Pages 161–166 in *Proceedings of the Fifth Conference of the European ACL*, Berlin, April 1991.

Generating Referring Expressions Involving Relations

Robert Dale
Department of Artificial Intelligence
and Centre for Cognitive Science
University of Edinburgh
Edinburgh EH8 9LW
Scotland
R.Dale@uk.ac.edinburgh

Nicholas Haddock
Hewlett Packard Laboratories
Filton Road
Stoke Gifford
Bristol BS12 6QZ
England
njh@com.hp.hpl.hplb

Abstract

In this paper, we review Dale's [1989] algorithm for determining the content of a referring expression. The algorithm, which only permits the use of one-place predicates, is revised and extended to deal with n-ary predicates. We investigate the problem of blocking 'recursion' in complex noun phrases and propose a solution in the context of our algorithm.

Introduction

In very simple language generation systems, there is typically a one-to-one relationship between entities known to the system and the linguistic forms available for describing those entities; in effect, each entity has a canonical name. In such systems, deciding upon the form of reference required in a given context requires at most choosing between a pronoun and the canonical name.[1]

As soon as a generation system has access to a knowledge base which contains richer knowledge about the entities in the domain, the system has to face the problem of deciding what particular properties of an entity should be used in describing it in a given context.[2] Producing a description which includes all of the known properties of the entity is likely to be both inefficient and misleading.

[1] We do not mean to imply, of course, that the decision as to whether or not to use a pronoun is simple.

[2] This problem exists quite independently of any considerations of the different **perspectives** that might be taken upon an entity, where, for example, one entity can be viewed from the perspective of being a father, a bicyclist and a teacher, with separate clusters of properties in each case. Even if the system is restricted to a single perspective upon each entity (as almost all language generation systems are), in any sophisticated knowledge base there will still be more information available about the entity than it is sensible to include in a description.

The core of the problem is finding a way of describing the intended referent that distinguishes it from other potential referents with which it might be confused. We refer to this problem as the **content determination** task. In this paper, we point out some limitations in an earlier solution proposed in Dale [1988, 1989], and discuss the possibilites of extending this solution by incorporating a use of constraints motivated by the work of Haddock [1987, 1988].

Generating Referring Expressions

The Principles of Reference

Dale [1988, 1989] presents a solution to the content determination task which is motivated by three principles of reference. These are essentially Gricean conversational maxims rephrased from the perspective of generating referring expressions:

1. The **principle of sensitivity** states that the referring expression chosen should take account of the state of the hearer's knowledge.
2. The **principle of adequacy** states that the referring expression chosen should be sufficient to identify the intended referent.
3. The **principle of efficiency** states that the referring expression chosen should provide no more information than is necessary for the identification of the intended referent.

The solution proposed in Dale [1988, 1989] focuses on the second and third of these principles of reference as constraints on the content determination task.

Distinguishing Descriptions

Other researchers (see, for example, [Davey 1978; Appelt 1985a]) have suggested that the process of determining the content of a referring expression should be governed by principles like those just described. Detailed algorithms for satisfying these requirements are rarely provided, however.

Suppose that we have a set of entities C (called the **context set**) such that $C = \{a_1, a_2, \ldots, a_n\}$ and our task is to distinguish from this context set some intended referent r where $r \in C$. Suppose, also, that each entity a_k is described in the system's knowledge base by means of a set of properties, $p_{k_1}, p_{k_2}, \ldots, p_{k_m}$.

In order to distinguish our intended referent r from the other entities in C, we need to find some set of properties which are together true of r, but of no other entity in C.[3] The linguistic realisation of this set of properties constitutes a **distinguishing description** (DD) of r with respect to the context C. A **minimal distinguishing description** is then the linguistic realisation of the smallest such set of properties.

