Overgenerating Referring Expressions Involving Relations and Booleans

Sebastian Varges

Information Technology Research Institute,
University of Brighton, UK
`Sebastian.Varges@itri.brighton.ac.uk`

Abstract. We present a new approach to the generation of referring expressions containing attributive, type and relational properties combined by conjunctions, disjunctions and negations. The focus of this paper is on generating referring expressions involving positive and negated relations. We describe rule-based overgeneration of referring expressions based on the notion of 'extension', and show how to constrain the search space by interleaving logical form generation with realization and expressing preferences by means of costs. Our chart-based framework allows us to develop a search-based solution to the problem of 'recursive' relations.

1 Introduction

The task of generating referring expressions (GRE) can be characterized by the following question: given a domain model and a set of referents, how can we uniquely identify the referent(s) using natural language? This task has often been combined with the requirement to be minimal, i.e. to use only a minimal number of properties, and with considerations of computational complexity. Earlier work often centered around the 'incremental algorithm' [3] which originally dealt with selecting attributive properties but which can also be taken as a processing framework for selecting relations [2] and Boolean combinations of attributes for reference to sets [13]. A separate strand of recent work originates from the graph-based approach to GRE which provides an elegant treatment of relations [9] and which has also been extended to deal with Boolean combinations [14]. These approaches see GRE as a task separated from surface realization.

Previously, overgeneration has mostly been explored in the context of surface realization [10,1,15]. In this work, we apply it to the generation of referring expressions including both logical form construction (or 'description building', as we call it) and realization. First, we make sure that we can actually handle all the types of properties (attributes, relations, types) and their combinations (conjunction, disjunction, negation) we want to cover, and only then do we start weeding out unwanted solutions and address efficiency issues. Thus, at first we overgenerate vastly, giving priority to completeness of coverage over efficiency and correctness of all outputs. The advantage of this methodology is a separation of concerns: we can first concentrate on giving a consistent treatment to a wide

range of phenomena without worrying about efficiency and algorithmic complexity. After that we can concentrate on defining constraints and preferences, and search for 'good' solutions. At the implementation level, some of these constraints are applied in the rule-based part, some are implemented as filters on the output of the rule system.

In the next section, we introduce the general processing framework and the three levels of representation of our approach: domain representation, description building and realization. In section 3, we examine the output of a first version of the system and introduce a constraint designed to prevent the inclusion of 'redundant' information. In section 3.1 we address the issue of search for optimal referring expressions, followed by a discussion of the problem of 'recursive' relations. In section 4 we compare our work to previous work on GRE and in section 5 we discuss possible extensions of our approach.

2   Processing Framework and Representations

Overgeneration requires a flexible processing framework that allows one to produce a large number of candidate outputs and experiment with different search strategies. Chart-based algorithms offer just this: different agenda orderings can be used to model different search strategies, and intermediate results can be reused in order to avoid unnecessary recomputations. Chart generation algorithms [8] have often been applied to surface realization. We apply them to description building as well: starting from a domain specification, we first derive basic description edges, and then recombine these bottom-up using logical connectives. Employing overgeneration at the level of description building can lead to the generation of vast numbers of descriptions that cannot be realized. As a consequence, we closely couple description building with surface realization. Such a coupling has previously been advocated in [12]. In the context of our approach, this means that the chart algorithm attempts to immediately realize new description edges. Only descriptions that have been successfully realized can be used in further rule applications. Surface realizations of partial referring expressions are always combined compositionally (see section 2.3).

