KEES VAN DEEMTER AND EMIEL KRAHMER

GRAPHS AND BOOLEANS: ON THE GENERATION OF REFERRING EXPRESSIONS

1. Introduction

Generation of Referring Expressions (GRE) is a key task of Natural Language Generation NLG systems (e.g., Reiter and Dale, 2000, section 5.4). The task of a GRE algorithm is to find combinations of properties that allow the generator to refer uniquely to an object or set of objects, called the *target* of the algorithm. Older GRE algorithms tend to be based on a number of strongly simplifying assumptions. For example, they assume that the target is always one object (rather than a set), and they assume that properties can always only be conjoined, never negated or disjoined. Thus, for example, they could refer to a target object as "the small violinist", but not as "the musicians *not* holding an instrument". As a result of such simplifications, many current GRE algorithms are *logically incomplete*. That is, they sometimes fail to find an appropriate description where one does exists.[1] To remedy such limitations, various new algorithms have been proposed in recent years, each of which removes one or more simplifying assumptions. They extend existing GRE algorithms by allowing targets that are sets (Stone, 2000; van Deemter, 2000), gradable properties (van Deemter, 2000, 2006), salience (Krahmer and Theune, 2002), relations between objects (Dale & Haddock, 1991; Horacek, 1997), and Boolean properties (van Deemter, 2001, 2002).

Recently a new formalism, based on labelled directed graphs, was proposed as a vehicle for expressing and implementing different GRE algorithms (Krahmer et al., 2001, 2003). Although the formalism was primarily argued to support relatively simple descriptions (not involving negations or disjunctions, for example), we will show that it can be used beyond these confines. Far from claiming that this will solve all the problems in this area, we do believe that a common formalism

[1] We use the term 'description' to denote either a combination of properties or its linguistic realization.

H. Bunt and R. Muskens, (eds.), Computing Meaning, Volume 3, 397–422.

would be extremely useful, as a basis for comparing and combining existing algorithms. An additional advantage is that the computational properties of graphs are well understood and efficient algorithms for manipulating graphs are available 'off the shelf' (e.g., Mehlhorn, 1984).

In this paper, we will explore to what extent the graph-based approach to GRE can be extended to express a variety of algorithms in this area. Our discussion will be limited to semantic aspects of GRE and, more specifically, to the problem of constructing combinations of properties that identify a referent uniquely (i.e., constructing a *distinguishing description*). Our main finding will be that most existing GRE algorithms carry over without difficulty, but one algorithm, which focusses on the generation of Boolean descriptions that also contain relational properties, does not. For this reason, we propose an alternative algorithm that produces different types of Boolean descriptions from the original algorithm, using graphs in a very natural way.

The paper is structured as follows. In section 2 we briefly describe the basic graph-based GRE approach. Then, in section 3, we describe how various earlier GRE algorithms aimed at the generation of sets, gradable properties, salience and negated properties can be reformulated in terms of the graph approach. In section 4 we describe two graph-based algorithms for the generation of full Boolean expressions, one based directly on van Deemter (2001, 2002) and one new alternative. In the concluding section, we list some of the new questions that come up when the different types of algorithms discussed in this paper are combined.

2. Graph-based GRE

A number of GRE algorithms were proposed in the 1990s, of which the Incremental Algorithm from Dale and Reiter (1995) is probably the best known. These 'basic' GRE algorithms generate distinguishing descriptions of individual objects. The descriptions generated consist of conjunctions of atomic properties that are represented in a shared Knowledge Base (KB) that is formalized as an attribute/alue structure. Using the attributes TYPE, SIZE, and HOLDS, for example, a very simple KB may look as follows:

DOMAIN: $\{s_1, s_2, s_3, s_4\}$

TYPE: Musician $= \{s_1, s_2\}$, Technician $= \{s_3\}$, Trumpet $= \{s_4\}$
SIZE: Big $= \{s_1, s_3\}$, Small $= \{s_2, s_4\}$
HOLDS: $s_4 = \{s_2\}$

Note that the first argument of a relation like HOLDS is, for now, treated as another attribute, expressing that the object that holds s_4 (a trumpet) is s_2 (a musician). In the abbreviated notation used here, only those attributes are listed that have a nonempty set of values. Thus, for example, it follows that HOLDS: $s_3 = \{\}$ (i.e., nobody holds the technician).

Given this KB, the Incremental Algorithm can describe s_1 by conjoining the properties ⟨ SIZE, Big ⟩ and ⟨ TYPE, Musician ⟩, for example, because the intersection of their extensions equals $\{s_1, s_3\} \cap \{s_1, s_2\} = \{s_1\}$. Simplifying considerably, the algorithm proceeds by incrementally conjoining more and more properties, removing more and more '*confusables*' (i.e., objects with which the target object may be confused). This process continues until only the target itself is left. The Incremental Algorithm does not allow backtracking, which is what makes it fast (Dale and Reiter, 1995).

In Krahmer et al. (2001), it was shown that algorithms such as the Incremental Algorithm can be mirrored in a graph-based formalism, by expressing the description as well as the KB as a labelled directed graph. Let $\mathcal{D}$ be the domain of discourse, P a set of names for properties, and R a set of names for relations, then $L = P \cup R$ is the set of *labels*. Formally, $G = \langle V_G, E_G \rangle$ is a labelled directed graph, where $V_G \subseteq \mathcal{D}$ is the set of *nodes* (the potential referents) and $E_G \subseteq V_G \times L \times V_G$ is the set of labelled directed *edges*. The KB above can now be reformulated as the graph in Figure 56. Properties are modelled as loops, i.e., edges which start and end in the same node, whereas relations are modelled as edges between nodes.[2] Note that the object of the relation is no longer hidden within the attribute HOLDS, allowing, for example, relations to be used iteratively, as in 'The man who *holds* a trumpet *owned* by a woman'.

We call the graph S that represents the KB the *scene graph*; if $s \in V_S$ is the target object then s can be singled out as the designated element of S and we call $\Sigma = \langle s, S \rangle$ the *scene pair*. Crucially, a description is represented using a similar pair, consisting of a connected *description graph* D and a designated element $d \in V_D$; the pair $\Delta = \langle d, D \rangle$ is called a *description pair*. Representing both the description and the scene using graphs allows one to view GRE as a *graph construction*

[2] In fact, nothing forbids relations which start and end in the same node (which is correct, in view of potentially reflexive relations such as 'shaving' and 'washing'). However, for simplicity, we shall assume throughout this paper that all relations are non-reflexive.

