

Generating Referring Expressions: Boolean Extensions of the Incremental Algorithm

Kees van Deemter, ITRI,
University of Brighton, UK
Kees.van.Deemter@itri.brighton.ac.uk

This paper brings a logical perspective to the generation of referring expressions, addressing the incompleteness of existing algorithms in this area. After studying references to individual objects, we discuss references to sets, including Boolean descriptions that make use of negated and disjoined properties. To guarantee that a distinguishing description is generated whenever such descriptions exist, the paper proposes generalizations and extensions of the Incremental Algorithm of Dale and Reiter (1995).

1. Introduction

Generation of referring expressions (GRE) is a key task of most Natural Language Generation (NLG) systems (e.g., Reiter and Dale (2000), section 5.4). Regardless of the type of Knowledge Base (KB) forming the input to the generator, many objects will not be designated in it via an ordinary proper name. A person like Mr. Jones, for example, may be designated using an artificial name like '#Jones083' if the name 'Jones' alone is not uniquely distinguishing. The same is true for a piece of furniture, a tree or an atomic particle, for instance, for which no proper name is in common usage at all, or (in most cases) if the generator tries to refer to an entire set of objects. In all such cases, the generator has to 'invent' a description that enables the hearer to identify the intended referent. In the case of Mr. Jones, for example, the program could identify him by providing his full name and address; in the case of a tree, some longer description may be necessary. Henceforth, we will call the intended referent of the description the *target* of the GRE algorithm.

The question that we set out to answer is whether existing GRE algorithms produce adequate descriptions whenever such descriptions exist: in short, whether these algorithms are, as we shall say, *complete*. The paper brings a degree of formal precision to this issue and reveals a number of reasons why current GRE algorithms are incomplete; we sketch remedies and discuss their consequences in terms of linguistic coverage and computational tractability. We take the Incremental Algorithm (Dale and Reiter 1995) to represent the state of the art in this area, and we minimize the deviations from this algorithm. As a result, this paper might be read as an investigation into how widely the ideas underlying the Incremental Algorithm can be used, and the extent to which they may be generalized. The main generalization that we will investigate involves complex Boolean combinations of properties, that is, descriptions that involve more than a merely intersective (i.e., logically conjunctive) combination of properties. Such generalizations are natural because the properties involved are implicitly present in the KB, as we will explain; they become especially relevant when the algorithms are also general-

ized to generate references to sets, rather than individual objects. But, before we arrive at these generalizations, we will identify and confront a number of cases in which current GRE algorithms are incomplete even with respect to merely intersective descriptions.

In this paper, we will deal with ‘first mention’ descriptions only, (unlike Dale 1992, chapter 5; Mittal et al. 1998; Kibble 1999), assuming that the information used for generating the description is limited to a KB containing complete information as to which properties are true of each object. Also, we focus on ‘one shot’ descriptions, disregarding cases where an object is described through its relations with other objects (Dale and Haddock 1991, Horacek 1997, Krahmer et al. 2001). More crucially, we follow Dale and Reiter (1995) in focusing on the semantic content of a description (i.e., the problem of *content determination*, for short), assuming that any combination of properties can be expressed by the NLG module responsible for linguistic realization. This modular approach allows us to separate logical aspects of generation (which are largely language-independent) from purely linguistic aspects, and it allows the realization module to base its decisions on complete information as to which combination of properties is to be realized. Accordingly, when we write ‘Generation of Referring Expressions’ or GRE, we will refer specifically to determination of the semantic content of a description. Analogously, the word ‘description’ will refer to the semantic content of a linguistic expression only. Note that our modular approach makes it unnatural to assume that a description is always expressed by a single noun phrase: If several sentences are needed then so be it.

After summarizing the Incremental Algorithm in section 2, section 3 will take a closer look at the algorithm in its standard, ‘intersective’ form, in which it identifies an object by intersecting a number of atomic properties. We discuss cases in which this algorithm fails to find an adequate description even though such a description exists, and propose a number of possible remedies. Having established a completeness result for a version of the intersective Incremental algorithm, we turn to questions of completeness that involve more complex Boolean combinations in Section 4. Section 5 summarizes the main results of our exploration and puts them in perspective.

2. Dale and Reiter (1995): the Incremental Algorithm

The Incremental Algorithm of Dale and Reiter (1995) singles out a target object from among some larger domain of entities. This is done by logically conjoining a number of properties found in a part of the KB that represents information shared between speaker and hearer. The authors observed that the problem of finding a (‘Full Brevity’) description that contains the *minimum* number of properties is computationally intractable (i.e., NP Hard). They combined this with the known fact that people often produce non-minimal descriptions anyway (e.g., Pechman 1989). Accordingly, they proposed an algorithm that only *approximates* Full Brevity, while being of only linear complexity. Our summary of the algorithm will gloss over many details, while still allowing us to discuss completeness. In particular, we disregard any special provisions that might be made for the selection of head nouns because, arguably, this has to involve realizational issues.¹

The Incremental Algorithm produces a set L of properties P_1, \dots, P_n such that their logical conjunction forms a ‘distinguishing description’ (Dale 1989) of the target object r .

¹ Cf. Dale and Reiter (1995), where head nouns are taken into account during content determination. Head nouns can also be selected during linguistic realization or by interleaving of content determination and realization (e.g., Horacek 1997, Stone and Webber 1998, Krahmer and Theune (1999)).

In other words, writing $[[Q]]$ for the extension of Q (i.e., the set of objects that have the property Q), the intersection $[[P_1]] \cap \dots \cap [[P_n]]$ must equal the singleton set $\{r\}$. It is a ‘hillclimbing’ algorithm, which finds better and better approximations of the target set $\{r\}$ by accumulating more and more properties, which is why the algorithm is called Incremental. There is no backtracking. Consequently, if some property P_i in L is made redundant by later additions (i.e., when $([[P_1]] \cap \dots \cap [[P_i - 1]] \cap [[P_i + 1]] \cap \dots \cap [[P_n]]) \subseteq [[P_i]]$) then P_i is retained as a member of L nevertheless.

In the full algorithm, (see below, D&R_{Att}) properties are analysed as pairs consisting of an Attribute and a Value. Attributes are ordered in a list \mathbb{A} . If A_i precedes A_j in \mathbb{A} then A_i is ‘more preferred than’ A_j ; as a consequence, A_i will be considered before A_j by the algorithm. Suppose r is the target object, and \mathcal{D} (the ‘domain’) is the set of elements from which r is to be selected. The algorithm iterates through \mathbb{A} ; for each Attribute A_i , it checks whether specifying a Value for that Attribute would rule out at least one object that has not already been ruled out; if so, the Attribute is added to L , with a suitable Value (FindBestValue, below). C is the set of ‘confusables’ at any given stage of the algorithm.² Objects that are ruled out are removed from C . The process of expanding L and contracting C continues until $C = \{r\}$; if and when this condition is met, L is a distinguishing set of properties.

