ the/la ⟨$_{Nom}$ witch/bruja green/verde⟩]]]]]

There are a number of related kinds of synchronous grammars, including synchronous context-free grammars (Chiang, 2005), multitext grammars (Melamed, 2003), lexicalized ITGs (Melamed, 2003; Zhang and Gildea, 2005), and synchronous tree-adjoining and tree-insertion grammars (Shieber and Schabes, 1992; Shieber, 1994;

$$
\begin{array}{rcl}
S & \rightarrow & [NP\ VP] \\
NP & \rightarrow & [Det\ Nominal] \quad |\ Maria/María \\
Nominal & \rightarrow & \langle Adj\ Noun \rangle \\
VP & \rightarrow & [V\ PP] \quad |\ [Negation\ VP] \\
Negation & \rightarrow & didn't/no \\
V & \rightarrow & slap/dió\ una\ bofetada \\
PP & \rightarrow & [P\ NP] \\
P & \rightarrow & \varepsilon/a \quad |\ from/de \\
Det & \rightarrow & the/la \quad |\ the/le \\
Adj & \rightarrow & green/verde \\
N & \rightarrow & witch/bruja
\end{array}
$$

Figure 25.33 A mini Inversion Transduction Grammar grammar for the *green witch* sentence.

Nesson et al., 2006). The synchronous CFG system of Chiang (2005), for example, learns hierarchical pairs of rules that capture the fact that Chinese relative clauses appear to the left of their head, while English relative clauses appear to the right of their head:

<① de ②, the ② that ①>

Other models for translation by aligning parallel parse trees including (Wu, 2000; Yamada and Knight, 2001; Eisner, 2003; Melamed, 2003; Galley et al., 2004; Quirk et al., 2005; Wu and Fung, 2005).

25.11 Advanced: IBM Model 3 for fertility-based alignment

The seminal IBM paper that began work on statistical MT proposed five models for MT. We saw IBM's Model 1 in Sec. 25.5.1. Models 3, 4 and 5 all use the important concept of **fertility**. We'll introduce Model 3 in this section; our description here is influenced by Kevin Knight's nice tutorial (Knight, 1999b). Model 3 has a more complex generative model than Model 1. The generative model from an English sentence $E = e_1, e_2, ..., e_I$ has 5 steps:

FERTILITY

1. For each English word e_i, we choose a **fertility** ϕ_i.[6] The fertility is the number of (zero or more) Spanish words that will be generated from e_i, and is dependent only on e_i.

SPURIOUS WORDS

2. We also need to generate Spanish words from the NULL English word. Recall that we defined these earlier as **spurious words**. Instead of having a fertility for NULL, we'll generate spurious words differently. Every time we generate an

[6] This ϕ is not related to the ϕ that was used in phrase-based translation.

English word, we consider (with some probability) generating a spurious word (from NULL).

3. We now know how many Spanish words to generate from each English word. So now for each of these Spanish potential words, generate it by translating its aligned English word. As with Model 1, the translation will be based only on the English word. Spurious Spanish words will be generated by translating the NULL word into Spanish.
4. Move all the non-spurious words into their final positions in the Spanish sentence.
5. Insert the spurious Spanish words in the remaining open positions in the Spanish sentence.