An Algorithm to Compute Distinguishing Descriptions

Let L_r be the set of properties to be realised in our description; and let P_r be the set of properties known to be true of our intended referent r (we assume that P_r is non-empty). The initial conditions are thus as follows:

- $C_r = \{\langle$*all entities in the knowledge base*$\rangle\}$;
- $P_r = \{\langle$*all properties true of* $r\rangle\}$;
- $L_r = \{\}$

In order to describe the intended referent r with respect to the context set C_r, we do the following:

1. Check Success

   if $|C_r| = 1$ **then** return L_r as a DD
   elseif $P_r = \emptyset$ **then** return L_r as a non-DD
   else goto Step 2.

2. Choose Property

   for each $p_i \in P_r$ **do**: $C_{r_i} \leftarrow C_r \cap \{x | p_i(x)\}$
   Chosen property is p_j, where C_{r_j} is the smallest set.[4]
   goto Step 3.

3. Extend Description (wrt the chosen p_j)

   $L_r \leftarrow L_r \cup \{p_j\}$
   $C_r \leftarrow C_{r_j}$
   $P_r \leftarrow P_r - \{p_j\}$
   goto Step 1.

If we have a distinguishing description, a definite determiner can be used, since the intended referent is described uniquely in context. If the result is a non-distinguishing description, all is not lost: we can realise the description by means of a noun phrase of the form *one of the Xs*, where X is the realisation of the properties in L_r.[5] For simplicity, the remainder of this paper concentrates on the generation of distinguishing descriptions only; the extended algorithm presented later will simply fail if it is not possible to produce a DD.

The abstract process described above requires some slight modifications before it can be used effectively for noun phrase generation. In particular, we should note that, in noun phrases, the head noun typically appears even in cases where it does not have any discriminatory power. For example, suppose there are six entities on a table, all of which are cups although only one is red: we are then likely to describe that particular cup as as *the red cup* rather than simply *the red* or *the red thing*. Thus, in order to implement the above algorithm, we always first add to L that property of the entity that would typically be denoted by a head noun.[6] In many cases, this means that no further properties need be added.

Note also that Step 2 of our algorithm is non-deterministic, in that several properties may independently yield a context set of the same minimal size. For simplicity, we assume that one of these equally viable properties is chosen at random.

Some Problems

There are some problems with the algorithm just described.

As Reiter [1990:139] has pointed out, the algorithm does not guarantee to find a *minimal* distinguishing description: this is equivalent to the minimal set cover problem and is thus intractable as stated.

Second, the mechanism doesn't necessarily produce a *useful* description: consider the example offered by Appelt [1985b:6], where a speaker tells

[3] A similar approach is being pursued by Leavitt (personal communication) at CMU.

[4] In the terminology of Dale [1988, 1989], this is equivalent to finding the property with the greatest **discriminatory power**.

[5] One might be tempted to suggest that a straightforward indefinite, as in *an X*, could be used in such cases; this is typically not what people do, however.

[6] For simplicity, we can assume that this is that property of the entity that would be denoted by what Rosch [1978] calls the entity's **basic category**.

a hearer (whom she has just met on the bus) which bus stop to get off at by saying *Get off one stop before I do*. This may be a uniquely identifying description of the intended referent, but it is of little use without a supplementary offer to indicate the stop; ultimately, we require some computational treatment of the Principle of Sensitivity here.

Third, as has been demonstrated by work in psycholinguistics (for a recent summary, see Levelt [1989:129–134]), the algorithm does not represent what people seem to do when constructing a referring expression: in particular, people typically produce referring expressions which are redundant (over and above the inclusion of the head noun as discussed above). This fact can, of course, be taken to nullify the impact of the first problem described above.

We do not intend to address any of these problems in the present paper. Instead, we consider an extension of our basic algorithm to deal with relations, and focus on an orthogonal problem which besets any algorithm for generating DDs involving relations.