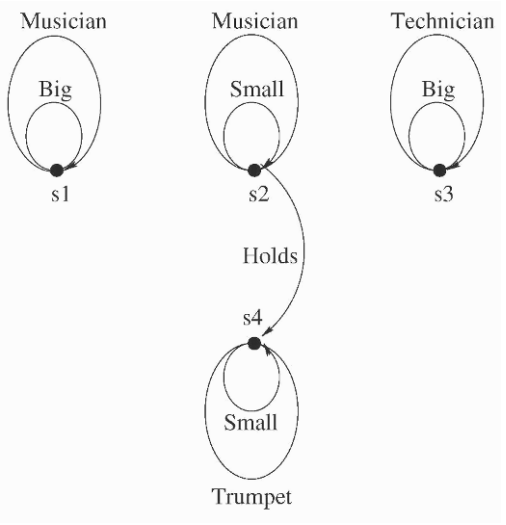

Figure 56. A scene graph involving four objects

problem. More particularly, the task is to construct a description pair that 'refers uniquely' to a given scene pair.[3] The notion 'refers uniquely' is defined via the notion of a *subgraph isomorphism*. A graph S' is a *subgraph* of S if $V_{S'} \subseteq V_S$ and $E_{S'} \subseteq E_S$.

π is a *subgraph isomorphism* between D and S (Notation: $D \sqsubseteq_\pi S$) iff there exists a subgraph S' of S such that π is a bijection $\pi : V_D \rightarrow V_{S'}$ such that for all nodes $v, w \in V_D$ and all labels $l \in L$, $(v, l, w) \in E_D \Leftrightarrow (\pi(v), l, \pi(w)) \in E_{S'}$.

A description pair $\Delta = \langle d, D \rangle$ *refers* to a scene pair $\Sigma = \langle s, S \rangle$ iff D is connected and $\exists \pi : (D \sqsubseteq_\pi S$ and $\pi(d) = s)$.

Thus, a description pair $\Delta = \langle d, D \rangle$ refers to a scene pair $\Sigma = \langle s, S \rangle$ iff there exists a subgraph isomorphism between D and S that maps d to s. Note that, using this terminology, a description pair can 'refer' to more than one scene pair. Consider the description graphs depicted in Figure 57, each of which has s as its designated element. Let $\Sigma = \langle s_2, S \rangle$, that is, we want to generate an expression referring to s_2 in S, where S is the scene graph depicted in Figure 1. Then the first of the three corresponding description pairs refers to Σ but not uniquely (it may

[3] Equivalently, one could say that the description pair refers to the designated element *given* the scene graph.

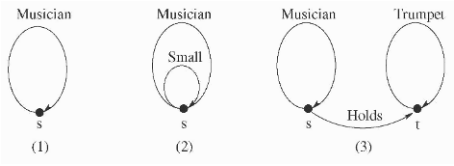

Figure 57. Three possible description graphs

also refer to the 'confusable' $\langle s_1, S\rangle$), while both of the other pairs refer to Σ uniquely. We define:

Given a graph S and a description pair Δ, the set of *confusables*, *Conf*(Δ, S), is the set of those nodes $s' \in V_S$ such that Δ refers to $\langle s', S\rangle$.

A description pair $\Delta = \langle d, D\rangle$ *refers uniquely* to a scene pair $\Sigma = \langle s, S\rangle$ iff $\langle d, D\rangle$ *refers* to $\langle s, S\rangle$ and $\forall\pi : (D \sqsubseteq_\pi S \Rightarrow \pi(d) = s)$.

Note that if a description pair $\Delta = \langle d, D\rangle$ refers uniquely to a scene pair $\Sigma = \langle s, S\rangle$, then *Conf*$(\Delta, S) = \{s\}$, i.e., the set of confusables is a singleton.

We have seen that there are multiple unique (distinguishing) descriptions for our target s_2 in S. As usual in GRE, certain solutions may be given preference over others. There are various ways to do this. One way is by considering properties in some fixed order and to let the algorithm proceed incrementally by adding suitable properties one by one, stopping once a uniquely referring description is found (Dale and Reiter, 1995). A more general way would be to use *cost functions* (Krahmer et al. 2001, 2003). Costs are associated with subgraphs D of the scene graph S (notated *cost*(D)). We require the cost function to be *monotonic*. This implies that extending a graph D with an edge e can never result in a graph which is cheaper than D. Formally,

$$\forall D \subseteq S\ \forall e \in E_S : \mathit{cost}(D) \leq \mathit{cost}(D + e)$$

Here we assume that if $D = \langle V_D, E_D\rangle$ is a subgraph of S, the costs of D can be determined by summing over the costs associated with the edges of D. For the time being we assume that each edge costs 1 point. Naturally, this is a simplification, which does not do justice to the potential benefits of cost functions (but see Krahmer et al. (2003) for discussion). Thus, the first distinguishing graph in Figure 2 costs 2 points and is cheaper than the other one (which costs 3 points).

Figure 2 contains the sketch of a basic graph-based GRE algorithm, called **makeReferringExpression**. It takes as input a scene pair Σ

```
makeReferringExpression(s, S) {
    bestGraph := ⊥;
    D := ⟨{s}, ∅⟩;
    return findGraph(s, bestGraph, D, S);
}

findGraph(s, bestGraph, D, S) {
    if [bestGraph ≠ ⊥ and cost(bestGraph) ≤ cost(D)]
    then return bestGraph;
    Conf := {n : n ∈ V_S ∧ ⟨s, D⟩ refers to ⟨n, S⟩};
    if Conf = {s} then return D;
    for each adjacent edge e do
       I := findGraph(s, bestGraph, D + e, S );
       if [bestGraph = ⊥ or cost(I) ≤ cost(bestGraph)]
       then bestGraph := I;
    rof;
    return bestGraph;
}
```

Figure 58. Sketch of the main function (**makeReferringExpression**) and the subgraph construction function (**findGraph**)

consisting of the target s in a scene graph S. The description pair Δ is initialized with the target s and the initial description graph D whose only node is s. In addition, a variable *bestGraph* is introduced, for the best solution found so far. Since no solutions have been found at this stage, *bestGraph* is initialized as the empty graph $\perp$. In the **findGraph** function the algorithm systematically tries expanding D by adding adjacent edges (i.e, edges from s, or possibly from any of the other vertices added to the D under construction). For each D it is checked what the set of confusables is. A successful description is found iff $Conf = \{s\}$. The first distinguishing description that is found is stored in *bestGraph*. At that point the algorithm only looks for description graphs that are cheaper then the best (i.e., cheapest) solution found so far, performing a complete, depth-first search. (Naturally, graph-based generation is compatible with different search strategies as well.) It follows from the above-mentioned monotonicity requirement that the algorithm outputs the cheapest distinguishing description graph, if one exists. Otherwise it returns the empty graph.[4]

[4] This basic algorithm has been implemented in JAVA 2 (J2SE, version 1.4). For implementation and performance details we refer to Krahmer et al., 2003.