2.1   Domain Representation

Domain objects are defined by means of three types of properties: attributes, types and relations. A domain D contains a set of domain objects $\{o_1, o_2, ..., o_n\}$. As in [3], every domain object is required to have exactly one type t (e.g. $t(o_1) =$ `cup`) and has zero or more attributes and relations. Types can be arranged in a subsumption hierarchy, e.g. `artifact` $\succ$ `cup`. Attributes are defined by attribute value pairs where both attributes and values are atomic (e.g. $attr(o_1) =$ `colour:red`). Similar to the graph representation language of [9], relations are directed arcs between domain objects (e.g. $rel(o_1, o_2) =$ `in`). In the following, we illustrate our approach with an example domain similar to the one in [2]. It contains three cups (c_1, c_2, c_3), three bowls (b_1, b_2, b_3), a table (t_1) and a floor (f_1).
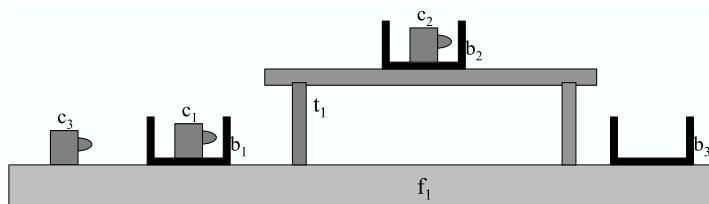
Fig. 1. Example domain

Cups c_1 and c_2 are 'in' b_1 and b_2, respectively. b_1, b_3 and c_3 are 'on' the floor but b_2 is 'on' the table which in turn is 'on' the floor. c_1 and c_2 are red, c_3 is blue and all other objects are black. All these objects are subtypes of `artifact` which in turn is a subtype of `domain_object`. Figure 1 shows the relative positions of the domain objects in the example domain (colours are not depicted).

2.2   Description Building

We regard constructing descriptions of domain objects as an inference process working over the domain model. Descriptions are represented as pairs of logical form and their extension which is defined as the set of domain objects for which the description is true. Logical forms contain the properties of the domain model, combined by conjunction, disjunction and negation.

The first step in building up increasingly complex descriptions is to determine the extension of the explicitly defined domain properties excluding relations, for example $<type(cup), \{c_1, c_2, c_3\}>$ or $<attr\_val(colour : red), \{c_1, c_2\}>$. The subsumption hierarchy is used to compute the extension of the supertypes of the domain objects as well, for example $<type(artifact), \{c_1, c_2, c_3, b_1, b_2, b_3, t_1, f_1\}>$. This allows us to deal with all types in a uniform way.

The main description building phase takes the basic description edges and recursively generates new description edges. We employ the following rules:

1. Conjunction: given two description edges d_1 and d_2, generate a new edge d_3 whose extension ext_{d_3} is the intersection $ext_{d_1} \wedge ext_{d_2}$, for example $<(type(artifact) \wedge attr\_val(colour : blue)), \{c_3\}>$.
2. Disjunction: given two description edges d_1 and d_2, generate a new edge d_3 whose extension ext_{d_3} is the union $ext_{d_1} \vee ext_{d_2}$, for example $<(type(floor) \vee type(table)), \{f_1, t_1\}>$.
3. Negation: given a description edge d_1, generate a new edge d_2 whose extension ext_{d_2} is the complement $\overline{ext_{d_1}}$, for example $<\neg attr\_val(colour : black), \{c_1, c_2, c_3\}>$.

Relations are introduced in a separate rule. Generally, we see relations as combining two existing descriptions. For example, *the cups* can be combined with *the bowls* through relation *in* into *the cups in the bowls*. Thus, we define

a rule that takes two description facts and establishes whether a given relation holds between them. This allows us to introduce both positive and negated relations in the same rule. Since our descriptions are defined by exactly one extension, we have to make a decision when building up descriptions involving relations: do we focus on the domain or range of the relation? (Given our over-generation approach, we generate relation edges for all cases, but other strategies are possible.) Relations seem to have similarities to conjunctions in that they add constraints to the description. In the above example, extending *the cups* to *the cups in the bowls* reduces the size of the focus extension of 'cups' from three to two elements. Introducing negative relations has the same constraining effect: the focus extension of *the cup not in the bowl* leaves only one element, c_3. We define the following relation introducing rule:

4. Given two description edges d_1 and d_2 and a relation name rel:

  – let extension $ext_{dom,pos}$ contain all $o_i \in ext_{d_1}$ that are in domain of rel with at least one member of ext_{d_2} in its range.
  – let extension $ext_{dom,neg}$ contain all $o_{ii} \in ext_{d_1}$ that are *not* in domain of rel with any member of ext_{d_2} in its range.
  – let extension $ext_{ran,pos}$ contain all $o_j \in ext_{d_2}$ that are in range of rel with at least one member of ext_{d_1} its domain.
  – let extension $ext_{ran,neg}$ contain all $o_{jj} \in ext_{d_2}$ that are *not* in range of rel with any member of ext_{d_1} in its domain.