Fig. 25.34 shows a visualization of the Model 3 generative process

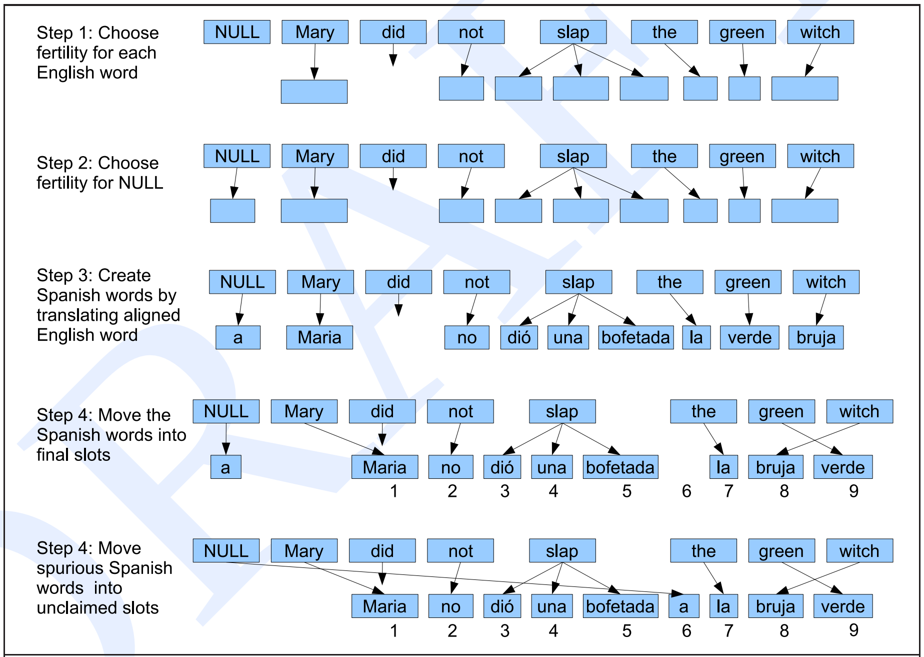

Step 1: Choose fertility for each English word
NULL Mary did not slap the green witch
Step 2: Choose fertility for NULL
NULL Mary did not slap the green witch
Step 3: Create Spanish words by translating aligned English word
NULL Mary did not slap the green witch
a Maria no dió una bofetada la verde bruja
Step 4: Move the Spanish words into final slots
NULL Mary did not slap the green witch
a Maria no dió una bofetada la bruja verde
1 2 3 4 5 6 7 8 9
Step 4: Move spurious Spanish words into unclaimed slots
NULL Mary did not slap the green witch
Maria no dió una bofetada a la bruja verde
1 2 3 4 5 6 7 8 9

Figure 25.34 The five steps of IBM Model 3 generating a Spanish sentence and alignment from an English sentence.

N
T
D
P1
DISTORTION

Model 3 has more parameters than Model 1. The most important are the **n**, **t**, **d**, and **p1** probabilities. The fertility probability ϕ_i of a word e_i is represented by the parameter n. So we will use $n(1|green)$ to represent the probability that English *green* will produce one Spanish word, $n(2|green)$ is the probability that English *green* will produce two Spanish words, $n(0|did)$ is the probability that English *did* will produce no Spanish words, and so on. Like IBM Model 1, Model 3 has a translation probability $t(f_j|e_i)$. Next, the probability that expresses the word position that English words end up in in the Spanish sentence is the **distortion** probability, which is conditioned on the

English and Spanish sentence lengths. The distortion probability $d(1,3,6,7)$ expresses the probability that the English word e_1 will align to Spanish word f_3, given that the English sentence has length 6, and the Spanish sentence is of length 7.

As we suggested above, Model 3 does not use fertility probabilities like $n(1|NULL)$, or $n(3|NULL)$ to decide how many spurious foreign words to generate from English NULL. Instead, each time Model 3 generates a real word, it generates a spurious word for the target sentence with probability p_1. This way, longer source sentences will naturally generate more spurious words. Fig. 25.35 shows a slightly more detailed version of the 5 steps of the Model 3 generative story using these parameters.

```
1. for each English word $e_i$, $1 < i < I$, we choose a fertility $\phi_i$ with probability $n(\phi_i|e_i)$
2. Using these fertilities and $p_1$, determine $\phi_0$, the number of spurious Spanish words, and hence $m$.
3. for each $i$, $0 < i < I$
     for each $k$, $1 < k < \phi_i$
        Choose a Spanish word $\tau_{ik}$ with probability $t(\tau_{ik}, e_i)$
4. for each $i$, $1 < i < I$
     for each $k$, $1 < k < \phi_i$
        Choose a target Spanish position $\pi_{ik}$ with probability $d(\pi_{ik}, i, I, J)$
5. for each $k$, $1 < k < \phi_0$
        Choose a target Spanish position $\pi_{0k}$ from one of the available Spanish slots, for a total probability of $\frac{1}{\phi_0!}$
```

Figure 25.35 The Model 3 generative story for generating a Spanish sentence from an English sentence. Remember that we are not translating from English to Spanish; this is just the generative component of the noisy channel model. Adapted from Knight (1999b).

Switching for a moment to the task of French to English translation, Fig. 25.36 shows some of the t and ϕ parameters learned for French-English translation from Brown et al. (1993). Note that *the* in general translates to a French article like *le*, but sometimes it has a fertility of 0, indicating that English uses an article where French does not. Conversely, note that *farmers* prefers a fertility of 2, and the most likely translations are *agriculteurs* and *les*, indicating that here French tends to use an article where English does not.

Now that we have seen the generative story for Model 3, let's build the equation for the probability assigned by the model. The model needs to assigns a probability $P(F|E)$ of generating the Spanish sentence F from the English sentence E. As we did with Model 1, we'll start by showing how the model gives the probability $P(F,A|E)$, the probability of generating sentence F via a particular alignment A. Then we'll sum over all alignments to get the total $P(F|E)$.

In order to compute $P(F,A|E)$, we'll need to multiply the main three factors n, t, and d, for generating words, translating them into Spanish, and moving them around.

the				***farmers***				***not***			
f	$t(f\|e)$	ϕ	$n(\phi\|e)$	f	$t(f\|e)$	ϕ	$n(\phi\|e)$	f	$t(f\|e)$	ϕ	$n(\phi\|e)$
le	0.497	1	0.746	agriculteurs	0.442	2	0.731	ne	0.497	2	0.735
la	0.207	0	0.254	les	0.418	1	0.228	pas	0.442	0	0.154
les	0.155			cultivateurs	0.046	0	0.039	non	0.029	1	0.107
l’	0.086			producteurs	0.021			rien	0.011		
ce	0.018										
cette	0.011										

Figure 25.36 Examples of Model 3 parameters from the Brown et al. (1993) French-English translation system, for three English words. Note that both *farmers* and *not* are likely to have fertilities of 2.