Relations and the Problem of 'Recursion'

Suppose that our knowledge base consists of a set of facts, as follows:

$$\{\mathsf{cup}(c_1), \mathsf{cup}(c_2), \mathsf{cup}(c_3), \mathsf{bowl}(b_1), \mathsf{bowl}(b_2), \mathsf{table}(t_1), \mathsf{table}(t_2), \mathsf{floor}(f_1), \mathsf{in}(c_1, b_1), \mathsf{in}(c_2, b_2), \mathsf{on}(c_3, f_1), \mathsf{on}(b_1, f_1), \mathsf{on}(b_2, t_1), \mathsf{on}(t_1, f_1), \mathsf{on}(t_2, f_1)\}$$

Thus we have three cups, two bowls, two tables and a floor. Cup c_1 is in bowl b_1, and bowl b_1 is on the floor, as are the tables and cup c_3; and so on. The algorithm described above deals only with one-place predicates, and says nothing about using **relations** such as $\mathsf{on}(b_1, f_1)$ as part of a distinguishing description. How can we extend the basic algorithm to handle relations? It turns out that this is not as simple as it might seem: problems arise because of the potential for infinite regress in the construction of the description.

A natural strategy to adopt for generating expressions with relations is that used by Appelt [1985a:108–112]. For example, to describe the entity c_3, our planner might determine that the predicate to be realized in our referring expression is the abstraction $\lambda x[\mathsf{cup}(x) \wedge \mathsf{on}(x, f_1)]$, since this complex predicate is true of only one entity, namely c_3. In Appelt's TELEGRAM, this results first in the choice of the head noun *cup*, followed by a recursive call to the planner to determine how f_1 should be described. The resulting noun phrase is then *the cup on the floor*.

In many cases this approach will do what is required. However, in certain situations, it will attempt to describe a referent in terms of itself and generate an infinite description.

For example, consider a very specific instance of the problem, which arises in a scenario of the kind discussed in Haddock [1987, 1988] from the perspective of interpretation. Such a scenario is characterised in the above knowledge base: we have two bowls and two tables, and one of the bowls is on one of the tables. Given this situation, it is felicitous to refer to b_2 as *the bowl on the table*. However, the use of the definite article in the embedded NP *the table* poses a problem for purely compositional approaches to interpretation, which would expect the embedded NP to refer uniquely in isolation.

Naturally, this same scenario will be problematic for a purely compositional approach to generation of the kind alluded to at the beginning of this section. Taken literally, this algorithm could generate an infinite NP, such as:[7]

> the bowl on the table which supports the bowl on the table which supports ...

Below, we present an algorithm for generating relational descriptions which deals with this specific instance of the problem of repetition. Haddock [1988] observes the problem can be solved by giving both determiners scope over the entire NP, thus:

$$(\exists!x)(\exists!y)\mathsf{bowl}(x) \wedge \mathsf{on}(x, y) \wedge \mathsf{table}(y)$$

In Haddock's model of interpretation, this treatment falls out of a scheme of incremental, left-to-right reference evaluation based on an incremental accumulation of constraints. Our generation algorithm follows Haddock [1988], and Mellish [1985], in using constraint-network consistency to determine the entities relating to a description (see Mackworth [1977]). This is not strictly necessary, since any evaluation procedure such as generate-and-test or backtracking, can produce the desired result; however, using network consistency provides a natural evolution of the existing algorithm, since this already models the problem in terms of incremental refinement of context sets. We conclude the paper by investigating the implications of our approach for the more general problem of recursive repetition.

[7] We ignore the question of determiner choice in the present paper, and assume for simplicity that definite determiners are chosen here.

A Constraint-Based Algorithm

Data Structures

We assume three global kinds of data structure.

1. The **Referent Stack** is a stack of referents we are trying to describe. Initially this stack is set to contain just the top-level referent:[8]

   $[\mathsf{Describe}(b_2, x)]$

   This means that the goal is to describe the referent b_2 in terms of predicates over the variable x.