For easy generalizability, the algorithm will be cast in set-theoretic terms. We first present a version that focuses on *properties*, without separating these into Attributes and Values, and assume the properties themselves are ordered in a list \mathbb{P} (cf., Reiter and Dale 2000). This version of the algorithm will be called D&R_{Prop}, or D&R when there is no risk of confusion. We assume that the domain contains one or more objects other than the target object, the so-called distractors: thus, $r \in \mathcal{D}$ but $\{r\} \neq \mathcal{D}$.

```

L := ∅ { L is initialized to the empty set }
C := D { C is initialized to the domain }

For each Pi ∈ P do

  If r ∈ [[Pi]] & C ⊄ [[Pi]] { Pi removes distractors from C } then do

    L := L ∪ { Pi } { Property Pi is added to L }
    C := C ∩ [[Pi]] { All elements outside [[Pi]] are removed }
    If C = { r } then Return L { Success }

Return Failure { All prop's in P have been tested, and still C ≠ { r } }

```

Assuming (like Dale and Reiter 1995) that the tests in the body of the loop take some constant amount of time, the worst-case running time is in the order of n_a (i.e., $O(n_a)$) where n_a is the total number of properties. So, the algorithm has only linear complexity.

A slightly closer approximation of Full Brevity can be achieved if Attributes and Values are separated (Dale and Reiter 1995), allowing the algorithm to choose the ‘best’ Value for each Attribute. Given an Attribute, FindBestValue selects the Value that removes most distractors while still including the target r . If no Value includes r , the function returns nil. In case of a tie (i.e., no Value removes more distractors than all others), FindBestValue chooses the least specific of the contestants. For example, when ‘dog’

² Thus, C contains r , unlike (Dale and Reiter 1995). The difference is purely presentational.

rules out as many distractors as ‘chihuahua’, ‘chihuahua’ cannot be chosen. \mathbb{A} is the list of Attributes; L is the set of Attribute/Value combinations returned by the algorithm. A further notational convention will be useful: Values will be identified by two indices, the first of which identifies the Attribute. Thus, to denote Value j of Attribute A_i , we write $V_{i,j}$. This version of the algorithm will be called $D\&R_{Att}$. The initializations of L and \mathcal{D} are omitted for brevity.

For each $A_i \in \mathbb{A}$ do

$V_{i,j} = \text{FindBestValue}(r, A_i)$

If $r \in [[V_{i,j}]] \ \& \ C \not\subseteq [[V_{i,j}]]$ then do

$L := L \cup \{V_{i,j}\}$

$C := C \cap [[V_{i,j}]]$

If $C = \{r\}$ then Return L

Return Failure

We will switch back and forth between $D\&R$ and $D\&R_{Att}$ depending on what is at stake. Like $D\&R$, $D\&R_{Att}$ has linear complexity. This can be made precise in the following way.³ If the running time of a call of $\text{FindBestValue}(r, A_i)$ is a constant times the number of Values of the Attribute A_i , then the worst-case running time of $D\&R_{Att}$ is $O(n_v n_a)$, where n_a equals the number of Attributes in the language and n_v the average number of Values of all Attributes.

3. Completeness of the Incremental Algorithm

Some new definitions will be useful. A GRE algorithm is *successful* with respect to a given situation (i.e., with respect to a KB and a target) if it produces a distinguishing description of r in that situation. We will call an algorithm *complete* if it is successful in every situation in which a distinguishing description exists. Success is not always possible: the properties in the KB may not be sufficient for individuating a given object; such no-win situations will not be held against an algorithm.

The Incremental Algorithm generates descriptions that contain set intersection as their only Boolean operation. We define a GRE algorithm to be **intersectively complete** if it has the following property: whenever an object can be characterized by intersecting a finite number of properties, the algorithm will find such an intersection. We would like to prove the Incremental Algorithm to be intersectively complete, but we shall meet a few obstacles before we get there.

3.1 Completeness and Overlapping Values

One assumption without which the Incremental Algorithm cannot be proven to be intersectively complete concerns the semantic relation between different Values of a given Attribute: their extensions should not ‘overlap’ in the following precise sense.

³ Dale and Reiter arrived at linearity via the difficult concept of *typical* running time. They assumed that, typically, n_l (i.e., the number of properties in the description) is proportional to the number of Attributes examined by the algorithm (E. Reiter, *p.c.*). This allowed them to argue that the typical running time is $O(n_d n_l)$, where n_d is the number of distractors (Dale and Reiter 1995, section 3.1). Our own worst-case assessment does not rely on assumptions of typicality.

Values $V_{i,j}$ and $V_{i,k}$ (and equally, their extensions) *overlap* iff $V_{i,j} \cap V_{i,k}$, $V_{i,j} - V_{i,k}$, and $V_{i,k} - V_{i,j}$ are all non-empty.

(If $V_{i,j}$ and $V_{i,k}$ do *not* overlap then either $[[V_{i,j}]] \subseteq [[V_{i,k}]]$, or $[[V_{i,k}]] \subseteq [[V_{i,j}]]$, or $[[V_{i,j}]]$ and $[[V_{i,k}]]$ have an empty intersection.) Values can overlap for different reasons. Some Attributes, like COLOUR for example, have ‘vague’ Values such as RED, ORANGE, etc., which may be modeled as overlapping: some objects may count as both red and orange. Also, Values may derive from particular parts or aspects of an object; for example, if an object counts as METAL (PLASTIC) because it has *some* METAL (PLASTIC) parts then an object may be listed as both METAL and PLASTIC. Further examples arise if the KB models relations through unanalysed properties. For example, a desk, or a particular *type* of desk, can stand in a given relation (e.g., ‘being considered by’ or ‘being bought by’) to more than one other company. To illustrate the problems arising from overlapping Values, consider a KB modeling which customer bought which types of desks, and where $C = \{a, b, c, d, e, f\}$:

BOUGHT-BY: Philips ($\{a, b, e\}$), Sony ($\{a, c, d, f\}$),
COLOUR: Brown ($\{a, b\}$), Yellow ($\{c, d\}$)

(Desks of types a, b and e were bought by Philips, and so on. Note that desks of type a were bought by two different companies.) Suppose a is the target, while the Attribute BOUGHT-BY is more ‘preferred’ than COLOUR. The Value ‘Philips’ (being the BestValue of BOUGHT-BY, since it removes more distractors than the Value ‘Sony’) is chosen first, reducing the initial set C to $\{a, b, e\}$. Now, the algorithm is doomed to end in Failure, since the different Values of COLOUR are unable to remove the unwanted b without also sacrificing a . None of this can be corrected, since the algorithm does not use backtracking. Note that a uniquely identifying description of a would have been possible if only ‘Sony’ had been chosen instead of ‘Philips’, leading to a description like ‘the brown desk bought by Sony’. The algorithm does not just fail: it fails in a situation where Success was perfectly achievable!