So a first pass at $P(F,A|E)$ would be:

$$\prod_{i=1}^{I} n(\phi_i|e_i) \times \prod_{j=1}^{J} t(f_j|e_{a_j}) \times \prod_{j=1}^{J} d(j|a_j,I,J) \tag{25.38}$$

But (25.38) isn’t sufficient as it stands; we need to add factors for generating spurious words, for inserting them into the available slots, and a factor having to do with the number of ways (permutations) a word can align with multiple words. Equation (25.39) gives the true final equation for IBM Model 3, in Knight’s modification of the original formula. We won’t give the details of these additional factors, but encourage the interested reader to see the original presentation in Brown et al. (1993) and the very clear explanation of the equation in Knight (1999b).

$$\begin{aligned} P(F,A|E) \;=\; & \overbrace{\binom{J-\phi_0}{\phi_0} p_0^{J-2\phi_0} p_1^{\phi_0}}^{\text{generate spurious}} \times \overbrace{\frac{1}{\phi_0!}}^{\text{insert spurious}} \times \overbrace{\prod_{i=0}^{I} \phi_i!}^{\text{multi-align permutations}} \\ & \times \prod_{i=1}^{I} n(\phi_i|e_i) \times \prod_{j=1}^{J} t(f_j|e_{a_j}) \times \prod_{j:a_j\neq 0}^{J} d(j|a_j,I,J) \end{aligned} \tag{25.39}$$

Once again, in order to get the total probability of the Spanish sentence we’ll need to sum over all possible alignments:

$$P(F|E) \;=\; \sum_{A} P(F,A|E)$$

We can also make it more explicit exactly how we sum over alignments (and also emphasize the incredibly large number of possible alignments) by expressing this formula as follows, where we specify an alignment by specifying the aligned English a_j for each of the J words in the foreign sentence:

$$P(F|E) \;=\; \sum_{a_1=0}^{J}\sum_{a_2=0}^{J}\cdots\sum_{a_J=0}^{I} P(F,A|E)$$

25.11.1 Training for Model 3

Given a parallel corpus, training the translation model for IBM Model 3 means setting values for the n, d, t, and p_1 parameters.

As we noted for Model 1 and HMM models, if the training-corpus was hand-labeled with perfect alignments, getting maximum likelihood estimates would be simple. Consider the probability $n(0|did)$ that a word like *did* would have a zero fertility. We could estimate this from an aligned corpus just by counting the number of times *did* aligned to nothing, and normalize by the total count of *did*. We can do similar things for the t translation probabilities. To train the distortion probability $d(1,3,6,7)$, we similarly count the number of times in the corpus that English word e_1 maps to Spanish word f_3 in English sentences of length 6 that are aligned to Spanish sentences of length 7. Let's call this counting function dcount. We'll again need a normalization factor;

$$d(1,3,6,7) = \frac{\text{dcount}(1,3,6,7)}{\sum_{i=1}^{I} \text{dcount}(i,3,6,7)} \tag{25.40}$$

Finally, we need to estimate p_1. Again, we look at all the aligned sentences in the corpus; let's assume that in the Spanish sentences there are a total of N words. From the alignments for each sentence, we determine that a total of S Spanish words are spurious, i.e. aligned to English NULL. Thus $N-S$ of the words in the Spanish sentences were generated by real English words. After S of these $N-S$ Spanish words, we generate a spurious word. The probability p_1 is thus $S/(N-S)$.

Of course, we don't have hand-alignments for Model 3. We'll need to use EM to learn the alignments and the probability model simultaneously. With Model 1 and the HMM model, there were efficient ways to do training without explicitly summing over all alignments. Unfortunately, this is not true for Model 3; we actually would need to compute all possible alignments. For a real pair of sentences, with 20 English words and 20 Spanish words, and allowing NULL and allowing fertilities, there are a very large number of possible alignments (determining the exact number of possible alignments is left as Exercise 25.7). Instead, we approximate by only considering the best few alignments. In order to find the best alignments without looking at all alignments, we can use an iterative or bootstrapping approach. In the first step, we train the simpler IBM Model 1 or 2 as discussed above. Then we use these Model 2 parameters to evaluate $P(A|E,F)$, giving a way to find the best alignments to bootstrap Model 3. See Brown et al. (1993) and Knight (1999b) for details.