2. The **Property Set** for the intended referent r is the set of facts, or predications, in the knowledge base relating to r; we will notate this as P_r. For example, given the knowledge base introduced in the previous section, the floor f_1 has the following Property Set:

   $$P_{f_1} = \{\mathsf{floor}(f_1), \mathsf{on}(c_3, f_1), \mathsf{on}(b_1, f_1), \mathsf{on}(t_1, f_1), \mathsf{on}(t_2, f_1)\}$$

3. A **Constraint Network** N will be viewed abstractly as a pair consisting of (a) a set of constraints, which corresponds to our description L, and (b) the context sets for the variables mentioned in L. The following is an example of a constraint network, viewed in these terms:

   $$\langle\{\mathsf{cup}(x), \mathsf{in}(x, y)\}, [C_x = \{c_1, c_2\}, C_y = \{b_1, b_2\}]\rangle$$

The Algorithm

For brevity, our algorithm uses the notation $N \oplus p$ to signify the result of adding the constraint p to the network N. Whenever a constraint p is added to a network, assume the following actions occur: (a) p is added to the set of constraints L; and (b) the context sets for variables in L are refined until their values are consistent with the new constraint.[9] Assume that every variable is initially associated with a context set containing all entities in the knowledge base.

In addition, we use the notation $[r \backslash v]p$ to signify the result of replacing every occurence of the constant r in p by the variable v. For instance,

$$[c_3 \backslash x]\mathsf{on}(c_3, f_1) = \mathsf{on}(x, f_1)$$

The initial conditions are as follows:

- $\mathsf{Stack} = [\mathsf{Describe}(r, v)]$
- $P_r = \{\langle \textit{all facts true of } r\rangle\}$
- $N = \langle\{\}, [C_v = \{\langle \textit{all entities}\rangle\}]\rangle$

Thus, initially there are no properties in L. As before, the problem of finding a description L involves three steps which are repeated until a successful description has been constructed:

1. We first check whether the description we have constructed so far is successful in picking out the intended referent.
2. If the description is not sufficient to pick out the intended referent, we choose the most useful fact that will contribute to the description.
3. We then extend the description with a constraint representing this fact, and add Describe goals for any constants relating to the constraint.

The essential use of constraints occurs in Step 2 and 3; the detail of the revised algorithm is shown in Figure 1.

An Example

There is insufficient space to go through an example in detail here; however, we summarise some steps for the problematic case of referring to b_2 as the *the bowl on the table*.[10] For simplicity here, we assume our algorithm will always choose the head category first. Thus, we have the following constraint network after one iteration through the algorithm:

$$N = \langle\{\mathsf{bowl}(x)\}, [C_x = \{b_1, b_2\}]\rangle$$

Let us suppose that the second iteration chooses $\mathsf{on}(b_2, t_1)$ as the predication with which to extend our description. When integrated into the constraint network, we have

$$N = \langle\{\mathsf{bowl}(x), \mathsf{on}(x, y)\}, [C_x = \{b_1, b_2\}, C_y = \{f_1, t_1\}]\rangle$$

Note that the network has determined a set for y which does not include the second table t_2 because it is not known to support anything.

Given our head-category-first strategy, the third iteration through the algorithm adds $\mathsf{table}(t_1)$ as a constraint to N, to form the new network

[8] We represent the stack here as a list, with the top of the stack being the left-most item in the list.

[9] We do not address the degree of network consistency required by our algorithm. However, for the examples treated in this paper, a node and arc consistency algorithm, such as Mackworth's [1977] AC-3, will suffice. (Haddock [1991] investigates the sufficiency of such low-power techniques for noun phrase interpretation.) We assume that our algorithm handles constants as well as variables within constraints.

[10] Again, we ignore the question of determiner choice and assume definites are chosen.

Note that in Steps 1, 2 and 3, r and v relate to the current $\mathsf{Describe}(r, v)$ on top of the stack.

1. Check Success

   if Stack is empty **then** return L as a DD
   elseif $|C_v| = 1$ **then** pop Stack & **goto** Step 1
   elseif $P_r = \emptyset$ **then** fail
   else goto Step 2

2. Choose Property

   for each property $p_i \in P_r$ **do**
   $p_i' \leftarrow [r \backslash v]p_i$
   $N_i \leftarrow N \oplus p_i'$
   Chosen predication is p_j, where N_j contains the smallest set C_v for v.
   goto Step 3

