Discussion of Dale and Reiter (1995)

by Matthew Monaghan

The Problem

At the time this paper was written Dale and Reiter's Incremental Algorithm was being run as part of the Intelligent Documentation Advisory System (IDAS) natural language generation program. IDAS was designed to help technicians use a device that tested faulty electronic equipment.

Example I/0: What-is-it **Printer 12**?
Answer: The printer is a white Epson LQ 1010 printer. (Reiter, 1992)

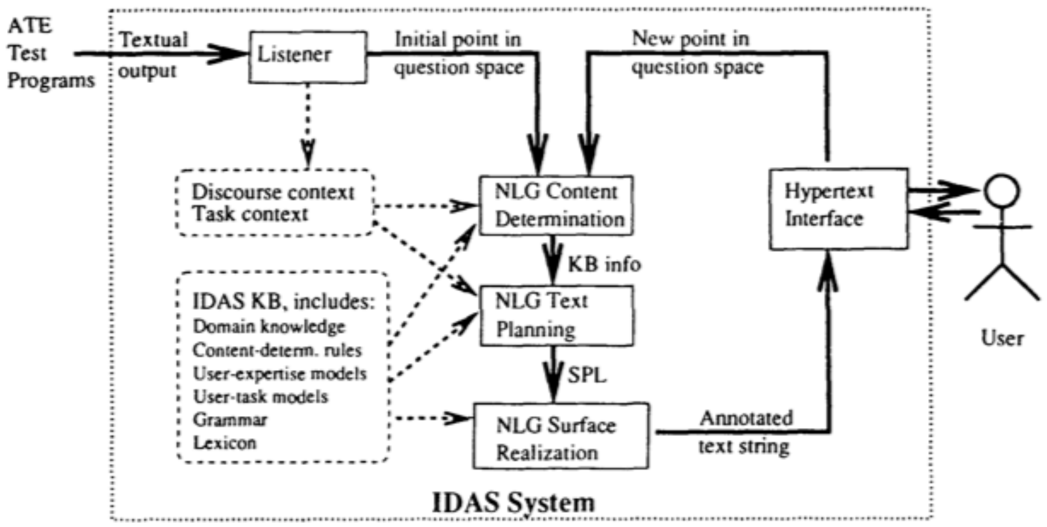


Figure 1: Simplified IDAS architecture

In this paper, they target and delimit the generation of Referring Expressions (REs) as a subproblem. They define REs as the following:

I. REs will consist of only definite noun phrases (e.g. the red cup), not pronouns, one-anaphora, etc.
II. They must refer to physical objects (e.g. dogs and tables) not abstract entities.
III. They are solely intended to identify the target object to the hearer, accurately describe one object and no other, and not satisfy any other communicative goal.

Also, the criteria of a RE generating algorithm:

I. REs should satisfy the "referential communicative goal" i.e. after receiving the RE, the hearer/reader should be able to identify the target object.
II. REs should not lead hearer/reader to make false conversational implicatures.
III. The underlying algorithm should be "computationally efficient".

The Algorithm

Assumptions:

I. Every entity is characterized in terms of a collection of **attributes** and their **values**; e.g. <colour, red>.
II. Every entity has as one of its attributes some **type**; e.g. <type, dog>.
III. The knowledge base may organize some attribute values in a subsumption taxonomy e.g. animal subsumes dog; red subsumes scarlet.

The computational cost of generating REs are stated in terms of the these parameters:

n_a = the number of attributes that are available to a referring expression (i.e. the number of properties known to be true of the intended referent);
n_d = the number of distracters in the current context set;
n_l = the number of attributes mentioned in the final referring expression.

From Dale and Reiter's dog example:

n_a= 3 (type, size, and colour);
n_d =2 (Object2, Object3); and
n_l =2 (<type, dog>, <colour, black>).

Other Models:

We should keep in mind the cost of these algorithms (run time as a function of *n* as well as what is at stake i.e. human technician performing job) and comparisons to human cognition.

3.1.1 Full Brevity

Dale (1989, 1992) suggested that a generation system should output the shortest possible RE i.e. use as few attributes as possible. Such an algorithm satisfies a strict interpretation of Quantity, Relevance and Brevity maxims, but finding such an RE is NP-Hard (Can be thought of as an exhaustive search; checks if any one-component RE is successful, then two-component, etc., until success).

Suppose n_l= 3, and n_a = 10: then, 175 descriptions must be checked.
Suppose n_l= 4, and n_a = 20: over 6,000 descriptions must be checked.
Suppose n_l= 5, and n_a = 50: over 2,000,000 descriptions must be checked.