25.12 Advanced: Log-linear Models for MT

While statistical MT was first based on the noisy channel model, much recent work combines the language and translation models via a log-linear model in which we directly search for the sentence with the highest posterior probability:

$$\hat{E} = \underset{E}{\operatorname{argmax}} P(E|F) \tag{25.41}$$

This is done by modeling $P(E|F)$ via a set of M feature functions $h_m(E,F)$, each of which has a parameter λ_m. The translation probability is then:

$$P(E|F) = \frac{\exp[\sum_{m=1}^{M}\lambda_m h_m(E,F)]}{\sum_{E'}\exp[\sum_{m=1}^{M}\lambda_m h_m(E',F)]} \tag{25.42}$$

The best sentence is thus:

$$\begin{aligned}\hat{E} &= \underset{E}{\operatorname{argmax}} P(E|F)\\ &= \underset{E}{\operatorname{argmax}} \exp[\sum_{m=1}^{M}\lambda_m h_m(E,F)]\end{aligned} \tag{25.43}$$

In practice, the noisy channel model factors (the language model $P(E)$ and translation model $P(F|E)$), are still the most important feature functions in the log-linear model, but the architecture has the advantage of allowing for arbitrary other features as well; a common set of features would include:

REVERSE TRANSLATION MODEL

WORD PENALTY

PHRASE PENALTY

UNKNOWN WORD PENALTY

- the language model $P(E)$
- the translation model $P(F|E)$
- the **reverse translation model** $P(E|F)$,
- lexicalized versions of both translation models,
- a **word penalty**,
- a **phrase penalty**
- an **unknown word penalty**.

See Foster (2000), Och and Ney (2002, 2004) for more details.

Log-linear models for MT could be trained using the standard maximum mutual information criterion.

MINIMUM ERROR RATE TRAINING

MERT

In practice, however, log-linear models are instead trained to directly optimize evaluation metrics like Bleu in a method known as **Minimum Error Rate Training**, or **MERT** (Och, 2003; Chou et al., 1993).

Bibliographical and Historical Notes

Work on models of the process and goals of translation goes back at least to Saint Jerome in the fourth century (Kelley, 1979). The development of logical languages, free of the imperfections of human languages, for reasoning correctly and for communicating truths and thereby also for translation, has been pursued at least since the 1600s (Hutchins, 1986).

By the late 1940s, scant years after the birth of the electronic computer, the idea of MT was raised seriously (Weaver, 1955). In 1954 the first public demonstration of a MT system prototype (Dostert, 1955) led to great excitement in the press (Hutchins, 1997). The next decade saw a great flowering of ideas, prefiguring most subsequent developments. But this work was ahead of its time — implementations were limited by, for example, the fact that pending the development of disks there was no good way to store dictionary information.

As high quality MT proved elusive (Bar-Hillel, 1960), a growing consensus on the need for better evaluation and more basic research in the new fields of formal and computational linguistics, culminating in the famous ALPAC (Automatic Language Processing Advisory Committee) report of 1966 (Pierce et al., 1966), led in the mid 1960s to a dramatic cut in funding for MT. As MT research lost academic respectability, the Association for Machine Translation and Computational Linguistics dropped MT from its name. Some MT developers, however, persevered, slowly and steadily improving their systems, and slowly garnering more customers. Systran in particular, developed initially by Peter Toma, has been continuously improved over 40 years. Its earliest uses were for information acquisition, for example by the U.S. Air Force for Russian documents; and in 1976 an English-French edition was adopted by the European Community for creating rough and post-editable translations of various administrative documents. Another early successful MT system was Météo, which translated weather forecasts from English to French; incidentally, its original implementation (1976), used "Q-systems", an early unification model.

The late 1970s saw the birth of another wave of academic interest in MT. One strand attempted to apply meaning-based techniques developed for story understanding and knowledge engineering (Carbonell et al., 1981). There were wide discussions of interlingual ideas through the late 1980s and early 1990s (Tsujii, 1986; Nirenburg et al., 1992; Ward, 1994; Carbonell et al., 1992). Meanwhile MT usage was increasing, fueled by globalization, government policies requiring the translation of all documents into multiple official languages, and the proliferation of word processors and then personal computers.

Modern statistical methods began to be applied in the early 1990s, enabled by the development of large bilingual corpora and the growth of the web. Early on, a number of researchers showed that it was possible to extract pairs of aligned sentences from bilingual corpora (Kay and Röscheisen, 1988, 1993; Warwick and Russell, 1990; Brown et al., 1991; Gale and Church, 1991, 1993). The earliest algorithms made use of the words of the sentence as part of the alignment model, while others relied solely on other cues like sentence length in words or characters.