3. Extend Description (w.r.t. the chosen p)

   $P_r \leftarrow P_r - \{p\}$
   $p \leftarrow [r \backslash v]p$
   for every other constant r' in p **do**
   associate r' with a new, unique variable v'
   $p \leftarrow [r' \backslash v']p$
   push $\mathsf{Describe}(r', v')$ onto Stack
   initialise a set $P_{r'}$ of facts true of r'
   $N \leftarrow N \oplus p$
   goto Step 1

Figure 1: A Constraint-Based Algorithm

$$N = \langle \{\mathsf{bowl}(x), \mathsf{on}(x,y), \mathsf{table}(y)\}, [C_x = \{b_2\}, C_y = \{t_1\}] \rangle$$

After adding this new constraint, f_1 is eliminated from C_y. This leads to the revision of to C_x, which must remove every value which is not on t_1.

On the fourth iteration, we exit with the first component of this network, L, as our description; we can then realize this content as *the bowl on the table*.

The Problem Revisited

The task of referring to b_2 in our knowledge base is something of a special case, and does not illustrate the nature of the general problem of recursion. Consider the task of referring to c_1. Due to the non-determinism in Step 2, our algorithm might either generate the DD corresponding to *the cup in the bowl on the floor*, or it might instead get into an infinite loop corresponding to *the cup in the bowl containing the cup in the bowl containing ...* The initial state of the referent stack and c_1's property set will be:

$$\begin{aligned} \mathsf{Stack} &= [\mathsf{Describe}(c_1, x)] \\ P_{c_1} &= \{\mathsf{cup}(c_1), \mathsf{in}(c_1, b_1)\} \end{aligned}$$

At the beginning of the fourth iteration the algorithm will have produced a partial description corresponding to *the cup in the bowl*, with the top-level goal to uniquely distinguish b_1:

$$\begin{aligned} \mathsf{Stack} &= [\mathsf{Describe}(b_1, y), \mathsf{Describe}(c_1, x)] \\ P_{c_1} &= \emptyset \\ P_{b_1} &= \{\mathsf{in}(c_1, b_1), \mathsf{on}(b_1, f_1)\} \\ N &= \langle \{\mathsf{cup}(x), \mathsf{in}(x, y), \mathsf{bowl}(y)\}, \\ &\quad [C_x = \{c_1, c_2\}, C_y = \{b_1, b_2\}] \rangle \end{aligned}$$

Step 2 of the fourth iteration computes two networks, for the two facts in P_{b_1}:

$$\begin{aligned} N_1 &= N \oplus \mathsf{in}(c_1, y) \\ &= \langle \{\mathsf{cup}(x), \mathsf{in}(x, y), \mathsf{bowl}(y), \mathsf{in}(c_1, y)\}, \\ &\quad [C_x = \{c_1\}, C_y = \{b_1\}] \rangle \\ N_2 &= N \oplus \mathsf{on}(y, f_1) \\ &= \langle \{\mathsf{cup}(x), \mathsf{in}(x, y), \mathsf{bowl}(y), \mathsf{on}(y, f_1)\}, \\ &\quad [C_x = \{c_1\}, C_y = \{b_1\}] \rangle \end{aligned}$$

Since both networks yield singleton sets for C_y, the algorithm might choose the property $\mathsf{in}(c_1, b_1)$. This means extending the current description with a constraint $\mathsf{in}(z, y)$, and stacking an additional commitment to describe c_1 in terms of the variable z. Hence at the end of the fourth iteration, the algorithm is in the state

$$\begin{aligned} \mathsf{Stack} &= [\mathsf{Describe}(c_1, z), \mathsf{Describe}(b_1, y), \\ &\quad \mathsf{Describe}(c_1, x)] \\ P_{c_{1_x}} &= \emptyset \\ P_{b_1} &= \{\mathsf{on}(b_1, f_1)\} \\ P_{c_{1_z}} &= \{\mathsf{cup}(c_1), \mathsf{in}(c_1, b_1)\} \\ N &= \langle \{\mathsf{cup}(x), \mathsf{in}(x, y), \mathsf{bowl}(y), \mathsf{in}(z, y)\}, \\ &\quad [\ldots] \rangle \end{aligned}$$

and may continue to loop in this manner.