Discussion The graph-based GRE approach has a number of attractive properties. First, there are many efficient algorithms for dealing with graph structures (see for instance Mehlhorn, 1984, Gibbons 1985, and Chartrand and Oellermann, 1993). Second, the treatment of relations between objects is not plagued by some of the problems facing earlier approaches; there is, for instance, no need for making any *ad hoc* stipulations (e.g., that a property can only be attributed to a given object once per description, Dale and Haddock, 1991). This is because relational properties are handled in the same way as other properties, namely as edges in a graph. Relational properties cause testing for a subgraph isomorphism to have exponential complexity (see Garey & Johnson, 1979, Appendix A 1.4, GT48, on subgraph isomorphisms), but special cases are known in which the problem has lower complexity (e.g., when both graphs are *planar*, that is, drawable without crossing edges). The availability of results of this kind is an important advantage of using graphs in GRE. Many existing GRE algorithms can be reformulated using graphs. In the following section, we will show how some of these, each of which extends 'basic' GRE, can be recast in the graph-based approach. Our exposition of the original algorithms is necessarily sketchy; for details we refer to the original papers. In the section thereafter we show in more detail how the graph-based approach enables two different algorithms for the generation of boolean expressions.

3. Some simple extensions of graph-based GRE

3.1. *Referring to sets*

Firstly, we consider extensions of GRE algorithms that generate references to sets. Suppose, for example, we want to refer to the set $\{s_1, s_2\}$ in Figure 56 (to say that its elements are famous, for instance). This type of reference can be achieved by a simple extension of existing GRE algorithms: properties are conjoined as normal, removing from $Conf(\Delta, S)$ any objects that lie outside the target set, and the algorithm stops if and when the remainder equals the target set (i.e., all other confusables are removed). The target $\{s_1, s_2\}$, for example, may be described by the single property Musician (and realized as 'the musicians'). We will show that a similar procedure can be followed using a graph-based approach.

In fact, the algorithm described in the previous section is almost ready to generate simple descriptions referring to non-singular objects. The input of the algorithm is then no longer a single node $s \in V_S$, but a set of nodes $W \subseteq V_S$ (in this case $W = \{s_1, s_2\}$). The algorithm now tries to generate a description pair Δ uniquely referring to the scene pair $\Sigma = \langle W, S\rangle$. This requires a slight update of the definition of what it means to refer (uniquely): the constructed subgraph should refer (in the sense defined in section 2) to *each* of the nodes in the set W, but *not* to any of the nodes in the scene graph outside this set. Formally,

$\Delta = \langle d, D\rangle$ *refers* to $\Sigma = \langle W, S\rangle$ iff D is connected and *for each* $w \in W$ $\exists\pi(D \sqsubseteq_\pi S$ and $\pi(d) = w)$.

$\Delta = \langle d, D\rangle$ *uniquely refers* to $\Sigma = \langle W, S\rangle$ iff $\langle d, D\rangle$ *refers* to $\langle W, S\rangle$ and there exists no $w' \in V_S - W$ such that $\langle d, D\rangle$ refers to $\langle\{w'\}, S\rangle$.

Note that this redefines reference as always involving a set as its target; the singular case is obtained by restricting W to singleton sets. This does not affect the theoretical complexity of the algorithm, which depends on calculating sets of confusables (i.e., calculating $Conf(\Delta, S)$, for different Δ).

As observed in Stone (2000), GRE should also allow reference to sets based on properties that they have as collectives (such as 'being parallel to each other'). Solutions to this problem are proposed in Stone (1999) and van Deemter (2002), the latter of which can be mirrored directly in terms of graphs if nodes are allowed to represent *sets* of objects and edges are allowed to represent properties of collectives.

3.2. *Gradable properties*

The analysis of vague or gradable properties such as Small that we have used so far (consistent with Dale and Reiter, 1995) is not really satisfactory, for example because being small means something else for a person than for a musical instrument (Figure 56). The analysis is even more clearly inapplicable to superlatives. 'The smallest musician', for example, does not necessarily denote the object that is both the smallest object *in the domain* and a musician. It is better to let the KB list absolute values, such as a person's size in centimeters, and let GRE decide what counts as small in a given context (van Deemter, 2006). This approach allows the generator to describe someone as the *small(est) musician*, for example, even if the KB contains smaller objects, as long as these others are not musicians.

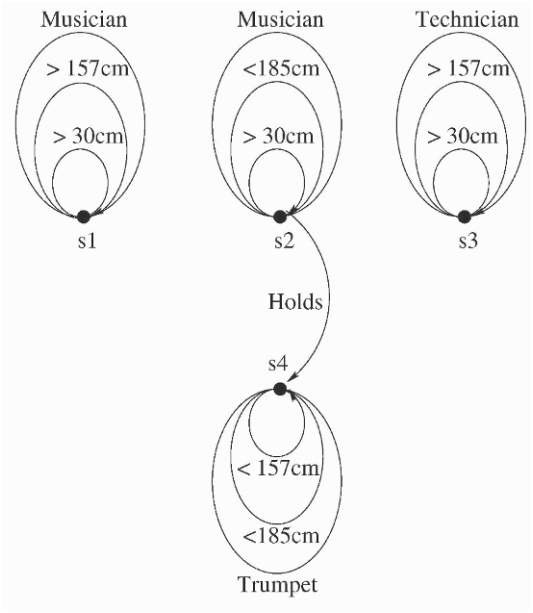

Figure 59. Graph with gradable properties made explicit

Looking at this proposal in a bit more detail, it is worth noting that it works by adding 'derived' properties to the database: properties of the form Size(x) > *value*, Size(x) < *value*, which can be inferred from absolute ones (Size(x) = *value*) listed in the database. Note that only a limited number of inequalities needs to be added, since absolute values not occurring in the database will not be relevant for derived properties. These derived properties are then used for removing confusables in the usual way.

Luckily, this procedure can be mirrored using graphs. The way to do this is by *extending* the scene graph by adding the derived inequalities to it as additional edges. Suppose that Size (s_1) = Size (s_3) = 185cm, whereas Size (s_2) = 157cm, and Size (s_4) = 30cm. Then, after transformation, the graph looks as in Figure 3.2.

Once this extended graph has been constructed, GRE proceeds in the usual way. We now find that the target object s_2 can be referred to uniquely by the description graph containing only the edges Musician and Size < 185cm. This graph can then be realized as 'the small musician' or 'the smallest musician' (for the details of this realization procedure we refer to van Deemter, 2006).

The theoretical complexity of the construction algorithm, as a function of the numbers of nodes and edges in the *revised* scene graph, does not change. In the worst case, however, the number of edges in the scene graph grows quadratically: if the original graph contains c nodes, each with a different absolute value, the operation illustrated in Figure 3.2 produces a graph with $c(c-1)$ *additional* edges. (Each of the c nodes has an absolute value and now acquires $c-1$ comparative values.)

3.3. *Salience*

Another recent innovation in GRE concerns the treatment of *salience*. Earlier algorithms simplified by assuming that all objects in the KB are equally salient (Reiter and Dale, 2000, section 5.4). Krahmer and Theune (2002) refined this account by one which allows degrees of salience in the style of Praguean topic/focus theory or centering. Formally, this is done using a salience weight function, which assigns a number between 0 (non salient) and 10 (maximally salient) to each object in the domain. All objects in the domain can be referred to, but the more salient an object is, the further its description can be reduced. The original algorithm works by adding properties until the set of confusables contains no object that is at least as salient as the target object. A faster, and probably slightly more natural version of the algorithm is obtained if the algorithm starts out by restricting the domain to the set of those objects that are at least as salient as the target set, causing only those properties to be added that remove salient distractors. This idea is easily implemented in the graph-based approach if we redefine the set of confusables as follows:

Given a scene graph S containing a target object s, and a description pair Δ, $\mathit{Conf}(\Delta, S)$ is the set of those nodes in $s' \in V_S$ that are at least as salient as s such that Δ refers to $\langle s', S\rangle$.