CANDIDE

EGYPT

GIZA++

At the same time, the IBM group, drawing directly on algorithms for speech recognition (many of which had themselves been developed originally at IBM!) proposed the **Candide** system, based on the IBM statistical models we have described (Brown et al., 1990, 1993). These papers described the probabilistic model and the parameter estimation procedure. The decoding algorithm was never published, but it was described in a patent filing (Brown et al., 1995). The IBM work had a huge impact on the research community, and by the turn of this century, much or most academic research on machine translation was statistical. Progress was made hugely easier by the development of publicly-available toolkits, particularly tools extended from the **EGYPT** toolkit developed by the Statistical Machine Translation team in during the summer 1999 research workshop at the Center for Language and Speech Processing at the Johns Hopkins University. These include the **GIZA++** aligner, developed by Franz Josef Och by extending the GIZA toolkit (Och and Ney, 2003), which implements IBM models 1-5 as well as the HMM alignment model.

Initially most research implementations focused on IBM Model 3, but very quickly researchers moved to phrase-based models. While the earliest phrase-based translation model was IBM Model 4 (Brown et al., 1993), modern models derive from Och's (1998) work on **alignment templates**. Key phrase-based translation models include Marcu and Wong (2002), Zens et al. (2002). Venugopal et al. (2003), Koehn et al. (2003), Tillmann (2003) Och and Ney (2004), Deng and Byrne (2005), and Kumar and Byrne (2005),

Other work on MT decoding includes the A^* decoders of Wang and Waibel (1997) and Germann et al. (2001), and the polynomial-time decoder for binary-branching stochastic transduction grammar of Wu (1996).

MOSES

PHARAOH

The most recent open-source MT toolkit is the phrase-based **MOSES** system (Koehn et al., 2006; Koehn and Hoang, 2007; Zens and Ney, 2007). MOSES developed out of the **PHARAOH** publicly available phrase-based stack decoder, developed by Philipp Koehn (Koehn, 2004, 2003b), which extended the A^* decoders of (Och et al., 2001) and Brown et al. (1995) and extended the EGYPT tools discussed above.

Modern research continues on sentence and word alignment as well; more recent algorithms include Moore (2002, 2005), Fraser and Marcu (2005), Callison-Burch et al. (2005), Liu et al. (2005).

Research on evaluation of machine translation began quite early. Miller and Beebe-Center (1958) proposed a number of methods drawing on work in psycholinguistics. These included the use of cloze and Shannon tasks to measure intelligibility, as well as a metric of edit distance from a human translation, the intuition that underlies all modern automatic evaluation metrics like Bleu. The ALPAC report included an early evaluation study conducted by John Carroll that was extremely influential (Pierce et al., 1966, Appendix 10). Carroll proposed distinct measures for fidelity and intelligibility, and had specially trained human raters score them subjectively on 9-point scales. More recent work on evaluation has focused on coming up with automatic metrics, include the work on Bleu discussed in Sec. 25.9.2 (Papineni et al., 2002), as well as related measures like **NIST** (Doddington, 2002), **TER (Translation Error Rate)** (Snover et al., 2006),**Precision and Recall** (Turian et al., 2003), and **METEOR** (Banerjee and Lavie, 2005).

Good surveys of the early history of MT are Hutchins (1986) and (1997). The textbook by Hutchins and Somers (1992) includes a wealth of examples of language phenomena that make translation difficult, and extensive descriptions of some historically significant MT systems. Nirenburg et al. (2002) is a comprehensive collection of classic readings in MT. (Knight, 1999b) is an excellent tutorial introduction to Statistical MT.

Academic papers on machine translation appear in standard NLP journals and conferences, as well as in the journal *Machine Translation* and in the proceedings of various conferences, including MT Summit, organized by the International Association for Machine Translation, the individual conferences of its three regional divisions, (Association for MT in the Americas – AMTA, European Association for MT – EAMT, and Asia-Pacific Association for MT – AAMT), and the Conference on Theoretical and Methodological Issue in Machine Translation (TMI).

Exercises

25.1 Select at random a paragraph of Ch. 12 which describes a fact about English syntax. a) Describe and illustrate how your favorite foreign language differs in this respect. b) Explain how a MT system could deal with this difference.

25.2 Choose a foreign language novel in a language you know. Copy down the shortest sentence on the first page. Now look up the rendition of that sentence in an English translation of the novel. a) For both original and translation, draw parse trees. b) For both original and translation, draw dependency structures. c) Draw a case structure representation of the meaning which the original and translation share. d) What does this exercise suggest to you regarding intermediate representations for MT?

25.3 Version 1 (for native English speakers): Consider the following sentence:

> These lies are like their father that begets them; gross as a mountain, open, palpable.
>
> Henry IV, Part 1, act 2, scene 2

Translate this sentence into some dialect of modern vernacular English. For example, you might translate it into the style of a New York Times editorial or an Economist opinion piece, or into the style of your favorite television talk-show host.

Version 2 (for native speakers of other languages): Translate the following sentence into your native language.

> One night my friend Tom, who had just moved into a new apartment, saw a cockroach scurrying about in the kitchen.