4.1 generate a new description edge d_3 with extension $ext_{dom,pos}$ and focus on domain of rel, for example $<in(FOCUS(type(cup)), type(bowl)), \{c_1, c_2\}>$.
4.2 generate a new description edge d_5 with extension $ext_{dom,neg}$ and focus on domain of rel, for example $<\neg in(FOCUS(type(cup)), type(bowl)), \{c_3\}>$.
4.3 generate a new description edge d_4 with extension $ext_{ran,pos}$ and focus on range of rel, for example $<in(type(cup), FOCUS(type(bowl))), \{b_1, b_2\}>$.
4.4 generate a new description edge d_6 with extension $ext_{ran,neg}$ and focus on range of rel, for example $<\neg in(type(cup), FOCUS(type(bowl))), \{b3\}>$.

Since relations are represented by *directed* arcs, relation rule 4 needs to test d_1 and d_2 at both domain and range of a relation. The relation rule introduces positive relations if the relation holds for at least one pair of domain objects. Negated relations use the complements of the extension of the positive cases at the respective relation 'ends', i.e. $ext_{dom,neg} = \overline{ext_{dom,pos}}$ and $ext_{ran,neg} = \overline{ext_{ran,pos}}$.

Description building using relation rule 4 (and also rules 1-3) is subject to the constraint that the (focus) extension should not be the empty set. For example, let us assume a domain consisting of two bowls containing a cup each. We should be able to generate *the cups in the bowls* but we should not produce *the cup not in the bowl* as there is none. Rule 4 computes an empty focus extension in this case and we 'block' the resulting description. However, if we add another empty bowl, we can generate *the bowl not containing the cup* while still excluding *the cup not in the bowl*. Requiring description extensions to be non-empty seems

a reasonable restriction for GRE since in the end we want to refer to at least one domain object. Due to the availability of a single extension in descriptions involving relations, the results of the relation rule can be combined further in rules 1-3 like all other descriptions.

2.3   Realization

Grammar rules and lexicon map description facts to surface forms. Basic description edges for types and attributes are realized by means of a phrasal lexicon. Complex descriptions are realized by collecting the realizations of the component descriptions and combining them bottom-up. The derivation structure of descriptions resulting from rules 1-4 is mirrored by the syntactic structure of the realizations. For example, when we conjoin attribute value edge `colour:red` with type edge `cup` into a new conjunction edge in the description builder, this is mirrored in the realizer by combining the realizations of the two descriptions, adjective edge *red* and Nbar edge *cup*, into Nbar *red cup*. Every realized description contains at least one type for the head noun of the top-level NP. In contrast to the incremental algorithm [3], this is a result of the grammar rules and the close coupling of description building and realization rather than a hard-wired constraint.

Relations are realized by combining already realized NPs for domain and range of the relation with a verb or preposition provided by the lexicon, possibly introducing a negation, for example *the cup* + *not* + *in* + *the bowl*. The same relation can be realized by different syntactic categories depending on the focus of the description. For example, relation 'in' is realized as verb gerund *containing* rather than a preposition if the focus is on the range of the relation.

3   Overgeneration and Search

The currently implemented system uses 34 description building rules (distinguishing between negation of different description types, amongst others) and 21 rules for NP generation. The rules are expressed as productions in a production system [4], the knowledge base of which serves as the chart. In the following, we first look at the output of the system using bottom-up breadth-first search without specific referents in mind. After that we define goal referents and refine the search strategy.

Figure 2 shows some output NPs generated from the example domain of section 2.1. The last column ('complexity') measures the number of description rules involved. Introducing a negated relation increases the complexity by 2. The first 3 NPs in figure 2 are realizations of basic descriptions. The following NPs are complex descriptions. 4 is an example of a disjunction. 5 and 6 exhibit negations which, as 6 shows, can be applied repeatedly. 7 and 8 use adjectives and 9-15 use relations of increasing complexity.