How can this limitation be remedied? One might introduce a limited kind of backtracking, which “remembers” where the algorithm has encountered overlapping Values and, when it results in Failure, goes back to the last-encountered situation where it has made a choice between overlapping Values; if this does not lead to Success, the algorithm tracks back to the previous choice situation, and so on until no more choice situations are left (Failure) or a distinguishing description has been reached (Success). Unfortunately, this algorithm becomes intractable if Values overlap too often: in the worst case, we are back to having to check all combinations of properties.

A simpler and computationally more efficient algorithm would include *all* overlapping Values that are true of the target while also removing some distractors. This could be done as follows: whenever a Value $V_{i,j}$ of an Attribute A_i is selected for inclusion in L , search for other Values of the same Attribute that have the target r as an element; if such a Value $V_{i,k}$ is found, check whether it stands in the subset relation to $V_{i,j}$ (i.e., either $[[V_{i,j}]] \subseteq [[V_{i,k}]]$ or $[[V_{i,k}]] \subseteq [[V_{i,j}]]$); if not, then include $V_{i,k}$ as well; next, search for yet another Value $V_{i,l}$ of the same Attribute that has r as an element, and include $V_{i,l}$ if it does not stand in the subset relation to $V_{i,j}$ or $V_{i,k}$; and so on until no other Values of A_i exist that have r as an element; then move on to the next Attribute. This algorithm has a worst-case running time of $O(n_a n_v^2)$.⁴

⁴ This assumes that, once FindBestValue has found a Value V that removes distractors, one may need to inspect all the other Values of the same Attribute to find Values overlapping with V . Shortcuts are

In our example, this algorithm would produce a set consisting of the properties ‘bought by Sony’ and ‘bought by Philips’, which can be realized as ‘the desk bought by Sony *and* by Philips’; if we change the example by letting Philips buy *c* as well as *a*, the algorithm will go on to select the property ‘brown’, resulting in a set of properties that may be realized as ‘the brown desk bought by Sony and by Philips’. Such descriptions appear to be quite natural. One might even argue, on Gricean grounds (Grice 1975), that identifying *a* simply as being bought by Philips can give rise to the false implicature that *a* was *not* bought by Sony. This suggests that the proposed algorithm might also be empirically more accurate than the one using limited backtracking provided, of course, properties are properly aggregated (e.g., Dalianis and Hovy 1996).

3.2 Assumptions concerning infinite sets

To prove intersective completeness, certain assumptions concerning the cardinality of sets need to be made. To give an extreme example, suppose one wanted to refer to a real number that does not have a ‘proper name’ (unlike, e.g., π); then the class of potentially useful properties is so vast that no GRE algorithm can take them all into consideration. As long as the number of **properties** (i.e., Attribute/Value combinations) is *denumerably* infinite (Kleene 1971) then only termination becomes problematic: *if* a uniquely referring description $[[P_1]] \cap \dots \cap [[P_n]]$ exists, *then* the algorithm will find one in finite time, since each of the *n* properties in the description will be found in finite time; if no distinguishing description exists, however, the algorithm never terminates. In the less likely case where the set of properties is *nondenumerably* infinite, (i.e., it does not stand in a 1-1 relation to any set of natural numbers) completeness becomes problematic as well, since it is impossible for the algorithm to consider all properties, hence successful combinations may be overlooked (cf., Kleene 1971, p. 6-8).

Infinity of the set of **distractors** results in a different problem. The key question is whether there exists an effective procedure for removing distractors (i.e., for calculating $C \cap [[P_i]]$). If no such procedure exists, the Incremental Algorithm can only be applied after a property has been found that cuts down the set of distractors to a manageable size. To be on the safe side when we prove completeness, we will assume that the set of properties is at most denumerably infinite, while the set of distractors is finite. These assumptions are harmless in connection with present NLG systems, all of which work with relatively small sets. It is unclear how *human* speakers cope with large sets of properties and/or distractors, but this question goes beyond our present concerns.

3.3 Proving Intersective Completeness

Based on these considerations, we prove intersective completeness under some assumptions concerning infinity and overlapping Values. We first deal with D&R, then with the more complex D&R_{Att}.

Theorem 1. Completeness of D&R. Suppose there are at most denumerably many properties, and finitely many (one or more) distractors. Then if an object can be individuated by intersecting a finite number of properties, D&R will find such an intersection.

Proof. Suppose, $[[Q_1]] \cap \dots \cap [[Q_m]] = \{r\}$, where the properties Q_1, \dots, Q_m occur in \mathbb{P} in the order indicated by the subscripts. Now either D&R returns *Success* *before* it has inspected all of Q_1, \dots, Q_m , or it reaches the

possible if Values are stored using a structure that reflects their semantic relationships.

point where all of Q_1, \dots, Q_m have been inspected. This does not mean that all of Q_1, \dots, Q_m have necessarily been included in L , since other properties in \mathbb{P} may have been selected which cause some of Q_1, \dots, Q_m not to remove any distractors. Yet, when all of Q_1, \dots, Q_m have been inspected **Success** must have been achieved. To see this, let Des_i be the description that results after processing (i.e., inspecting and possibly including) Q_i . Then a proof by induction over i shows that $[[Des_i]] \subseteq [[Q_1]] \cap \dots \cap [[Q_i]]$, for all $i \leq m$. (Consider the basic case, where $i = 1$, and assume that $Q_1 \notin [[Des_1]]$; Q_1 was rejected, so it did not remove any distractors, hence $[[Des_1]] \subseteq [[Q_1]]$. The induction step is analogous.) It follows that $[[Des_m]] \subseteq [[Q_1]] \cap \dots \cap [[Q_m]] = \{r\}$. But $r \in [[Des_m]]$, so $[[Des_m]] = \{r\}$. \square

Theorem 2. Completeness of D&R_{Att}. Assume **(1)** Attributes have no overlapping Values (see section 3.1), and **(2)** there are at most denumerably many Attributes and Values, and finitely many (one or more) distractors. Then if an object can be individuated by intersecting a finite number of properties, D&R_{Att} will find such an intersection.