3.1.2 Greedy Heuristic

The algorithm is easier and quicker to implement than Full Brevity, but does not always generate RE with fewest possible attributes. It chooses attributes by trying to maximize the number of distracters ruled out by each property.

Computational cost: algorithm takes n_l passes through the problem, checking each of the n_a potential attributes to determine how many n_d distracters they rule out.
Total run time = $n_a * n_d * n_l$

3.1.3 Local Brevity

Reiter (1990a) proposed RE generation system should adhere to following the preference rules:

I. No unnecessary components.
II. It should not be possible to produce a shorter RE by replacing a set of existing components by a single new component.
III. Lexical Preference

These resulted in an algorithm that starts with an initial distinguishing description and checks if a new distinguishing description can be formed by applying preference rules (i.e. removing an attribute, replacing two attributes by a single attribute, replacing a value by lexically preferred value.

Using the Greedy Heuristic to generate initial description, total run time = $n_a * n_d * n_l$

Note: Many of these functions are derived from interpretations of Grice's work and other theories/hypotheses from psychology.

Incremental Algorithm

Interface functions:

MakeReferringExpression is the top-level function. This returns a list of attribute-value pairs for the RE.

MoreSpecificValue(object, attribute, value) returns a new value for the attribute where that value is more specific than the current value.

BasicLevelValue(object, attribute) returns the basic-level value of an attribute of an object, from the point of view of the current user i.e. **BasicLevelValue(**Garfield, type**)** might be cat.

UserKnows(object, attribute-value-pair**)** returns true if the user can easily determine (e.g., by direct visual perception) that the attribute-value pair applies to the object; false if the user can easily determine that the attribute-value pair does not apply to the object; and unknown otherwise.

PreferredAttributes contains the attributes that human speakers and hearers prefer (task specific).

FindBestValue takes an attribute and an initial value; returns a value for that attribute that is subsumed by the initial value, accurately describes the intended referent (i.e., subsumes the value the intended referent possesses for the attribute), rules out as many distracters as possible, and, subject to these constraints, is as close as possible in the taxonomy to the initial value.

RulesOut returns the elements of the set of remaining distracters that are ruled out by a given attribute-value pair.

The algorithm iterates through the list **PreferredAttributes**, checking if including an attribute would rule out at least one member of the contrast set that has not already been ruled out. This process continues until all members of the contrast set have been ruled out. Once an attribute is added to the RE it cannot be removed. A head noun (value for the **type** attribute) is always included.

```
$\boxed{\text{MakeReferringExpression}(r,\ C,\ P)}$
$L \leftarrow \{\}$
for each member $A_i$ of list $P$ do
  $V$ = FindBestValue($r$, $A_i$, BasicLevelValue($r$, $A_i$))
  if RulesOut($\langle A_i, V\rangle$) $\neq$ nil
  then $L \leftarrow L \cup \{\langle A_i, V\rangle\}$
       $C \leftarrow C$ − RulesOut($\langle A_i, V\rangle$)
  endif
  if $C = \{\}$ then
    if $\langle$type, $X\rangle \in L$ for some $X$
      then return $L$
      else return $L \cup \{\langle$type, BasicLevelValue($r$, type)$\rangle\}$
      endif
  endif
return failure

$\boxed{\text{FindBestValue}(r,\ A,\ \textit{initial-value})}$
if UserKnows($r$, $\langle A$, *initial-value*$\rangle$) = true
then *value* $\leftarrow$ *initial-value*
else *value* $\leftarrow$ no-value
endif
if (*more-specific-value* $\leftarrow$ MoreSpecificValue($r$, $A$, *value*)) $\neq$ nil $\wedge$
  (*new-value* $\leftarrow$ FindBestValue($A$, *more-specific-value*)) $\neq$ nil $\wedge$
  (|RulesOut($\langle A$, *new-value*$\rangle$)| > |RulesOut($\langle A$, *value*$\rangle$)|)
then *value* $\leftarrow$ *new-value*
endif
return *value*

$\boxed{\text{RulesOut}(\langle A,\ V\rangle)}$
if $V$ = no-value
then return nil
else return $\{x : x \in C \wedge$ UserKnows($x$, $\langle A, V\rangle$) = false$\}$
endif
```

Figure 6. The Algorithm.