One problem of the generated NPs involves 'redundant' information. This is most obvious in cases of direct repetitions, for example NP 7. Moreover, in

	Realization	Extension	Complexity
(1)	the cups	c1 c2 c3	1
(2)	the bowls	b1 b2 b3	1
(3)	the artifacts	c2 c3 b2 b3 c1 b1 t1 f1	1
(4)	the table or the cups	t1 c1 c2 c3	3
(5)	the non-blue domain objects	c1 c2 b1 b2 b3 t1 f1	4
(6)	not not a cup	c1 c2 c3	3
(7)	the black black bowls	b1 b2 b3	5
(8)	the black non-blue table	t1	6
(9)	the bowl on the table	b2	3
(10)	the table under the bowl	t1	3
(11)	the cups in the bowls	c1 c2	3
(12)	the bowls containing the cups	b1 b2	3
(13)	the bowl not containing the cup	b3	4
(14)	the cups in the bowls containing the cups in the bowls containing the cups	c1 c2	9
(15)	the bowls containing the cups in the bowls containing the cups in the bowls	b1 b2	9

Fig. 2. Some realized NPs of overgeneration experiment

10 *the table* would have been sufficient to describe the referent. As Dale and Reiter [3] point out, 'redundant' information may violate the Gricean maxim of quantity, leaving the reader wondering why information such as *under the bowl* (NP 10) was included. Since description building is extension-based, it is quite natural to require description building rules to reduce the extension size. This only applies to operations that are intersective in nature, i.e. conjunctions and relations. Concerning relations, additional information should constrain the focus extension, preventing, for example, NP 10 because *under the cup* does not reduce the extension of *the table*. Conjunctions of descriptions, apparently symmetric, also seem to have a focus extension when we consider how they are realized: in *the red cups*, *red* should only constrain the number of cups. If we also require the extension of 'red' to be reduced, we cannot generate *the red cups* in our example domain because all red objects are cups. This observation again points to a close coupling of content determination and realization: the syntactic head corresponds to the focus extension used in the description builder. Applying the extension-reducing constraint prevents the generation of NPs 7, 8 and 10.

3.1   Filtering and Search for Optimal Referring Expressions

Our bottom-up breadth-first chart algorithm generates referring expressions in order of increasing complexity. Thus, it can stop as soon as the system has generated a description for some *intended* referent(s), knowing that there cannot be a less complex one. This is essentially the 'full brevity' algorithm described by Dale and Reiter [3]. What, however, if we want to use a different cost metric? For example, integration with surface realization allows one to define costs based on

	NO CONSTRAINTS			EXT. CONSTR.			EXT. CONSTR. + EQUI.		
c	edges	realizations	secs	edges	realizations	secs	edges	realizations	secs
1	33	21	1	33	21	1	33	21	1
2	51	32	2	51	32	1	51	32	2
3	128	78	3	112	67	3	96	54	3
4	272	158	6	219	122	6	155	76	4
5	887	443	37	521	261	19	277	128	9
6	2042	979	189	1073	504	64	372	169	15

Table 1. Performance figures for different levels of NP complexity ('c')

the number of words. These alternative metrics can be incorporated by making costs explicit and ordering the agenda in terms of increasing cost. Under the assumption of monotonically increasing costs, we know that a current best NP describing the intended referent(s) is indeed the optimal one according to the cost metric once all other descriptions on the agenda have a higher cost. This is similar to the branch-and-bound algorithm used in graph-based GRE [9]. We can reduce the search space further by defining an admissible heuristic for estimating the minimal additional cost of the descriptions on the agenda. For example, we may know that further rule applications will add at least one more word to the realizations. We then use the increased minimal cost to compare descriptions on the agenda to the cost of the current best solution. The result is A^* search [11] for intended referents.

In addition, we can filter descriptions by means of equivalence classes. These are defined by the extension of the descriptions and the syntactic features of their realization. For example, an equivalence class for c_3 and syntactic category NP may contain descriptions realized as *the blue cup* and *the blue artifact*, amongst others. Dropping one of these will not reduce the referential capabilities of the generator since the alternatives are substitutable. Which one is dropped depends on the cost metric: for example, if costs reflect the distance from 'basic level types' [3], we can eliminate *the blue artifact*.