For either version, now:

a) Describe how you did the translation: What steps did you perform? In what order did you do them? Which steps took the most time? b) Could you write a program that would translate using the same methods that you did? Why or why not? c) What aspects were hardest for you? Would they be hard for a MT system? d) What aspects would be hardest for a MT system? are they hard for people too? e) Which models are best for describing various aspects of your process (direct, transfer, interlingua or statistical)? f) Now compare your translation with those produced by friends or classmates. What is different? Why were the translations different?

25.4 Type a sentence into a MT system (perhaps a free demo on the web) and see what it outputs. a) List the problems with the translation. b) Rank these problems in order of severity. c) For the two most severe problems, suggest the probable root cause.

25.5 Build a very simple direct MT system for translating from some language you know at least somewhat into English (or into a language in which you are relatively fluent), as follows. First, find some good test sentences in the source language. Reserve half of these as a development test set, and half as an unseen test set. Next, acquire a bilingual dictionary for these two languages (for many languages, limited dictionaries can be found on the web that will be sufficient for this exercise). Your program should translate each word by looking up its translation in your dictionary. You may need to implement some stemming or simple morphological analysis. Next, examine your output, and do a preliminary error analysis on the development test set. What are the major sources of error? Write some general rules for correcting the translation mistakes. You will probably want to run a part-of-speech tagger on the English output, if you have one. Then see how well your system runs on the test set.

25.6 Continue the calculations for the EM example on page 30, performing the second and third round of E-steps and M-steps.

25.7 (Derived from Knight (1999b)) How many possible Model 3 alignments are there between a 20-word English sentence and a 20-word Spanish sentence, allowing for NULL and fertilities?

Banerjee, S. and Lavie, A. (2005). METEOR: An automatic metric for Mt evaluation with improved correlation with human judgments. In *Proceedings of ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization*.

Bar-Hillel, Y. (1960). The present status of automatic translation of languages. In Alt, F. (Ed.), *Advances in Computers 1*, pp. 91–163. Academic Press.

Bickel, B. (2003). Referential density in discourse and syntactic typology. *Language*, *79*(2), 708–736.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lai, J., and Mercer, R. L. (1995). Method and system for natural language translation. U.S. Patent 5,477,451.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. (1990). A statistical approach to machine translation. *Computational Linguistics*, *16*(2), 79–85.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, *19*(2), 263–311.

Brown, P. F., Lai, J. C., and Mercer, R. L. (1991). Aligning sentences in parallel corpora. In *Proceedings of the 29th ACL*, Berkeley, CA, pp. 169–176.

Callison-Burch, C., Osborne, M., and Koehn, P. (2006). Re-evaluating the role of BLEU in machine translation research. In *EACL-06*.

Callison-Burch, C., Talbot, D., and Osborne, M. (2005). Statistical marchine translation with word- and sentences-aligned parallel corpora. In *ACL-05*, pp. 176–183. ACL.

Cao, X. (1792). *The Story of the Stone.* (Also known as *The Dream of the Red Chamber*). Penguin Classics, London. First published in Chinese in 1792, translated into English by David Hawkes and published by Penguin in 1973.

Carbonell, J., Cullingford, R. E., and Gershman, A. V. (1981). Steps toward knowledge-based machine translation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *3*(4), 376–392.

Carbonell, J., Mitamura, T., and Nyberg, E. H. (1992). The KANT perspective: A critique of pure transfer (and pure interlingua, pure statistics, ...). In *International Conference on Theoretical and Methodological Issues in Machine Translation*.

Chandioux, J. (1976). MÉTÉO: un système opérationnel pour la traduction automatique des bulletins météorologiques destinés au grand public. *Meta*, *21*, 127–133.

Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. In *ACL-05*, Ann Arbor, MI, pp. 263–270. ACL.

Chou, W., Lee, C. H., and Juang, B. H. (1993). Minimum error rate training based on *n*-best string models. In *IEEE ICASSP-93*, pp. 2.652–655.

Comrie, B. (1989). *Language Universals and Linguistic Typology*. Blackwell, Oxford. Second edition.

Croft, W. (1990). *Typology and Universals*. Cambridge University Press.

Deng, Y. and Byrne, W. (2005). Hmm word and phrase alignment for statistical machine translation. In *HLT-EMNLP-05*.

Doddington, G. (2002). Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *HLT-01*.

Dorr, B. (1994). Machine translation divergences: A formal description and proposed solution. *Computational Linguistics*, *20*(4), 597–633.

Dostert, L. (1955). The Georgetown-I.B.M. experiment. In *Machine Translation of Languages: Fourteen Essays*, pp. 124–135. MIT Press.

Eisner, J. (2003). Learning non-isomorphic tree mappings for machine translation. In *ACL-03*. ACL.

Foster, G. (2000). A maximum entropy/minimum divergence translation model. In *ACL-00*, Hong Kong.

Fraser, A. and Marcu, D. (2005). Isi's participation in the romanian-english alignment task. In *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, pp. 91–94. ACL.