But, in fact, this amounts to treating salience as an in-built gradable property, which allows us to describe the effect of salience on GRE by a variant of the algorithm for gradable properties. Limiting ourselves to singular references and assuming that salience is the only gradable property in the KB, this can be done as follows. First, relevant comparative values of Salience are added to the scene graph; this involves properties of the form Salience$(x) >$ *value* only, since values of the form Salience$(x) <$ *value* are irrelevant in this connection. These salience-related properties are given preference over all others (in terms of costs this can be achieved by offering salience properties for free), so that

the GRE algorithm will cause every description to take salience into account. If the algorithm terminates successfully, the result is the unique description of an object s by means of a graph that attributes a number of properties to s, for example Musician and British and Salience > 8. This is a unique description, therefore it follows that s is *the most salient British musician* in the domain. The graph may subsequently be realized by the expression 'the British musician', leaving the property of maximal salience implicit. In this way, salience is treated as just another gradable attribute, with the only difference that it is always selected by the GRE algorithm and never linguistically realized.

3.4. *Negations*

Suppose we wanted to refer to the set $T = \{s_3, s_4\}$ in our basic scene graph. A simple trick (based on the notion of *satellite sets*) suffices to see that this is not possible if only atomic properties are taken into account. For any set $X \subseteq \mathcal{D}$, let *Satellites*(X), the 'satellite set' of X, be defined as the intersection of all the extensions of properties of which X is a subset (cf. van Deemter and Halldorsson, 2001):[5]

$S_X = \{A : A \in I\!P \wedge X \subseteq [\![A]\!]\}$

Satellites$(X) = \bigcap_{A \in S_X} ([\![A]\!])$

Clearly, if *Satellites*$(X) \neq X$, then the properties in the KB do not allow X to be characterized uniquely; even if all the properties of X are intersected, some confusables are not ruled out. Applying this to our target $T = \{s_3, s_4\}$, we observe that $S_T = \emptyset$ (s_3 and s_4 share no properties in the KB). This implies that *Satellites*$(T) = \bigcap \emptyset = \mathcal{D}$, (and hence not T).[6] What this shows is that our target T cannot be characterized by algorithms like the ones discussed in Dale and Reiter (1995), which rely on using intersections alone. As was argued in Van Deemter (2001, 2002), this is a serious limitation because a simple characterization *is* possible if negations of atomic properties are allowed. For example, the set of elements in the domain that are *not* Musicians will do the trick.

It is worth mentioning that negations are not only useful from a purely logical point of view. Even where they do not add to the expressive power of the generator (i.e., where they do not make more targets uniquely distinguishable), describing an object in terms of properties it

[5] $I\!P$ is the set of properties; $[\![A]\!]$ is the extension of A.

[6] We assume that every object in the domain of discourse is in $\bigcap \emptyset$, as is usual when the domain $\mathcal{D}$ is given (but compare e.g., Suppes, 1972).

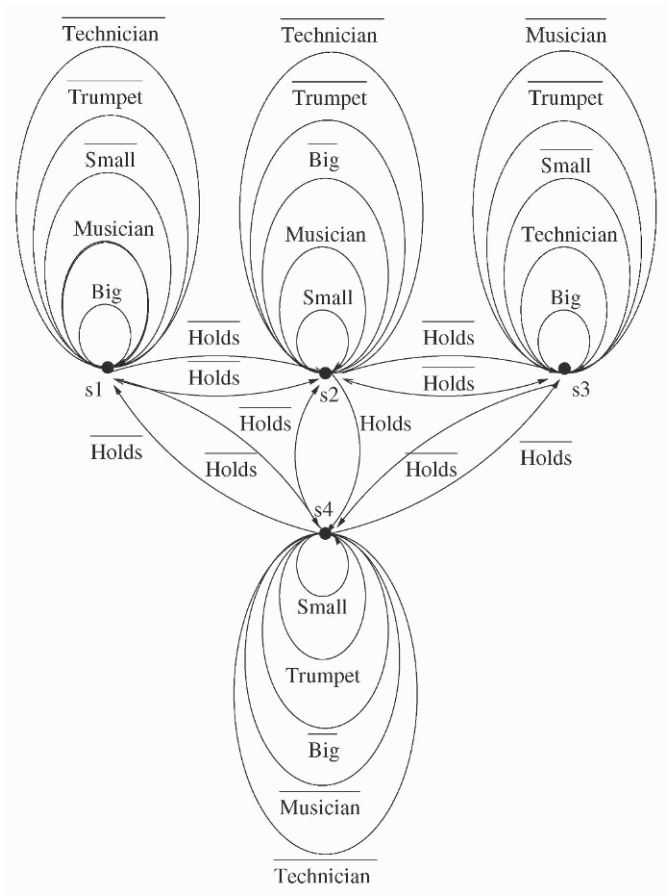

Figure 60. The scene graph enriched with negated edges

lacks can be the most efficient way to single it out from the rest: if all persons except one in a scene hold an instrument, then the odd one out may be best described as 'the person *not* holding an instrument', even if she could also have been described in purely positive (but possibly more complex) ways, for example 'the tall woman in the front row, with the pearl necklace'.

Adding negations to graph-based GRE is easy and follows a familiar pattern: we can extend the scene graph with additional edges, making explicit what was implicit in the original scene graph. For this we use the standard Closed World Assumption (according to which all atoms

not listed as true are false). For instance, according to our original scene in Figure 1, s_1 is not a technician and does not hold anything. This can be made explicit by adding negated edges. Let $\overline{P} = \{\overline{\text{Technician}}, \ldots\}$ be the set of names of negated properties and $\overline{R} = \{\overline{\text{Holds}}, \ldots\}$ the set of names of negated relations, then $L_{neg} = \overline{P} \cup \overline{R}$ is the set of negated labels. If $S = \langle V_S, E_S \rangle$ is the original scene graph, then the new scene graph with negated edges is $S' = \langle V_S, E_S{}' \rangle$, where

$$
\begin{aligned}
E_S{}' = E_S \cup & \\
& \{(v, \overline{p}, v) : (v, p, v) \notin E_S\} \cup \\
& \{(v, \overline{r}, w) : \ v \neq w \ \& \ (v, r, w) \notin E_S\}
\end{aligned}
$$

with $\overline{p} \in \overline{P}$ and $\overline{r} \in \overline{R}$. Now graph-based GRE proceeds as before: negated edges may be selected if they rule out confusables. Again, the theoretical complexity of the algorithm is not altered, but the scene graph may grow drastically. If our initial scene S contains c nodes, and our initial label set L contains n properties and m relations, then the scene graph with negated edges S' will contain $c.n + c.(c-1).m$ edges. That is: we get a *dense* graph in which every possible edge is present with either a positive or a negative label.