Table 1 shows some performance figures for the example domain for different levels of NP complexity ('c'). The number of edges includes both description and realization edges. Up to a complexity of 5 the simplest version of the implemented system ('NO CONSTRAINTS') generates 443 realizations within 37 seconds. The constraint that requires extensions to be reduced ('EXT. CONSTR.') decreases the number of realizations to 261 in 19 seconds. Adding filtering based on equivalence classes ('EXT. CONSTR. + EQUI.') leads to a further reduction to 128 realizations in 9 seconds. The system generates *the table or the floor* for goal referents t_1 and f_1 in 2 seconds, and finds *the cup in the bowl on the floor* for referent c_1 within 10 seconds. We believe that there is a lot of room for further efficiency gains which, as we argued earlier, should be seen as a separate issue from the generative capabilities of the system. For example, when referring to an individual, we can disable the disjunction rule. This reduces generation time for referent c_1 from 10 to 4 seconds. Furthermore, the use of a rule system that

runs on top of Java involves considerable overhead that could be removed by employing a lower-level implementation language.

3.2   The 'Recursion' Problem for Relations

Dale and Haddock [2] describe a problem of infinite regress or 'recursion' that occurs when extending the incremental algorithm to relations. The problem can be recreated in our example domain. Let us assume that our goal is to describe the cup that is in the bowl on the table, i.e. c_2. An infinite regress occurs because the system can always prefix *the cups in* to a description of the bowls containing the cups (like 12 and 15 in figure 2) and *the bowls containing* to a description of the cups in the bowls (like NPs 11 and 14). This is possible because in each case the focus extension is reduced from 3 to 2 domain objects so that the extension-reduction constraint defined earlier is not violated.

The fact that the recursion problem can be re-created in our approach shows its generality. However, in a breadth-first style algorithm like ours, the problem is less severe because a solution is found in one of the parallel branches of the search tree, after which search can stop, knowing that the infinite recursive branch will result in a higher cost than some alternative solution obtained already. This search-based solution to the recursion problem is different from the proposal of Dale and Haddock in that it does not require any form of check for structural repetitions.

4   Comparison to Previous Work on GRE

Our domain representation language bears similarities to the graph approach of Krahmer et al. [9], the major difference being the addition of a subsumption hierarchy and a different representation of attributes. In contrast to the graph approach, we separate descriptions (logical forms) and domain model. This has the advantage of not having to express logical formulae in the domain representation language where graphs encounter problems when it comes to representing disjunctions, for example. A further difference is the integration with surface realization: this makes surface-based costs available for decisions about referring expressions. In contrast, the graph approach requires costs to be attached to the edges of the domain graph. Arguably, these costs are more difficult to obtain. Using the number of words as a cost metric as we did in this work is just a simple illustration of the possibilities of this integration.

Gardent [5] integrates logical form construction and surface realization in a constraint-based approach. Search finds solutions in 'increasing order of size'. Other search strategies and cost functions are not discussed. Gardent observes that the graph-based approach to domain representation becomes less intuitive when applied to relations of arity higher than two. This also applies to our domain representation language. However, propositional nodes may offer a way out. For example, we could use ternary 'in-between' and 'give' nodes.

In contrast to the incremental algorithm [3] and its extensions, our approach crucially involves search. It finds globally best solutions and can correct local decisions rather than cutting off choices. A further difference is the notion of 'distractors' used in the incremental algorithm, which in our terminology corresponds to the extension set minus any referents it contains.

Horacek [7] describes a best-first search algorithm that works by expanding a tree structure representing the state of the search. The algorithm uses an A^*-style notion of future minimum costs. However, it is not entirely clear if choosing the locally best expansion point can lead to globally non-optimal results. Integration with realization is not described.

5   Discussion and Future Work

The work presented in this paper is limited to the generation of purely referential expressions, ignoring attributive descriptions [6]. However, it should be possible to incorporate goals such as motivating somebody to make a purchase into the computation of preference scores, allowing for trade-offs between the preference for least complexity and other goals.