Proof. Given Assumption **(1)**, if D&R is complete, then so is D&R_{Att}. To see this, let BV abbreviate $\text{FindBestValue}(r, A_i)$. Suppose there is a Value $V_{i,j}$ of Attribute A_i that leads to a distinguishing description whereas BV does not. Then a contradiction is derived as follows. For certain $V_{i,a_1}, \dots, V_{i,a_n}$

$$\begin{aligned} [[V_{i,a_1}]] \cap \dots \cap [[V_{i,a_n}]] \cap [[V_{i,j}]] &= \{r\}, \text{ whereas} \\ [[V_{i,a_1}]] \cap \dots \cap [[V_{i,a_n}]] \cap [[BV]] &\neq \{r\}. \end{aligned}$$

So, either **(i)** $r \notin BV$ or **(ii)** there exists $x \neq r$ for which $x \in [[V_{i,a_1}]] \cap \dots \cap [[V_{i,a_n}]] \cap BV$. But case **i** contradicts the definition of FindBestValue (see section 2); case **ii**, on the other hand, implies that

$$\begin{aligned} x &\in [[V_{i,a_1}]] \cap \dots \cap [[V_{i,a_n}]] \cap [[BV]], \text{ whereas} \\ x &\notin [[V_{i,a_1}]] \cap \dots \cap [[V_{i,a_n}]] \cap [[V_{i,j}]], \end{aligned}$$

hence $x \in [[BV]]$ while $x \notin [[V_{i,j}]]$. But $r \in [[BV]] \cap [[V_{i,j}]]$, so $[[BV]]$ and $[[V_{i,j}]]$ are not disjoint. Consequently, by Assumption **(1)**, case **ii** implies that $[[V_{i,a_1}]] \cap \dots \cap [[V_{i,a_n}]] \cap [[V_{i,j}]]$ is a real subset of $[[V_{i,a_1}]] \cap \dots \cap [[V_{i,a_n}]] \cap [[BV]]$, contradicting the fact that FindBestValue only prefers a more general Value (i.e., BV) over a more specific one (i.e., $V_{i,j}$) if it removes the same distractors. \square

4. Generalizing the Incremental Algorithm

Both versions of the Incremental Algorithm have been proven to be *intersectively* complete. Now we widen the issue to include all other Boolean combinations, involving negation (i.e., complementation) and disjunction (i.e., union).⁵ This is natural, since properties expressed by Boolean combinations are *implicit* in the KB: If the KB lists the property POODLE and the property ALSACIAN, then it implicitly contains the property

⁵ The well-known correspondence between set-theoretical operations like *set union* and propositional operators like *disjunction* will allow us to ‘mix and match’ the two terminologies.

of being *either* a poodle *or* an alsacian. This move will, however, only have its full impact when we also widen the issue to reference to *sets* of objects.

In the new setting, it will be useful to generalize our earlier notion of *intersective* completeness (section 3), calling a GRE algorithm **Boolean complete** iff it finds a Boolean description of a set whenever one can be given on the basis of the properties in the KB.

4.1 Describing sets

Generating descriptions is even more important if the target is a set than if it is a single object: even if the objects in the set have proper names, the set as a whole may lack a name (and enumerating the objects may be cumbersome). Yet, reference to sets has long been disregarded in NLG. In this section, we sketch generalizations of D&R that produce descriptions of sets. For starters, the algorithm $D\&R_{\text{plural}}$ finds intersections $P_1 \cap \dots \cap P_n$ of atomic properties P_1, \dots, P_n whose extension equals a given target set S (van Deemter 2000). Since S may or may not be a singleton, $D\&R_{\text{plural}}$ subsumes D&R. As before, we assume a nonempty set of distractors, that is, $S \subseteq \mathcal{D}$ but $S \neq \mathcal{D}$.⁶

$L := \emptyset$

$C := \mathcal{D}$

For each $P_i \in \mathbb{P}$ do

If $S \subseteq [[P_i]]$ & $C \not\subseteq [[P_i]]$ then do

$L := L \cup \{P_i\}$

$C := C \cap [[P_i]]$

If $C = S$ then Return L { Success }

Return Failure { All properties in \mathbb{P} have been tested, yet $C \neq S$ }

Note that S takes the place of the target object r in the earlier algorithms; the process of expanding L and contracting C continues until $C = S$. Because this is basically the same algorithm as D&R, it has the same computational complexity of $O(n_a)$, where n_a is the cardinality of \mathbb{P} .

$D\&R_{\text{plural}}$ characterizes a set by scrutinizing its elements. This does not work for properties like ‘being of the same age’, which crucially pertain to *sets* of objects (cf., Stone 2000). The algorithm can, however, be generalized to cover such cases if we initialize C not to \mathcal{D} but to the *powerset* of \mathcal{D} , after which the algorithm selects properties of *sets*, removing from $\mathcal{P}(\mathcal{D})$ all those sets for which the property is false. For example, selection of ‘being of the same age’ removes all those sets whose elements are not of the same age as each other; selection of ‘forming a football team’ removes all sets that do not make up a football team, etc. As a result, the algorithm generates descriptions of *sets* of collective entities (i.e., sets of sets). In this way, descriptions such as ‘those teams all of whose members are of the same age’ can be generated. In this collective version of $D\&R_{\text{plural}}$, the target S is a set of sets; \mathbb{P} is a list of properties of sets, so if $P_i \in \mathbb{P}$ then $[[P_i]]$ is also a set of sets. As in the case of distributive properties, describing one entity (i.e., in this case, one set) is a special case of describing a set of entities.

⁶ We continue to disregard special provisions for head nouns (cf., note 1). Note, however, that a head noun must be selected that suits *every* element of the target set.

$$L := \emptyset$$

$$C := \mathcal{P}(\mathcal{D})$$

For each $P_i \in \mathbb{P}$ do

If $S \subseteq \llbracket P_i \rrbracket$ & $C \not\subseteq \llbracket P_i \rrbracket$ Then do

$$L := L \cup \{P_i\}$$

$$C := C \cap \llbracket P_i \rrbracket$$

If $C = S$ then Return L { Success }

Return Failure

Once again, these adaptations leave the algorithm structurally unchanged: sets replace objects throughout. Yet, they cause the complexity of the algorithm to become exponential, since testing whether $C = S$ involves inspecting all elements of C , of which there can be up to 2^{n_d} (where n_d is the cardinality of the domain \mathcal{D}).

This algorithm can also be applied to distributive properties if these are upgraded to the level of sets: Let a newfangled distributive property be true of a set iff the property (in ordinary parlance) is true of all its elements (Kamp and Reyle 1993, p. 338). This requires that the target S is always cast as a set of sets, even if it is viewed distributively. For example, if a set of players, say a , b and c , are to be characterized as a *collection* (e.g., to say that they won as a team of three) then $S = \{\{a, b, c\}\}$; if they are to be characterized *distributively* (e.g., to say that each of them has the flu) then $S = \{\{a, b, c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a\}, \{b\}, \{c\}\}$. In this way, the algorithm is able to combine collective and distributive properties, as in ‘those football teams whose members are British’.