In the above, we have presented a treatment of negation based on extending the scene graph (i.e., on making a set of implicit properties explicit). Other treatments are possible, where the scene graph is left intact, and where the algorithm 'infers' a negative property where no positive property is found. Even though this is more efficient from a representational point of view, the conceptual difference is small. A difficult question, regardless of which of these strategies is chosen, is under which circumstances negations are to be chosen: in terms of the cost functions of section 2, what should be the *cost* of adding a negated property? Instead of discussing this issue, we will be content having established that negations can be treated in the graph-theoretical approach and explore the consequences of adding a further complication to GRE, which arises when disjunctions are taken into account as well. In this way, we will be giving the GRE algorithm full Boolean as well as relational coverage.

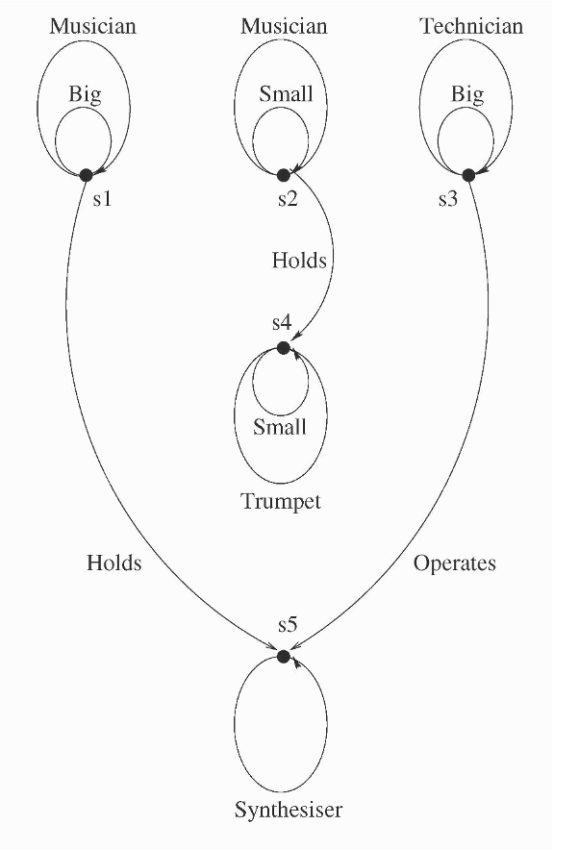

Figure 61. An extended scene graph, involving five objects

4. Boolean Descriptions

To put the graph-based approach to the test, let us now move on to a more challenging task: references using an arbitrary number of Boolean operators. To keep the problem manageable, we will limit the presentation to the case where gradable properties and salience do not play a role. (See the final section for brief discussion, however.)

For the discussion in this section, it will be convenient to use a slight extension of our example domain, as depicted in Figure 4. Suppose that the target for GRE is the set $T = \{s_1, s_2, s_3\}$. Basic GRE fails to find a unique reference to T, since there is no set of properties shared by all elements in T. If negative properties are taken into account

(see section 3.4), it turns out that we can characterize this set by $\overline{\text{Trumpet}} \cap \overline{\text{Synthesiser}}$ (i.e., the set of objects that are neither trumpets nor synthesisers), which would be a strange description of this target set. This result is due to the fact that basic GRE does nothing else than *conjoin* atomic properties (i.e., intersect their extensions).[7] Recently, proposals have been made for allowing GRE algorithms to use any combination of Boolean operators (van Deemter, 2001; 2002; Gardent, 2002). Using these, we find a non-atomic (positive) property shared by all elements of $\{s_1, s_2, s_3\}$ in our basic example scene: they are all *either* musicians *or* technicians ($\{s_1, s_2, s_3\}$ = Technician $\cup$ Musician). In section 4.1 we will explore ways in which existing algorithms for Boolean GRE may be reformulated using graphs. This discussion will lead on to a wholly new algorithm which is more easily cast in a graph-theoretical mold (section 4.2).

4.1. *Applying an Incremental Algorithm to Boolean expansions of graphs*

Van Deemter (2001, 2002) describes an extension of the Incremental Algorithm, covering equivalents of all Boolean combinations. The basic idea is to incrementally apply the Incremental Algorithm to Boolean properties of growing complexity. Thus, we first apply the Incremental Algorithm to a version of the KB to which negations of properties have been added. If no unique description is found, the KB is enriched with binary disjunctions (i.e., disjunctions of two positive or negative properties), and the Incremental Algorithm is applied to the extended KB. The process is repeated, with each phase adding longer disjunctions: ternary and so on.[8] In this way, logical equivalents of all Boolean combinations are covered, by constructing Conjunctive Normal Forms, that is, conjunctions of disjunctions of literals. Thus, for example, phase 1 may conjoin the atomic property A with the negation $\overline{D}$, after which phase 2 may add the disjunction $\overline{G} \cup E$ (resulting in the description $A \cap \overline{D} \cap (\overline{G} \cup E)$). Since this process is incremental, no property is ever removed, so even if $\overline{G} \cup E = A \cap \overline{D}$, the properties accumulated during the first two phases are kept, leading to a description that is far longer than necessary, thus exaggerating a property of Dale and Reiter's Incremental Algorithm. (See van Deemter, 2002 for discussion.)

[7] Other targets exist, which can not be described uniquely at all using conjunction and negation alone, e.g., $T = \{s_2, s_3\}$.

[8] Note that we freely mix logical with set-theoretic terminology, the relation between which is well understood.

Algorithms of this kind may be mirrored using graphs if we are able to let these express disjunctive information. Recall that, in section 3.4, negations of atomic properties were added to the scene graph to allow the inclusion of negative properties into descriptions. We would now have to do something analogous for disjunctions, and to extend the scene graph with edges making implicit information explicit. For example, we would make explicit that s_1, s_2 and s_3 are all musicians or technicians, by adding an edge labelled 'Musician or Technician' (notation: [Musician | Technician]) to each of them. We could apply the graph-based GRE algorithm in phases, and for each new phase extend the scene graph. A given phase may either result in a uniquely referring description (i.e., the algorithm terminates successfully), or not, in which case the next phase is entered. If the last phase terminates unsuccessfully then no uniquely referring description is found. Thus:

phase 1 Apply graph-based GRE algorithm **makeReferringExpression** to the scene graph, after addition of negative edges.

phase 2 Add edges labelled with binary disjunctions to the scene graph, then apply the algorithm to the resulting graph.

phase 3 Add edges labelled with ternary disjunctions. Etcetera.

In each phase, the basic graph-based **makeReferringExpression** from Figure 2 is used. Note that if our target is the set $\{s_1, s_2, s_3\}$, a simple solution, involving only one property, is found in phase 2, selecting the edge labelled [Musician | Technician], which may be realized as 'the musicians and the technician'. (Note the reversal, by which 'and' expresses disjunction.) The algorithm can be flavoured in different ways; for example, one might decide to do a complete search within each phase (in the style of section 2), but to never undo the results of previous phases. Or alternatively, one could use heuristic search within each phase, and try to find the 'cheapest' solution given these limitations.