Furthermore, scalability is an important issue. Using larger domain models and additional grammar rules will increase the search space. However, we believe that these issues can be solved in practical applications for all cognitively plausible levels of NP complexity. On the other hand, we freely acknowledge that this has to be demonstrated in an implementation.

Costs such as NP complexity and surface length do not take the extension size or the presence of intended referents in the extension set into account. It could be argued that negation constitutes an 'indirect' way of arriving at a description of intended referents: the system first aims to describe the complement of the intended referents (or at least a subset of these), and then negates it. Incorporating search strategies that take these considerations into account is left to future work.

In ongoing work we address remaining errors of the output candidates. For example, when edges are combined, their extension changes. This in turn can result in the apparent need for non-monotonic changes of the corresponding realizations, for example trying to re-use *the bowls* to generate *the bowl (on the table)*. One technique to prevent such non-monotonic changes is to delay morphological realization [16]. Our current implementation uses shallow methods to 'adjust' morphology.

Another problem is ambiguity of surface forms, in particular when using logical connectives. For example, in *the musicians holding the drum or the instrument*, the musicians can 1) only hold the drum or 2) hold the drum or the instrument. In our current implementation, we introduce a comma before *or* to at least enforce one of the readings in such cases. This may not be a very principled approach but it should be noted that ambiguity from a parsing (reader) perspective is a general issue for NLG systems which is not confined to GRE or overgeneration approaches.

6   Conclusions

We presented a new approach to the generation of referring expressions. By using overgeneration, we can focus on how to consistently treat a wide range of phenomena rather than trying to prematurely optimize or restrict our algorithm. We found that at the level of logical form construction (description building), using the extension as the main 'interface' between descriptions results in a consistent treatment of arbitrarily complex descriptions involving positive and negated relations. Our search algorithm can be seen as a generalization of the full-brevity algorithm to arbitrary monotonic cost functions. We offer an alternative solution to the 'recursion' problem for relations identified by Dale and Haddock.

Acknowledgements

References

1. Bangalore, S., Rambow, O. 2000: Exploiting a probabilistic hierarchical model for generation. Proceedings of COLING-00.
2. Dale, R., Haddock, N. 1991: Generating referring expressions involving relations. Proceedings of EACL-91.
3. Dale, R., Reiter, E. 1995: Computational interpretations of the gricean maxims in the generation of referring expressions. Cognitive Science, 19:233-263.
4. Friedman-Hill, E. 2004. JESS - the Java Expert System Shell, Version 6.x. Sandia National Laboratories.
5. Gardent, C. 2002: Generating minimal definite descriptions. Proceedings of ACL-02.
6. Green, N., Carenini, G., Moore, J. 1998: A principled representation of attributive descriptions for generating integrated text and information graphics presentations. Proceedings of IWNLG-98.
7. Horacek, H. 2003: A best-first search algorithm for generating referring expression. Proceedings of EACL-03 (research notes).
8. Kay, M. 1996: Chart generation. Proceedings of ACL-96.
9. Krahmer, E., Verleg, A., van Erk, S. 2003: Graph-based generation of referring expressions. Computational Linguistics, 29(1):53-72.
10. Langkilde, I., Knight, K. 1998: Generation that exploits corpus-based statistical knowledge. Proceedings of COLING/ACL-98.
11. Pearl, J. 1984: Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley.
12. Stone, M., Doran, C. 1997: Sentence planning as description using Tree-Adjoining Grammar. Proceedings of ACL-97.
13. van Deemter, K. 2002: Generating referring expressions: Boolean extensions of the incremental algorithm. Computational Linguistics, 28(1):37-52.

---

[1] Towards a UNified Algorithm for the Generation of Referring Expressions.

14. van Deemter, K., Krahmer E. to appear: Graphs and Booleans: on the generation of referring expressions. Kluwer Academic Publishers.
15. Varges, S., Mellish, C. 2001: Instance-based natural language generation. Proceedings of NAACL-01.
16. Wilcock, G. and Matsumoto, Y. 1996: Reversible delayed lexical choice in a bidirectional framework. COLING-96.