The general problem of infinite repetition has been noted before in the generation literature. For example, Novak [1988:83] suggests that

> [i]f a two-place predicate is used to generate the restrictive relative clause, the second object of this predicate is characterized simply by its properties to avoid recursive reference as in *the car which was overtaken by the truck which overtook the car.*

Davey [1978], on the other hand, introduces the notion of a CANLIST (the Currently Active Node List) for those entities which have already been mentioned in the noun phrase currently under construction. The generator is then prohibited from describing an entity in terms of entities already in the CANLIST.

In the general case, these proposals appear to be too strong. Davey's restriction would seem to be the weaker of the two, but if taken literally, it will nevertheless prevent legitimate cases of bound-variable anaphora within an NP, such as *the man$_i$ who ate the cake which poisoned him$_i$*. We suggest the following, possibly more general heuristic: do not express a given piece of information more than once within the same NP. For our simplified representation of contextual knowledge, exemplified above, we could encode this heuristic by stipulating that any fact in the knowledge base can only be chosen once within a given call to the algorithm. So in the above example, once the relation $\mathsf{in}(c_1, b_1)$ has been chosen from the initial set P_{c_1}—in order to constrain the variable x—it is no longer available as a viable contextual constraint to distinguish b_1 later on. This heuristic will therefore block the infinite description of c_1. But as desired, it will admit the bound-variable anaphora mentioned above, since this NP is not based on repeated information; the phrase is merely self-referential.

Conclusion

We have shown how the referring expression generation algorithm presented in Dale [1988, 1989] can be extended to encompass the use of relations, by making use of constraint network consistency. In the context of this revised generation procedure we have investigated the problem of blocking the production of infinitely recursive noun phrases, and suggested an improvement on some existing approaches to the problem. Areas for further research include the relationship of our approach to existing algorithms in other fields, such as machine learning, and also its relationship to observed characteristics of human discourse production.

Acknowledgements

The work reported here was prompted by a conversation with Breck Baldwin. Both authors would like to thank colleagues at each of their institutions for numerous comments that have improved this paper.

References

Appelt, Douglas E [1985a] *Planning English Sentences*. Cambridge: Cambridge University Press.

Appelt, Douglas E [1985b] Planning English Referring Expressions. *Artificial Intelligence*, **26**, 1–33.

Dale, Robert [1988] Generating Referring Expressions in a Domain of Objects and Processes. PhD Thesis, Centre for Cognitive Science, University of Edinburgh.

Dale, Robert [1989] Cooking up Referring Expressions. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver BC, pp68–75.

Davey, Anthony [1978] *Discourse Production*. Edinburgh: Edinburgh University Press.

Haddock, Nicholas J [1987] Incremental Interpretation and Combinatory Categorial Grammar. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, pp.

Haddock, Nicholas J [1988] Incremental Semantics and Interactive Syntactic Processing. PhD Thesis, Centre for Cognitive Science, University of Edinburgh.

Haddock, Nicholas J [1991] Linear-Time Reference Evaluation. Technical Report, Hewlett Packard Laboratories, Bristol.

Levelt, Willem J M [1989] *Speaking: ¿From Intention to Articulation*. Cambridge, Mass.: MIT Press.

Mackworth, Alan K [1977] Consistency in Networks of Relations. *Artificial Intelligence*, **8**, 99–118.

Mellish, Christopher S [1985] *Computer Interpretation of Natural Language Descriptions*. Chichester: Ellis Horwood.

Novak, Hans-Joachim [1988] Generating Referring Phrases in a Dynamic Environment. Chapter 5 in M Zock and G Sabah (eds), *Advances in Natural Language Generation*, Volume 2, pp76–85. London: Pinter Publishers.

Reiter, Ehud [1990] Generating Appropriate Natural Language Object Descriptions. PhD thesis, Aiken Computation Laboratory, Harvard University.

Rosch, Eleanor [1978] Principles of Categorization. In E Rosch and B Lloyd (eds), *Cognition and Categorization*, pp27–48. Hillsdale, NJ: Lawrence Erlbaum Associates.