But is our premise correct? That is, can disjunctive information always be expressed by labelled directed graphs of the kind that we have been discussing? Let us see how this could work, focussing on the case of binary disjunctions, and focussing on first principles rather than representational economy, for clarity. As was the case for negations, first we have to define a new class of labels, L_{dis2}. Naturally, the new labels will be composites of existing labels, for example $[l \mid l']$ (also written as $l \mid l'$) will be the label denoting the disjunction of the literals denoted by the labels l and l':

$$L_{dis2} = L_{neg} \cup \{[l \mid l'] : l, l' \in L_{neg}\},$$

where l and l' are of the same arity (i.e., both are properties or both are relations). This would make [Musician | Technician] a label, and also [Hold | $\overline{\text{Operate}}$], for example. With the newly extended set of labels in place, let us see how the scene graph may be extended. If $S = \langle V_S, E_S \rangle$ is a scene graph (possibly containing negative as well as positive edges), then the new scene graph with disjunctive edges is $S' = \langle V_S, E_S{}' \rangle$, where (for $[l \mid l'] \in L_{dis2}$) the following holds:

$$\begin{aligned} E_S{}' = E_S \cup \\ \{(v, [l \mid l'], v) : (v, l, v) \in E_S \vee (v, l', v) \in E_S\} \cup \\ \{(v, [l \mid l'], w) : (v, l, w) \in E_S \vee (v, l', w) \in E_S\} \end{aligned}$$

So far, everything appears to be as it should. Unfortunately, however, this treatment leaves some types of disjunctions uncovered. The simplest problem is posed by mixtures between properties and relations, such as 'is a technician or holds a trumpet'. Cases like this might be accommodated by creating 'mixed' labels, which disjoin a property and a relation (that is, by dropping the requirement, in the definition of L_{dis2}, that disjoined labels must have the same arity). An example would be the label [Technician | Hold], which could now label both looping and non-looping edges. Though this pairing of labels of different arity is slightly counterintuitive, there is no technical obstacle against it.

There is a more difficult problem, however, which resists this type of fix. Consider our running example, depicted in Fig. 6. It ought to be possible to describe the target $\{s_2, s_3\}$ by means of the disjunctive relation 'holds a trumpet or operates a synthesiser'. Disjunctions of such a complex kind, where the things that are disjoined are essentially structured rather than atomic, are not covered by disjunctive labels. (Note that the structure of the disjuncts can be arbitrarily complex, e.g., 'hold a violin which is old', 'hold a trumpet owned by a woman who ...'.) Extensions of the framework are possible; for example, one might create a new set of relational labels including, for example, Hold-Violin ('holding a violin'), Hold-Violin-Old ('holding a violin that is old'), and so on. This extension would allow us to form the disjunctive label [Hold-Trumpet | Operate-Synthesiser], giving rise to a distinguishing description of the target $\{s_2, s_3\}$. It is doubtful, however, that all possible cases could be tackled in this fashion, and the approach would seem to be misguided for various reasons.

Firstly, it would be fiendishly complicated to add all the right extensions to the scene graph without getting into an infinite loop. Secondly, by condensing all information into a single label, this approach would make superfluous the idea of letting the subgraph isomorphism algorithm find matches between complex graphs, going against the grain of Krahmer et al. (2003). In addition, it would tend to destroy everything that is simple and intuitive about graphs as representations of meaning, calling to mind efforts to make Venn diagrams more expressive by letting lines indicate that a given object can live in either of a number of different regions of the diagram; such a strategy allows succinct expression of some disjunctions, but becomes extremely cumbersome in other cases (Peirce 1896, Shin, 1994).

At least two types of responses are possible to this problem: one is to represent only some disjunctions explicitly (or even none at all), and to let the algorithm infer the others, analogous to what was suggested concerning negations at the end of section 3.4. The other is to explore a different type of algorithm which does not hinge on conjoining disjunctive properties, but on disjoining conjunctive properties.

4.2. *Generating partitions: an alternative algorithm for the generation of Boolean descriptions*

In the present section we offer an alternative algorithm for the generation of Boolean descriptions. Unlike the previous algorithm, this algorithm will not be based on an extension of the scene graph which, as have have seen, leads to problems. In fact, the algorithm leaves the original graphs intact, embedding the basic algorithm **makeReferringExpression** (from Figure 2) in a larger algorithm. Also unlike the previous algorithm, which generates *conjunctive* normal forms (CNF i.e., conjunctions of disjunctions of literals), the new algorithm generates *disjunctive* normal forms (DNF). More specifically, the algorithm generates disjunctions of conjunctions of literals under the added constraint that all conjunctions are mutually disjoint. In other words, the new algorithm uses *partitionings*. Whether DNFs (including partitionings) or CNFs are more useful as an output of GRE is a question that we will not resolve here, but which will be briefly taken up in section 4.3, where optimisation strategies are discussed.

The logical point to observe, in connection with the new algorithm, is that every Boolean combination is logically equivalent to a partitioning. This can be seen as follows. Firstly, every Boolean combination is equivalent to a formula in DNF, that is, a formula of the form $X_1 \cup ... \cup X_n$

```
describePartition (W, S) {
    n := |W|;
    k := 1;
    D := ⊥;

   for k = 1 to k = n do
       k-part := {ω : ω is a k-partition of W};
       for each ω ∈ k-part do
          for each part w ∈ ω do
             D′ := makeReferringExpression (w, S);
             if D′ ≠ ⊥ then D := D ∪ D′
             else failure /* try next ω */
          rof;
          return D; /* one k-partition could be described */
       rof;
       return failure;
    rof;
}
```

Figure 62. Sketch of an algorithm describing partitions

where each X_i (with $1 \leq i \leq n$) is of the form $Y_1 \cap ... \cap Y_m$, and where each Y_j (with $1 \leq j \leq m$) is a positive or negative literal. Secondly, any DNF formula can be rewritten as a partition, that is, a DNF whose disjuncts are all disjoint. The rewriting process is most easily demonstrated using an example. Consider the DNF $A \cup B \cup C$ (a disjunction of length three), and suppose this is *not* a partition, for example because $A \cap B$, $A \cap C$ and $B \cap C$ are all nonempty. This DNF can be rewritten as another disjunction of length 3, namely $A \cup (B - A) \cup (C - (A \cup B))$: each disjunct is adapted to make sure that all elements of its predecessors are removed. This procedure generalises without difficulty to disjunctions of length n.