We will not explore collective versions of the Incremental Algorithm further here, focusing instead on the relatively simple case of $D\&R_{\text{Plural}}$, in which all properties are distributive. As in the case of $D\&R$, it is easy to separate Attributes and Values when referring to sets, allowing a closer approximation of Full Brevity: the resulting algorithm, $D\&R_{\text{Plural,Att}}$ is to $D\&R_{\text{Plural}}$ as $D\&R_{\text{Att}}$ is to $D\&R$; overlapping Values can be treated as described in section 3.1. In what follows we will, once again, take property-oriented versions of the Incremental Algorithm as our starting point, but implications for the separation between Attributes and Values will be mentioned where they are nontrivial.

4.2 Using Negations and Disjunctions

Now that we are able to generate references to sets, let us move away from purely intersective descriptions, on to full Boolean combinations of properties. Consider a KB whose domain is a set of animals (a, b, c, d, e) and whose only Attributes are TYPE and COLOUR:

TYPE: Dog ($\{a, b, c, d, e\}$), Poodle ($\{a, b\}$)

COLOUR: Black ($\{a, b, c\}$), White ($\{d, e\}$)

(All domain elements happen to be dogs.) In this situation, the Incremental Algorithm does not allow us to individuate any of the animals. Intuitively, however, the KB should enable one to refer to c , for example, since it is the only black dog that is *not* a poodle:

$$\{c\} = \text{Black} \cap \overline{\text{Poodle}}$$

A similar gap exists where *disjunctions* might be used. For example, the Incremental Algorithm does not make the set of dogs that are either white or poodles referable, whereas it *is* referable in English, e.g., ‘The white dogs and the poodles’.

In the next two sections, we will investigate how negation and disjunction can be taken into account in GRE. But first we introduce a trick for determining whether unique identification of an entity is possible, in a given situation.⁷ The idea is to calculate, for each element d in the domain, the *Satellite set* of d , that is, the intersection of the extensions of all the properties true of d . Taking all extensions from our dogs’ example, we have

$$\begin{aligned} \text{Satellites}(a) &= \text{Satellites}(b) = \text{Dog} \cap \text{Poodle} \cap \text{Black} = \{a, b\} \\ \text{Satellites}(c) &= \text{Dog} \cap \text{Black} = \{a, b, c\} \\ \text{Satellites}(d) &= \text{Satellites}(e) = \text{Dog} \cap \text{White} = \{d, e\} \end{aligned}$$

Satellite sets show which sets can be uniquely identified and which ones cannot. In the case of the dogs, for example, no intersective description of $\{c\}$ is possible because, in the Satellite sets, c is always accompanied by other objects (i.e., a and b); more generally, in this example, no object in the domain is uniquely identifiable, since no object occurs in a Satellite set that is a singleton.

Satellite sets can also be applied to the *construction* of descriptions. The entity $\{a, b\}$, for example, is uniquely described by the intersection $\text{Dog} \cap \text{Poodle} \cap \text{Black}$, and this can be read off the list of Satellite sets. Two of the three properties in $\text{Dog} \cap \text{Poodle} \cap \text{Black}$ are redundant, however. Using Satellite sets for the construction of descriptions can be particularly useful when properly generalized to Boolean descriptions, but shortening the resulting descriptions in a computationally efficient way is difficult (van Deemter and Halldórsson 2001). The present paper will focus on another approach to Boolean descriptions, which takes the Incremental Algorithm as its point of departure (van Deemter 2001).

4.3 Generating Boolean descriptions

In this section, we will show how full Boolean descriptions can be generated. This can be done in many different ways depending, among other things, on what form of descriptions are preferred, for example, disjunctions of conjunctions, or conjunctions of disjunctions. We will aim for the latter, while staying as close as possible to the Incremental Algorithm. The algorithm proceeds as follows. First we add negations to the list of atomic properties. Then $\text{D\&R}_{\text{Plural}}$ runs a number of times: first, in *phase 1*, the algorithm is performed using all positive and negative literals; if this algorithm ends before $C = S$, *phase 2* is entered in which further distractors are removed from C by making use of negations of intersections of two literals, and so on, until either $C = S$ (Success) or all combinations have been tried (Failure). Observe that the negation of an intersection comes down to set union, because of De Morgan’s Law: $\overline{P_1 \cap \dots \cap P_n} = \overline{P_1} \cup \dots \cup \overline{P_n}$. Thus, *phase 2* of the algorithm deals with disjunctions of length 2, *phase 3* deals with disjunctions of length 3, *etcetera*. Optimizations may be applied to shorten the resulting descriptions. For instance, a description of the form $(P \cup Q) \cap (P \cup R)$ can be simplified to $(P \cup (Q \cap R))$ using standard algorithms (e.g., McCluskey 1965). Such optimizations, however, are less urgent than in the case of the more verbose descriptions generated using Satellite sets (see above), and we will disregard optimizations here.

⁷ The idea of Satellite sets is due to Magnús Halldórsson, *p.c.*

A schematic presentation may be useful, in which $P_{+/-}$ stands for any literal, that is, any atomic property or its negation. (Different occurrences of $P_{+/-}$ denote potentially different literals.) The *length* of a property will equal the number of literals occurring in it. We will say that a $D\&R_{\text{Plural}}$ phase *uses* a set of properties X if it loops through the properties in X (i.e., X takes the place of \mathbb{P} in the original $D\&R_{\text{Plural}}$).

$D\&R_{\text{Boolean}}$:

Phase 1. Perform $D\&R_{\text{Plural}}$ using *all properties of the form* $P_{+/-}$;
if this is successful then stop, otherwise go to *phase* (2).

Phase 2. Based on the Values of L and C coming out of *phase* (1),
perform $D\&R_{\text{Plural}}$ using *all properties of the form* $P_{+/-} \cup P_{+/-}$;
if this is successful then stop, otherwise go to *phase* (3).

Phase 3. Based on the Values of L and C coming out of *phase* (2),
perform $D\&R_{\text{Plural}}$ using *all properties of the form* $P_{+/-} \cup P_{+/-} \cup P_{+/-}$;
if this is successful then stop, otherwise go to *phase* (4).

Etcetera.

One can require without loss of generality that no property, considered at any phase, may have different occurrences of the same atom. (For example, it is useless to consider the property $\overline{P_1} \cup \overline{P_2} \cup P_1$, which must be true of any element in the domain, or the property $\overline{P_1} \cup \overline{P_2} \cup \overline{P_1}$, which is equivalent to the earlier-considered property $\overline{P_1} \cup \overline{P_2}$.) Since, therefore, at *phase* n , there is room for properties of length n , the maximal number of phases equals the total number of atomic properties.