Gale, W. A. and Church, K. W. (1991). A program for aligning sentences in bilingual corpora. In *Proceedings of the 29th ACL*, Berkeley, CA, pp. 177–184.

Gale, W. A. and Church, K. W. (1993). A program for aligning sentences in bilingual corpora. *Computational Linguistics*, *19*, 75–102.

Galley, M., Hopkins, M., Knight, K., and Marcu, D. (2004). What's in a translation rule?. In *HLT-NAACL-04*.

Germann, U., Jahr, M., Knight, K., Marcu, D., and Yamada, K. (2001). Fast decoding and optimal decoding for machine translation. In *ACL-01*, pp. 228–235.

Hutchins, J. (1997). From first conception to first demonstration: the nascent years of machine translation, 1947–1954. A chronology. *Machine Translation*, *12*, 192–252.

Hutchins, W. J. and Somers, H. L. (1992). *An Introduction to Machine Translation*. Academic Press.

Hutchins, W. J. (1986). *Machine Translation: Past, Present, Future*. Ellis Horwood, Chichester, England.

Jelinek, F. (1969). A fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, *13*, 675–685.

Kay, M. and Röscheisen, M. (1988). Text-translation alignment. Tech. rep. P90-00143, Xerox Palo Alto Research Center, Palo Alto, CA.

Kay, M. and Röscheisen, M. (1993). Text-translation alignment. *Computational Linguistics*, *19*, 121–142.

Kelley, L. G. (1979). *The True Interpreter: A History of Translation Theory and Practice in the West*. St. Martin's Press, New York.

Knight, K. (1999a). Decoding complexity in word-replacement translation models. *Computational Linguistics*, *25*(4), 607–615.

Knight, K. (1999b). A statistical MT tutorial workbook..

Koehn, P. and Hoang, H. (2007). Factored translation models. In *EMNLP/CoNLL 2007*.

Koehn, P. (2003a). *Noun Phrase Translation*. Ph.D. thesis, University of Southern California.

Koehn, P. (2003b). Pharaoh: a beam search decoder for phrase-based statistical machine translation models: User manual and description..

Koehn, P. (2004). Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Proceedings of AMTA 2004*.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2006). Moses: Open Source Toolkit for Statistical Machine Translation. In *ACL-07*, Prague. Demonstration session.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *HLT-NAACL-03*, pp. 48–54.

Kumar, S. and Byrne, W. (2005). Local phrase reordering models for statistical machine translation. In *HLT-EMNLP-05*.

Levin, L., Gates, D., Lavie, A., and Waibel, A. (1998). An interlingua based on domain actions for machine translation of task-oriented dialogues. In *ICSLP-98*, Sydney, pp. 1155–1158.

Li, C. N. and Thompson, S. A. (1981). *Mandarin Chinese: A Functional Reference Grammar*. University of California Press.

Liu, Y., Liu, Q., and Lin, S. (2005). Log-linear models for word alignment. In *ACL-05*, pp. 459–466. ACL.

Marcu, D. and Wong, W. (2002). A phrase-based, joint probability model for statistical machine translation. In *EMNLP 2002*, pp. 133–139.

McLuhan, M. (1964). *Understanding media: The extensions of man*. New American Library, New York.

Melamed, I. D. (2003). Multitext grammars and synchronous parsers. In *HLT-NAACL-03*. ACL.

Miller, G. A. and Beebe-Center, J. G. (1958). Some psychological methods for evaluating the quality of translations. *Mechanical Translation*, *3*, 73–80.

Moore, R. C. (2002). Fast and accurate sentence alignment of bilingual corpora. In *Machine Translation: From Research to Real Users (Proceedings, 5th Conference of the Association for Machine Translation in the Americas, Tiburon, California)*, pp. 135–244.

Moore, R. C. (2005). A discriminative framework for bilingual word alignment. In *HLT-EMNLP-05*, pp. 81–88.

Nesson, R., Shieber, S. M., and Rush, A. (2006). Induction of probabilistic synchronous tree-insertion grammars for machine translation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA 2006)*, Boston, MA.

Nichols, J. (1986). Head-marking and dependent-marking grammar. *Language*, *62*(1), 56–119.

Nirenburg, S., Carbonell, J., Tomita, M., and Goodman, K. (1992). *Machine Translation: A Knowledge-based Approach*. Morgan Kaufmann.

Nirenburg, S., Somers, H. L., and Wilks, Y. A. (Eds.). (2002). *Readings in Machine Translation*. MIT Press.

Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *ACL-03*, pp. 160–167.

Och, F. J. and Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. In *ACL-02*, pp. 295–302.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, *29*(1), 19–51.

Och, F. J. and Ney, H. (2004). The alignment template approach to statistical machine translation. *Computational Linguistics*, *30*(4), 417–449.