We take these logical considerations to imply that, as a first step, it is sufficient to build an algorithm that generates partition-type descriptions wherever a distinguishing description is possible. The algorithm works as follows. Let $W = \{w_1, \ldots, w_n\}$ be the target, with $W \subseteq V_S$. We call a partitioning of W into k subsets (henceforth called *parts*) a *k-partitioning*. Figure 4.2 contains a description of the partition-based generation algorithm, using the function **makeReferringExpression** from Figure 2. (In Figure 4.2, the notation $D \cup D'$ designates the

unconnected graph that forms the union of the two connected graphs D and D'.) In a first iteration, the algorithm tries to describe W itself (as a 1-partitioning). If this fails, it attempts to describe one of the 2-partitionings of W. That is: for each 2-partition, we call the usual **makeReferringExpression** function from Figure 2 and apply it to each of its parts. As soon as one part cannot be described in the usual way, we move on to the next 2-partition. For our example target set, there are 3 partitions in 2 parts: $\{s_1\}, \{s_2, s_3\}$ and $\{s_2\}, \{s_1, s_3\}$ and $\{s_3\}, \{s_1, s_2\}$. Both parts of the latter 2-partition can be described in the usual way, as 'the technician' and 'the musicians' respectively. So, here the algorithm would terminate. In general, the partition algorithm will continue looking at k-partitions for ever larger values of k, until the target set is split up into singleton parts (i.e., until $k = n$, the number of parts equals the number of elements of the target set). Obviously, there is only one way to partition a target set W in singleton parts.

Note that this new algorithm, which covers Boolean combinations of properties and relations ("the bold technicians *and* the musicians who hold a trumpet") stays extremely close to the original graph-based algorithm outlined in section 2 and, unlike the approach outlined in the previous section, it does not require iterative extensions of the scene graph adding edges for ever more complex disjunctive labels. In addition, the approach is compatible with the treatment of gradable properties and salience. The algorithm, however, is computationally expensive in the worst case. The reason for this is that the number of partitions grows exponentially as a function of the size of the target set. In general, the number of k-partitionings of a target set with c elements can be determined using the second-order Stirling number $S(c, k)$ (Stirling 1730, see also e.g., Knuth, 1997:65). This number equals

$$S(c,k) = \frac{\sum^{k}_{j=0} \binom{k}{j} (-1)^j (k-j)^c}{k!}$$

Fortunately, there are various ways to speed up the algorithm. For example, one can use the notion of satellite sets, described in section 3.4 (which can be computed in linear time) to determine whether a purely conjunctive description for a given part exists; if not, the algorithm moves on to the next partitioning and tests whether a description for each of *its* parts exists. Alternatively, we could limit the set of partitions by relying on linguistic regularities. For example, by requiring that the properties corresponding with the different parts are of the same 'type'. Thus, for example, one might disallow "the musicians and the

small things", while allowing "the musicians and the technician" or "the trumpet and the synthesizer". Such a move, however, can sometimes result in a loss if descriptive power, because some sets may no longer be describable by the algorithm. In other words, the algorithm is no longer logically complete.

4.3. *Generate and Optimise*

We have simplified our discussion considerably by focussing on logical completeness only, that is, on the ability of a GRE algorithm to find a destinguishing description whenever there exists one. This means that we have largely disregarded the fact (noted in van Deemter, 2002 and more extensively in Gardent, 2002) that some algorithms deliver descriptions that are very lengthy and unnatural. In fact, it has been proven to be possible to construct a logically complete Boolean GRE algorithm in linear time, as long as the linguistic quality of descriptions is disregarded (van Deemter and Halldórsson, 2001): only when linguistic restrictions are placed on the output do things get complicated. It is well known, for instance, that finding *minimal* descriptions (e.g., Dale, 1992, Gardent, 2002) is computationally intractable, even when only conjunction is taken into account (Dale and Reiter 1995). The Boolean algorithms presented above do not guarantee minimal output, but the results are generally much shorter and easier to realize than those in van Deemter and Halldórsson (2001). The quality of the generated descriptions is ultimately an empirical issue which we cannot hope to address adequately within the confines of this paper. One point that we would like to stress here, however, is that Boolean descriptions can often be optimised automatically.

Consider the second, partition-based algorithm. Suppose the domain $\mathcal{D}$ is the set $\{d_1, d_2, d_3, d_4\}$, while the target T is $\{d_1, d_2, d_3\}$. Suppose Musician and Technician are the only properties, with Musician $= \{d_1, d_2\}$ and Technician $= \{d_2, d_3\}$. Then the algorithm based on partitionings behaves as follows: during the first phase there is only one partitioning (namely T itself), and it cannot be described. During the second phase, where 2-partitionings are considered, the partitioning $\{\{d_1, d_2\}, \{d_3\}\}$ may be chosen, whose two elements may be characterized as follows: the set $\{d_1, d_2\}$ equals the extension of the property Musician; the set $\{d_3\}$ equals the intersection of the extensions of Technician and $\overline{\text{Musician}}$: "the musicians and the technician that is not a musician". This would be overly verbose, since Musician $\cup$ (Technician

$\cap \overline{\text{Musician}}$) is logically equivalent with the simpler expression Musician $\cup$ Technician: "the musicians and the technician".

This illustrates that algorithms for Boolean GRE can be viewed as combining two different tasks: The first is to find a Boolean description that characterises the target, the second to determine whether there exists a *logically equivalent* characterisation that is more natural (e.g., briefer). Note that this makes the Boolean algorithms different from all the other ones discussed in this paper, where the second task was not relevant, but only another, similar task: determining whether there exist *non-equivalent* descriptions that nevertheless (i.e., given the domain and the extensions of properties) characterise the same target. In our original example domain, for instance, the property of being a *small musician* happens to be co-extensive with *holding a trumpet*, but this is something that can only be found out through an inspection of the domain. By contrast, inspection of the domain is not the natural way to find out that, for example, Musician $\cup$ (Technician $\cap \overline{\text{Musician}}$) is equivalent with Musician $\cup$ Technician. This separation into two different aspects of Boolean GRE suggests a 'generate then optimise' strategy reminiscent of the idea in Reiter (1990) in the context of simple GRE, which amounted to checking whether any set of properties, in a given description, may be replaced by another property without affecting the extension. In the current setting, where logical equivalence (not co-extensionality) is the issue, an obvious way to optimise is to use the type of algorithms that are used in chips design to simplify switching networks (van Deemter, 2002). The best known of these algorithms is the Quine-McCluskey algorithm, which performs the types of simplifications discussed here without difficulty (McCluskey, 1965).[9] A full discussion of the limitations of logical optimisation will not be offered here, since it is a more general issue of no particular relevance to the graph-theoretic approach.

5. Discussion

We have shown how labelled directed graphs can be used for generating many types of referring expressions, but we have also run into the limits of their usefulness. Starting from the basic graph-based algorithm of

[9] Although Boolean simplification is hard in general, algorithms like Quine-McCluskey are highly optimised and take little time in most cases that are likely to occur. Check `http://logik.phl.univie.ac.at/chris/qmo-uk.html`.