Consider our old example, where the preference order of atomic properties corresponds with the order in which they are listed, and where the same order extends to their negations, all of which are less preferred. Abbreviating $B = \text{Black}$, $D = \text{Dog}$, $P = \text{Poodle}$, and $W = \text{White}$, we have $\mathbb{P} = \langle B, D, P, W, \overline{B}, \overline{D}, \overline{P}, \overline{W} \rangle$. Now if $S = \{c, d, e\}$ or $S = \{c\}$ (as before) are to be characterized, nothing eventful happens; in both cases, a description is found during *phase* 1: \overline{P} in the first case, $B \cap \overline{P}$ in the second. The situation gets more interesting if $S = \{a, b, d, e\}$, which triggers *phase* 2. For instance, if positive literals precede negative literals, the properties relevant for *phase* 2 might be ordered as follows:

$$\{ B \cup D, B \cup P, B \cup W, D \cup P, D \cup W, P \cup W, B \cup \overline{D}, B \cup \overline{P}, B \cup \overline{W}, \\ D \cup \overline{B}, D \cup \overline{P}, D \cup \overline{W}, P \cup \overline{B}, P \cup \overline{D}, P \cup \overline{W}, W \cup \overline{B}, W \cup \overline{D}, W \cup \overline{P}, \\ \overline{B} \cup \overline{D}, \overline{B} \cup \overline{P}, \overline{B} \cup \overline{W}, \overline{D} \cup \overline{P}, \overline{D} \cup \overline{W}, \overline{P} \cup \overline{W} \}$$

During *phase* 1, no property is selected, since the only property true of all elements in $S = \{a, b, d, e\}$ is D , which fails to remove any distractors. During *phase* 2, one property after another is rejected. For example, the property $B \cup D$ is rejected because it does not remove any distractors. The first property that is true of all elements of S while also removing distractors is $P \cup W$. This property removes all distractors at once, causing the algorithm to end with $L = \{\text{Poodle} \cup \text{White}\}$ as the complete description. If we modify the example by letting $[[\text{Black}]] = \{a, c\}$ (rather than $\{a, b, c\}$) and $S = \{b, c, d, e\}$ (rather than $S = \{a, b, d, e\}$), then the description $L = \{\overline{\text{Black}} \cup \text{Poodle}\}$ is found.

$D\&R_{\text{Boolean}}$ is not only incremental *within* a phase, but also from one phase to the next, which causes shorter disjunctions to be favoured over longer ones. Once a property has been selected, it will not be abandoned even if properties selected during later phases make it logically superfluous. As a result, one may generate descriptions like $X \cap (Y \cup Z)$ (e.g., ‘white (cats and dogs)’) in a situation where $Y \cup Z$ (e.g., ‘cats and dogs’) would

have sufficed (because $(Y \cup Z) \subseteq X$). This is not unlike some of the redundancies generated by Dale and Reiter's algorithm and, as in their case, it is unclear whether this is descriptively adequate. Adaptations can be made if needed. For instance, phases might run separately before running in combination: first (as usual) phase 1, then 2, then (as usual) 1&2, then 3, then 1&3 then 2&3 then (as usual) 1&2&3, etc.⁸ As a result of this adaptation, the description $Y \cup Z$ would be generated on account of *phase 2* alone.

Double incrementality, however, does not save $D\&R_{\text{Boolean}}$ from intractability. To estimate running time as a function of the number of properties (n_a) in the KB and those in the description (n_l), we can mirror an argument in Dale and Reiter (1995, section 3.1.1) to show that the maximal number of properties to be considered equals

$$\sum_{i=1}^{n_l} 2 \binom{n_a}{i} = \sum_{i=1}^{n_l} 2 \frac{n_a!}{i!(n_a-i)!}$$

(The factor of 2 derives from inspecting both each atom and its negation.) If $n_l \ll n_a$ then this is in the order of $n_a^{n_l}$. To avoid intractability, the algorithm can be pruned. No matter where this is done, the result is a polynomial algorithm. By cutting off after *phase 1*, for example, only (negations of) atomic properties are combined, producing such descriptions as 'the black dog that is *not* a poodle', disregarding more complex descriptions; as a result, completeness is lost, but only for references to non-singleton sets, because set union does not add descriptive power where the description of singletons is concerned. The number of properties to be considered by this simpler algorithm equals $(n_a)^2 + 2n_a - 1$. To produce descriptions like $White \cap (Cat \cup Dog)$ (i.e., 'white (cats and dogs)') as well, the algorithm can be cut off one phase later, leading to a worst-case running time of $O(n_a^3)$, and so on for more and more complex descriptions.

$D\&R_{\text{Boolean}}$ can, of course, be modified to take advantage of the distinction between Attributes and Values. Suppose, for example, that $V_1 \cup \dots \cup V_n$ takes precedence over $W_1 \cup \dots \cup W_n$ whenever there are more *negative* Values among V_1, \dots, V_n than among W_1, \dots, W_n , then the preference ordering between Attributes may be taken into account if the number of negative Values is the same in both unions; in case of a tie, the number of distractors removed by each of the two unions may decide; if all this fails to tip the balance, the relative specificity of Attributes may be used. The situation resembles that of $D\&R_{\text{Att}}$ but, in the case of the new algorithm, $D\&R_{\text{Boolean, Att}}$, there is more scope for choice, because it compares *combinations* (i.e., unions) of properties: When the preference order of individual Attributes has been decided, it can happen that $[[V_i]]$ is more preferred than $[[W_j]]$, while $[[W_k]]$ is more preferred than $[[V_l]]$, in which case it is unclear whether $V_1 \cup \dots \cup V_n$ should be more preferred or $W_1 \cup \dots \cup W_n$. (Problems of this kind are not specific to *Boolean* combinations. For example, if an object x is identified through the *relation* $R(xy)$ and the *predicate* $P(y)$ then the degrees of preference of both R and P are relevant, and it is unclear which of the two is most important.)

Once $D\&R_{\text{Boolean, Att}}$ is constructed along these lines, the question of *overlapping* Values arises in exactly the same way as in the case of $D\&R_{\text{Att}}$ and $D\&R_{\text{Plur, Att}}$. The problem arises if components of different unions overlap, as when the algorithm compares $V_{i,j} \cup V_{k,l}$ and $V_{i,j} \cup V_{k,l'}$, where $V_{k,l}$ and $V_{k,l'}$ *overlap* in the sense of section 3.1: as in the case of $D\&R_{\text{Att}}$, simply choosing the option that removes most distractors may cause the algorithm to become incomplete. This problem can be overcome as before, using

⁸ This possibility was suggested to me by Richard Power, *p.c.*

either limited backtracking or inclusion of all relevant options (section 3.1). Instead of exploring $D\&R_{\text{Boolean,Att}}$ any further, we will return to its predecessor $D\&R_{\text{Boolean}}$ to prove that it is powerful enough to do its job.