Och, F. J., Ueffing, N., and Ney, H. (2001). An efficient a* search algorithm for statistical machine translation. In *Proceedings of the ACL Workshop on Data-Driven methods in Machine Translation*, pp. 1–8.

Och, F. J. (1998). *Ein beispielsbasierter und statistischer Ansatz zum maschinellen Lernen von natürlichsprachlicher Übersetzung*. Ph.D. thesis, Universität Erlangen-Nürnberg, Germany. Diplomarbeit (diploma thesis).

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *ACL-02*, Philadelphia, PA.

Pierce, J. R., Carroll, J. B., and et al (1966). *Language and Machines: Computers in Translation and Linguistics*. ALPAC report. National Academy of Sciences, National Research Council, Washington, DC.

Quirk, C., Menezes, A., and Cherry, C. (2005). Dependency treelet translation: Syntactically informed phrasal smt. In *ACL-05*. ACL.

Senellart, J., Dienes, P., and Váradi, T. (2001). New generation systran translation system. In *MT Summit 8*.

Shieber, S. M. (1994). Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, *10*(4), 371–385.

Shieber, S. M. and Schabes, Y. (1992). Generation and synchronous tree-adjoining grammars. *Computational Intelligence*, *7*(4), 220–228.

Slobin, D. I. (1996). Two ways to travel. In Shibatani, M. and Thompson, S. A. (Eds.), *Grammatical Constructions: Their Form and Meaning*, pp. 195–220. Clarendon Press, Oxford.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *AMTA-2006*.

Talmy, L. (1985). Lexicalization patterns: Semantic structure in lexical forms. In Shopen, T. (Ed.), *Language Typology and Syntactic Description, Volume 3*. Cambridge University Press. Originally appeared as UC Berkeley Cognitive Science Program Report No. 30, 1980.

Talmy, L. (1991). Path to realization: a typology of event conflation. In *BLS-91*, Berkeley, CA, pp. 480–519.

Taylor, W. L. (1953). Cloze procedure: a new tool for measuring readability. *Journalism Quarterly*, *30*, 415–433.

Taylor, W. L. (1957). Cloze readability scores as indices of individual differences in comprehension and aptitude. *Journal of Applied Psychology*, *4*, 19–26.

Tillmann, C. (2003). A projection extension algorithm for statistical machine translation. In *EMNLP 2003*, Sapporo, Japan.

Toutanova, K., Ilhan, H. T., and Manning, C. D. (2002). Extensions to HMM-based statistical word alignment models. In *EMNLP 2002*, pp. 87–94.

Tsujii, J. (1986). Future directions of machine translation. In *COLING-86*, Bonn, pp. 655–668.

Turian, J. P., Shen, L., and Melamed, I. D. (2003). Evaluation of machine translation and its evaluation. In *Proceedings of MT Summit IX*, New Orleans, LA.

Venugopal, A., Vogel, S., and Waibel, A. (2003). Effective phrase translation extraction from alignment models. In *ACL-03*, pp. 319–326.

Wang, Y. Y. and Waibel, A. (1997). Decoding algorithm in statistical machine translation. In *ACL/EACL-97*, pp. 366–372.

Ward, N. (1994). *A Connectionist Language Generator*. Ablex.

Warwick, S. and Russell, G. (1990). Bilingual concordancing and bilingual lexicography. In *EURALEX 4th International Congress*.

Waugh, L. R. (1976). The semantics and paradigmatics of word order. *Language*, *52*(1), 82–107.

Weaver, W. (1949/1955). Translation. In Locke, W. N. and Boothe, A. D. (Eds.), *Machine Translation of Languages*, pp. 15–23. MIT Press. Reprinted from a memorandum written by Weaver in 1949.

Wu, D. (1996). A polynomial-time algorithm for statistical machine translation. In *ACL-96*, Santa Cruz, CA, pp. 152–158.

Wu, D. (2000). Bracketing and aligning words and constituents in parallel text using stochastic inversion transduction grammars. In Veronis, J. (Ed.), *Parallel Text Processing: Alignment and Use of Translation Corpora*. Kluwer, Dordrecht.

Wu, D. and Fung, P. (2005). Inversion transduction grammar constraints for mining parallel sentences from quasi-comparable corpora. In *IJCNLP-2005*, Jeju, Korea.

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. In *ACL-01*, Toulouse, France.

Yamada, K. and Knight, K. (2002). A decoder for syntax-based statistical MT. In *ACL-02*, Philadelphia, PA.

Zens, R. and Ney, H. (2007). Efficient phrase-table representation for machine translation with applications to online MT and speech translation. In *NAACL-HLT 07*, Rochester, NY, pp. 492–499.

Zens, R., Och, F. J., and Ney, H. (2002). Phrase-based statistical machine translation. In *KI 2002*, pp. 18–32.

Zhang, H. and Gildea, D. (2005). Stochastic lexicalized inversion transduction grammar for alignment. In *ACL-05*, Ann Arbor, MI. ACL.