Krahmer et al. (2003), we have shown (1) how this algorithm can be used to generate descriptions of targets that are *sets*, (2) how it can accommodate *gradable* properties, including the property of being salient, and (3) how it can be extended to deal with *negative* literals. Our strategy has been the same in each of these cases: making information that is implicit in the original scene graph explicit by adding additional edges. After these relatively modest extensions, we have focussed on the generation of Boolean descriptions, arguably one of the most difficult GRE tasks. We have explored how the incremental Boolean algorithm of van Deemter (2002) might be recast in graphs, making implicit (disjunctive) information explicit by adding edges to the scene graph. Having seen that it is difficult to use this method for representing all the different types of disjunctions when relations are also taken into account (as in *"the men who are either holding a trumpet or playing a synthesiser"*), we were forced to consider alternative algorithms, and this has led to a simple alternative based on partitionings of the target set. This approach, which generates a different type of description from the incremental Boolean algorithm, outputs description graphs that appear to be natural and easy to realise in most cases. Having noted that, like its predecessors, the algorithm can sometimes generate unnecessarily lengthy descriptions, we have briefly explored the use of existing algorithms for the automatic simplification of Boolean expressions. We foresee that other problems with complex referring expressions, not dissimilar to the one arising when disjunctions and relations are combined (see section 4.1), may arise in other types of referring expressions (for example when booleans and quantifiers are combined), but an assessment of the challenges posed by these other expressions will have to await another occasion.

With the prospect of integrating the different GRE algorithms, plenty of new problems appear on the horizon. For example:

- **Relational and Boolean properties.** It is unclear how the generator should choose between different kinds of syntactic/ semantic complexity. Consider, for example, the addition of a *negation*, a *disjunction*, or a *relation* with another object. It is unknown, for example, which of the following descriptions should be preferred by the algorithm: "the musicians that are *not* technicians", "the violinists *and* the cellists", or "the musicians *in* the string section". New empirical research is needed to settle such questions. It is likely that the choice between different sets of properties can partly depend on ease of realization: "the string section", for example,

may be preferable to the "the violinists and the cellists" because it uses fewer words (rather than fewer concepts).

- **Salience and sets.** Suppose a quintet of musicians is performing on stage, thereby achieving a higher salience than all other people. Then two of them start to solo, thereby becoming even more salient. If existing accounts of salience are applied to sets, the generator can use the expression "the musicians" to describe the set of five *or* the set of two, which introduces an element of ambiguity into GRE that had always been kept out carefully.
- **Salience and vagueness.** We have shown that salience can be treated as (almost) 'just another' gradable property (section 3.3). But this is not only good news. Should, for example, "the *big* piano player" mean "the biggest of the piano players that are sufficiently salient"? Or "the most salient of the piano players that are sufficiently big"? Or is some sophisticated trade-off between size and salience implied? Expressions that combine gradable properties tend to be highly indeterminate in meaning. Determining under what circumstances such combinations are nevertheless acceptable is one of the many new challenges facing GRE.

Issues of this kind are to be addressed in our future research.

6. Acknowledgements

The work on this paper was done as part of the TUNA project, which is funded by the UK's Engineering and Physical Sciences Research Council (EPSRC) under grant number GR/S13330/01.[10]

References

Bateman, J., 1999, Using Aggregation for Selecting Content when Generating Referring Expressions, *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics* (ACL 1999), University of Maryland.

Chartrand, G. and O. Oellermann, 1993, *Applied and Algorithmic Graph Theory*, McGraw-Hill, New York.

Dale, R., 1992, *Generating Referring Expressions: Constructing Descriptions in a Domain of Objects and Processes*, The MIT Press, Cambridge, Mass.

[10] See `http://www.csd.abdn.ac.uk/research/tuna` for more information concerning TUNA.

Dale, R. and N. Haddock, 1991, Generating Referring Expressions involving Relations, In *Proceedings of the European Meeting of the Association for Computational Linguistics* (EACL 1991), Berlin, 161–166.

Dale, R. and E. Reiter, 1995, Computational Interpretations of the Gricean Maxims in the Generation of Referring Expressions, *Cognitive Science* **18**: 233–263.

van Deemter, K., 2000, Generating Vague Descriptions, *Proceedings of the First International Conference on Natural Language Generation* (INLG 2000), Mitzpe Ramon, 179–185.

van Deemter, 2001, Generating Referring Expressions: Beyond the Incremental Algorithm, in Procs. of 4th International Conf. on Computational Semantics (IWCS-4), Tilburg, 50–66.

van Deemter, K. and M. Halldórsson, 2001, Logical Form Equivalence: the Case of Referring Expressions Generation, *Proceedings of 8th European Workshop on Natural Language Generation* (EWNLG-2001), Toulouse.

van Deemter, K., 2002, Generating Referring Expressions: Boolean Extensions of the Incremental Algorithm" *Computational Linguistics* **28** (1): 37–52.

van Deemter, K. 2006, "Generating Referring Expressions that involve gradable properties". *Computational Linguistics* 32 (2).

Gardent, C., 2002, Generating minimal definite descriptions, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, (ACL 2002), Philadelphia, USA, 96–103.

Garey, M. and D. Johnson, 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H.Freeman.

Gibbons, A., 1985, *Algorithmic Graph Theory*, Cambridge University Press, Cambridge.

Horacek, H., 1997, An Algorithm for Generating Referential Descriptions with Flexible Interfaces, *Proceedings of 35th Annual Meeting of the Association for Computational Linguistics* (ACL 1997), Madrid, 206–213.

Knuth, D., 1997, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, 3rd ed., Addison-Wesley, Reading, MA.

Krahmer, E., S. van Erk and A. Verleg, 2001, A Meta-Algorithm for the Generation of Referring Expressions, *Proceedings of 8th European Workshop on Natural Language Generation* (EWNLG-2001), Toulouse, ??–??.

Krahmer, E. S. van Erk and A. Verleg, 2003, Graph-based Generation of Referring Expressions, *Computational Linguistics*, **29**(1): 53–72.

Krahmer, E. and M. Theune, 2002, Efficient context-sensitive generation of referring expressions, In: *Information Sharing*, K. van Deemter and R. Kibble (eds.), CSLI Publications, CSLI, Stanford, 223–264.

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

Mehlhorn, K., 1984, *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, EATCS Monographs on Theoretical Computer Science. Springer, Berlin and New York.

Peirce, C.S., 1896, *Collected Papers*, Vol. 4. Edited by Charles Hartshorne and Paul Weiss. Harvard University Press, Cambridge, Mass.

Reiter, E., 1990, The computational complexity of avoiding conversational implicatures, *Proceedings of 28th Annual Meeting of the Association for Computational Linguistics* (ACL 1990), 97–104.

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

Shin, S.-J., 1994, *The Logical Status of Diagrams.* Cambridge University Press, Cambridge, UK.

Stirling, J., 1730, *Methodus differentialis, sive tractatus de summation et interpolation serierum infinitarium*, London.

Stone, M., 2000, On Identifying Sets, *Proceedings of the First International Conference on Natural Language Generation* (INLG 2000), Mitzpe Ramon, 116–123.

Suppes, P., 1972, *Axiomatic set theory*, Dover Publications, New York.