4.4 Proving Boolean Completeness

In section 3.3, we proved Intersective Completeness for two versions of Dale and Reiter's Incremental Algorithm, $D\&R$ and $D\&R_{\text{Att}}$. We now prove Boolean Completeness for $D\&R_{\text{Boolean}}$, the Boolean extension of $D\&R_{\text{Plural}}$.

Theorem 3. Completeness of $D\&R_{\text{Boolean}}$. Assume there are at most denumerably many properties, and finitely many distractors (one or more). Then if a set can be individuated distributively by any Boolean combination of properties, $D\&R_{\text{Boolean}}$ will find such a combination.

Proof. Any Boolean expression can be written in Conjunctive Normal Form (CNF), that is, as an intersection of unions of literals (e.g., Fitting 1996). Theorem 3 follows from the following Lemma.

Lemma. Let φ be a CNF formula whose longest union has a length of n (i.e., it conjoins n literals). Then $D\&R_{\text{Boolean}}$ will find a description φ' that is coextensive with φ , in at most n phases. This is proven by induction on the size of n .

Basic case: If $n = 1$, the Lemma is equivalent to completeness of $D\&R_{\text{Plural}}$, the proof of which is analogous to that of the completeness of $D\&R$, replacing $\{r\}$ by S .

Induction step: Suppose the Lemma is true for all $n < i$. Now consider a CNF φ whose longest union has length i ; let φ contain m unions of length i , namely $\varphi_1 \cap \dots \cap \varphi_m$. Then φ can be written as the CNF $\chi \cap \varphi_1 \cap \dots \cap \varphi_m$, where all the unions in χ have length $< i$. The Lemma is true for all $n < i$, so if χ is sent to $D\&R_{\text{Boolean}}$ then the output is some χ' such that $[[\chi']] = [[\chi]]$, in fewer than i phases; so if, instead, φ is sent to $D\&R_{\text{Boolean}}$ then, after $i - 1$ phases, some possibly incomplete description η has been found, such that $[[\eta]] \subseteq [[\chi]]$. Also, $[[\varphi]] \subseteq [[\eta]]$. Phase i inspects all unions of length i , including each of $\varphi_1, \dots, \varphi_m$. Therefore, unless a description coextensive with φ is found before phase i , one will be found during phase i . To see this, suppose the algorithm finds ψ such that $[[\psi]] = [[\varphi_1]] \cap \dots \cap [[\varphi_m]]$, then $[[\chi]] \cap [[\psi]] = [[\varphi]]$; but $[[\varphi]] \subseteq [[\eta]] \subseteq [[\chi]]$, therefore also $[[\eta]] \cap [[\psi]] = [[\varphi]]$. \square

5. Conclusion

The GRE algorithms discussed in this paper are fairly limited in their aspirations. For example, they do not involve relational descriptions (Dale and Haddock 1991, Horacek 1997, Krahmer et al. 2001) or properties that are vague or context-dependent (van Deemter 2000). Moreover, they disregard shades of salience (unlike Krahmer and Theune 1999, Theune 2000), relying instead on a simple dichotomy between those objects that are salient enough (which end up in the domain \mathcal{D}) and those that are not (Reiter and Dale 2000, section 5.4). Finally, like all other GRE algorithms that we are aware of, they disregard the generation of descriptions in intensional contexts (e.g., 'John knows that x is the murderer of Jones', Dowty et al. 1981.)

But even within this limited brief, existing algorithms are incomplete. In particular, we

have shown Dale and Reiter (1995)'s Incremental Algorithm to be intersectively incomplete with respect to Attributes that have overlapping Values and (less surprisingly) in some situations where the class of properties is infinitely large. Furthermore, the Incremental Algorithm excludes reference to *sets* and limits itself to purely intersective combinations of atomic properties, causing the algorithm to be incomplete with respect to the set of all Boolean combinations. Having noted these shortcomings, we have modified the Incremental Algorithm in such a way that these limitations are removed. The result is a set of generalizations of the Incremental Algorithm, for which we have proven completeness under appropriate assumptions.

Integration of these different algorithms into one unified algorithm would be a non-trivial enterprise, as we have seen in section 4.3. Integration with previously-proposed extensions of the Incremental Algorithm would raise further questions, stemming from the fact that our descriptions are structurally complex. For example, consider the treatment of *relational properties*. What is better: adding a relational property to a given incomplete description ('... in the wooden shed'), or adding a negated property ('... which is not a poodle')? Making informed decisions about such questions, with proper attention for their combined effects, is a difficult task which is perhaps best tackled using the graph-theoretical approach outlined in Kraemer et al. (2001). Their approach is specifically suitable for accommodating different GRE algorithms and treats relations in the same way as properties.

Brevity. We have assumed that, on the whole, descriptions ought to be as brief as they can, as long as they are uniquely identifying. But in fact, a description can contain much *more* than is logically necessary for identification, even beyond the redundancies allowed by the Incremental Algorithm. Logically superfluous properties can, for example, be motivated by 'overloading' if they serve communicative purposes other than identification (Pollack 1991, Dale and Reiter 1995, section 2.4, Stone and Webber 1998, Jordan 1999). A description may also contain *fewer* properties than would be necessary for identification, for example when no distinguishing description exists. A non-distinguishing description may either take the form of a definite description (as in 'John's son', when John has several sons), or of an *indefinite* description (as in 'one of John's sons'; Horacek 1997, Stone and Webber 1998, Kraemer and Theune 1999). In both cases, the description may be useful even though it fails to be distinguishing.

Tractability. Computational tractability has also been paramount in our explorations. There is no agreement on the extent to which computational linguists should worry about the computational complexity of algorithms, or about the precise way in which complexity is most relevant (e.g., 'typical' or worst-case complexity, cf., note 3). Far from aiming to speak the last word on these issues, the material discussed here does shed some light on them. For example, even a fast algorithm can require a large number of calculations, in which case a solution may never be found; in the case of GRE, this happens when the set of distractors or the set of properties becomes extremely large (section 3.2). Conversely, a complex algorithm can be safe to use if the domain is small (or if key calculations can be performed off-line e.g., Bateman 1999). This may be achieved by putting a bound on the size of the search space, and this may be justifiable on empirical grounds (see the discussion of $D \& R_{\text{Boolean}}$ in section 4.3). One might, on the other hand, argue that bounding does not eliminate the disadvantages of an otherwise intractable algorithm, because the true nature of an algorithm is best revealed 'by considering how it operates on unlimited cases' (Barton et al. 1987, section 1.4.1). Be this as it may, we believe that complexity theory can offer valuable insights into the structure of GRE al-

gorithms, and that the growing attention for complexity in this area is a healthy development even if the practical implications are not always straightforward.