Example:

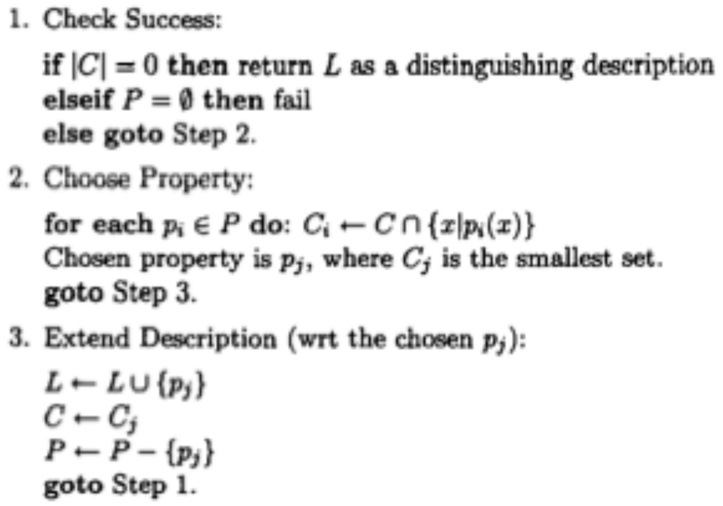
Object1: <type, Chihuahua>, <size, small >, < colour, black>
Object2: <type, Chihuahua>, <size, large>, <colour, white>
Object3: <type, Siamese-cat >, <size, small >, <colour, black >

r = Object1 and *C* = (Object2, Object3). Assume that *P* ={type, colour, size, etc.}.

Resulting RE: *L* = {<type, dog>, <colour, black>}

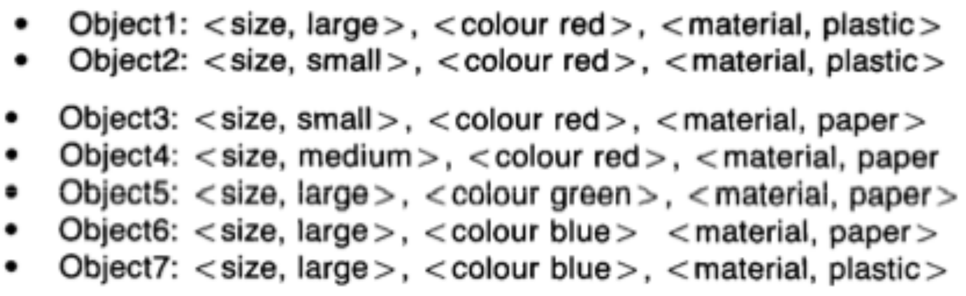
Greedy Algorithms

"An algorithm is greedy if it builds up a solution in small steps, choosing a decision myopically to optimize some underlying criterion." (Kleinberg 2014)



```
1. Check Success:
    if $|C| = 0$ then return $L$ as a distinguishing description
    elseif $P = \emptyset$ then fail
    else goto Step 2.
2. Choose Property:
    for each $p_i \in P$ do: $C_i \leftarrow C \cap \{x|p_i(x)\}$
    Chosen property is $p_j$, where $C_j$ is the smallest set.
    goto Step 3.
3. Extend Description (wrt the chosen $p_j$):
    $L \leftarrow L \cup \{p_j\}$
    $C \leftarrow C_j$
    $P \leftarrow P - \{p_j\}$
    goto Step 1.
```

Figure 3. An algorithm for the Greedy Heuristic Interpretation.



- Object1: <size, large>, <colour red>, <material, plastic>
- Object2: <size, small>, <colour red>, <material, plastic>
- Object3: <size, small>, <colour red>, <material, paper>
- Object4: <size, medium>, <colour red>, <material, paper
- Object5: <size, large>, <colour green>, <material, paper>
- Object6: <size, large>, <colour blue> <material, paper>
- Object7: <size, large>, <colour blue>, <material, plastic>

The algorithm first selects *plastic*, then selects *large* or *red*, and finally selects *red* or *large* (whichever of the two was not selected in the second step). The result, once the head noun has been added, is the noun phase *the large red plastic cup;* however, the true minimal description is *the large red cup*.

"When a Greedy algorithm succeeds in solving a nontrivial problem optimally, it typically implies something interesting and useful about the structure of the problem itself; there exists a local decision rule that one can use to construct optimal solutions." (Kleinberg 2014)

References

Kleinberg, Jon. Tardos, Eva. *Algorithm Design*. Pearson: India, 2014

Levine, Mellish, Reiter. (1992). Automatic generation of on-line documentation in the IDAS project. Association for Computational Linguistics: PA, USA.