Recent work also highlights an interesting mirror image of GRE complexity: a logically superfluous property may make it easier for the reader to *find* the referent. An interesting class of cases is explored in Paraboni (2000), which focuses on descriptions of document parts. Consider the description ‘the medicine depicted in section 2.3’. If section 2 happens to contain only one figure then the description ‘the medicine depicted in section 2’ would have been logically sufficient, but this description would have made it necessary for the reader, in the worst case, to search through all of section 2, making it less useful. Examples of this kind suggest that GRE should also take the computational complexity of *interpretation* into account. Experimental research on ‘minimal cooperative effort’ (Clark 1992, Cremers 1996) points in the same direction; we are, however, not aware of computationally applicable formalizations of these ideas so far.

Acknowledgments

Thanks are due to Robert Dale, Magnús Halldórsson, Emiel Krahmer, Ivandré Paraboni, Paul Piwek, Richard Power, Ehud Reiter, and Matthew Stone for useful discussions. Helpful comments from the reviewers of *Computational Linguistics* are also gratefully acknowledged.

References

- Barton, G. Edward, Robert C. Berwick, and Eric Sven Ristad. 1987. *Computational Complexity and Natural Language*. MIT Press, Cambridge, Mass.
- Bateman, John A. 1999. Using Aggregation for Selecting Content when Generating Referring Expressions. In Procs. of ACL 1999, University of Maryland.
- Herbert H. Clark. 1992. *Arenas of Language Use*. CSLI Publications, Stanford.
- Cremers, Anita. 1996. *Reference to Objects; an empirically based study of task-oriented dialogues*. Ph.D. thesis, University of Eindhoven.
- Dale, Robert. 1989. Cooking up referring expressions. In Procs. of 27th annual meeting of Assoc. for Comp. Ling. (ACL-89), pages 68-75.
- Dale, Robert. 1992. *Generating Referring Expressions: Constructing Descriptions in a Domain of Objects and Processes*. MIT Press, Cambridge, Mass.
- Dale, Robert and N. Haddock. 1991. Generating Referring Expressions involving Relations. Procs. of EACL, Berlin, pages 161-166.
- Dale, Robert and Ehud Reiter. 1995. Computational Interpretations of the Gricean Maxims in the Generation of Referring Expressions. *Cognitive Science* 18, pages 233-263.
- Dalianis, Hercules and Eduard Hovy. 1996. Aggregation in natural language generation. In G. Adorni and M. Zock (Eds.) “Trends in natural language generation: an artificial intelligence perspective”, pages 88-105. Springer-Verlag.
- Dowty, David, Robert Wall, and Stanley Peters. 1981. *Introduction to Montague Semantics*. Kluwer, Dordrecht.
- Fitting, Melvin. 1996. *First-order Logic and Automated Theorem Proving*. Springer, New York.
- Grice, Paul. 1975. Logic and Conversation. In P. Cole and J. Morgan (Eds.), *Syntax and Semantics: Vol 3, Speech Acts*, pages 43-58. New York, Academic Press.
- Horacek, Helmut. 1997. An Algorithm for Generating Referential Descriptions with Flexible Inter-

faces. In Procs. of 35th. ACL conference, 1997, Madrid.

Jordan, Pamela. 1999. Jordan. Contextual Influences on the Attributes included in Repeated Descriptions. In Proc. of workshop Generation of Nominal Expressions, 11th European Summer School in Logic, Language, and Information (ESSLI'99), Utrecht. Augmented version to appear in K. van Deemter and R. Kibble (Eds.), "Information Sharing". CSLI Publications, Stanford.

Kamp, Hans and Uwe Reyle. 1993. From Discourse to Logic. Kluwer Academic Publishers.

Kibble, Rodger. 1999. Cb or not Cb? Centering Theory applied to NLG. In procs. of ACL workshop on Discourse/Dialogue Structure and Reference, Maryland.

Kleene, Stephen C. 1971. *Introduction to Metamathematics*. Wolters-Noordhoff. Groningen.

Krahmer, Emiel and Mariët Theune. 1999. Generating Descriptions in Context. In Procs. of workshop Generation of Nominal Expressions, ESSLI'99. Augmented version to appear in K. van Deemter and R. Kibble (Eds.), "Information Sharing". CSLI Publications, Stanford.

Krahmer, Emiel, Sebastiaan van Erk, and André Verleg. 2001. A Meta-Algorithm for the Generation of Referring Expressions. In Procs. of 8th European Workshop on Natural Language Generation (EWNLG-2001), Toulouse.

McCluskey Edward J. 1965. *Introduction to the Theory of Switching*. New York. McGraw-Hill.

Mittal, V., J. Moore, G. Carenini, and S. Roth. 1998. Describing Complex Charts in Natural Language: a Caption Generation System. *Computational Linguistics* 24/3, pages 431-468.

Paraboni, Ivandr . 2000. An algorithm for generating document-deictic references. Procs. of workshop Coherence in Generated Multimedia, associated with First Int. Conf. on Natural Language Generation (INLG-2000), Mitzpe Ramon.

Pechman, Thomas. 1989. Incremental speech production and referential overspecification. *Linguistics*, 27, pages 89-110.

Pollack, Martha E. 1991. Overloading intentions for efficient practical reasoning. *No s* 25, pages 513-536.

Reiter, Ehud. 1990. The Computational Complexity of Avoiding Conversational Implicatures. In Proc. ACL-1990, Pittsburgh.

Reiter, Ehud and Robert Dale. 2000. *Building Natural language Generation Systems*. Cambridge University Press, Cambridge, UK.

Stone, Matthew and Bonnie Webber. 1998. Textual Economy through Close Coupling of Syntax and Semantics. In Procs. of INLG 1998, pages 178-187.

Stone, Matthew. 2000. On Identifying Sets. In Procs. of INLG-2000, Mitzpe Ramon.

Theune, Mari t. 2000. *From Data to Speech; Language Generation in Context*. Ph.D. thesis, University of Eindhoven.

van Deemter, Kees. 2000. Generating Vague Descriptions. In Procs. of First Int. Conf. on Natural Language Generation (INLG-2000), Mitzpe Ramon.

van Deemter, Kees. 2001. Generating Referring Expressions: Beyond the Incremental Algorithm. In Procs. of Fourth Int. Ws. on Computational Semantics (IWCS-4), Tilburg, Jan. 2001.

van Deemter, Kees and Magn s Halld rsson. 2001. Logical Form Equivalence: the Case of Referring Expressions Generation. In Procs. of 8th European Ws. on Natural Language Generation (EWNLG-2001), Toulouse.