AN ALGORITHM FOR GENERATING NON-REDUNDANT QUANTIFIER SCOPINGS

Espen J. Vestre
Department of Mathematics
University of Oslo
P.O. Box 1053 Blindern
N-0316 OSLO 3, Norway
Internet: espen@math.uio.no

ABSTRACT

This paper describes an algorithm for generating quantifier scopings. The algorithm is designed to generate only logically non-redundant scopings and to partially order the scopings with a given default scoping first. Removing logical redundancy is not only interesting per se, but also drastically reduces the processing time. The input and output formats are described through a few access and construction functions. Thus, the algorithm is interesting for a modular linguistic theory, which is flexible with respect to syntactic and semantic framework.

INTRODUCTION

Natural language sentences like the notorious

(1) *Every man loves a woman,*

are usually regarded to be *scope ambiguous.* There have been two ways to attack this problem: To generate the most probable scoping and ignore the rest, or to generate all theoretically possible scopings.

Choosing the first alternative is actually not a bad solution, since any sample piece of text usually contains few possibilities for (real) scope ambiguity, and since reasonable heuristics in most cases pick out the intended reading. However, there are cases which seem to be genuinely ambiguous, or where the selection of the intended reading requires extensive world knowledge.

If the second alternative is chosen, there are basically two possible approaches: To integrate the generation of scopings into the grammar (like e.g. in Johnson and Kay (90) or Halvorsen and Kaplan (88)), or to devise a procedure that generates the scopings from the parse output (like in Hobbs and Shieber (87)). In both cases, only structurally impossible scopings are ruled out, like the reading of

(2) *Every representative of a company saw most samples*

in which "most samples" is outscoped by "every representative" but outscopes "a company" (Hobbs and Shieber (87)).

Logically equivalent readings are *not* ruled out on either of these proposals. Hobbs and Shieber argue that

> "When we move beyond the two first-order quantifiers to deal with the so-called generalized quantifiers, such as "most", these logical redundancies become quite rare".

Theoretically, they become rare. But it may very well be that sentences with several occurrences of non-first-order generalized quantifiers are not very commonly used. On the other hand, sentences with several occurrences of existential or universal quantifiers may be quite common. What kinds of expressions that really resemble first-order quantifiers is of course a controversial question. But working natural language systems, with inference mechanisms that are based on first-order logic, often have to simplify the interpretation process by interpreting broad classes of expressions as plain universal or existential quantifiers. Thus, the gain of generating only non-equivalent scopings may be quite significant in practical systems.

Ordering of the scopings according to preference is also not treated on approaches like that of Hobbs & Shieber (87) or Johnson & Kay (90). Hobbs & Shieber (87) are quite aware of this, and give some suggestions on how to build ordering heuristics into the algorithm. On the approach of Johnson & Kay (90), scopings are generated with a DCG grammar augmented with procedure calls for "shuffling" and applying the quantifiers[1]. The program will return new scopings by backtracking. Because of the recursive inside-out nature of the algorithm, it seems difficult to preserve generation-by-backtracking if one wants to order the scopings.

[1]The quantifier shuffling method is essentially the same as in Pereira & Shieber (87), but correctly avoids the "structurally impossible" scopings mentioned above.

Scope islands: In English, only existential quantifiers may be extracted out of relative clauses. Notice the difference between

(3a) *An owner of every company attended the meeting.*

(3b) *A man who owns every company attended the meeting.*

A scoping algorithm *must* take this into account, since it will be very difficult to filter out such readings at a later stage. In the algorithm of Johnson & Kay (90), adding such a mechanism seems to be quite easy, since the shuffling and application of quantifiers are handled in the grammar rules. In the algorithm of Hobbs & Shieber (87), it is a bit more difficult, since the language of the input forms does not distinguish between relative clauses and other kinds of NP modifiers.

In general, any working scoping algorithm should meet as many linguistic constraints on scope generation as possible.

Modularity: The main concern of Johnson & Kay (90) is to build a grammar that is independent of semantic formalism. This is done by a DCG grammar using "curly bracket notation" to include calls to formalism-dependent constructor functions.

It is tempting to take this approach one step further, and let the generation of scopings be independent on *both* the syntactic and semantic theory chosen.

A MODULAR APPROACH

The algorithm I propose provides solutions to the four problems mentioned above simultaneously. It is an extension and generalisation of the algorithm presented in Vestre (87)[2].

In the following I will make the (commonly made) assumption that quantified formulas are *4-part objects*. I will occasionally use a simple language of generalized quantifiers, where the formula format is

$$\mathrm{DET}(x, \phi(x,\ldots), \psi(x,\ldots))$$

for determiners DET and formulas ϕ, ψ. DET will be referred to as the *determiner of* the quantifier, x is its *variable*, ϕ its *restriction*, and ψ is its *scope*. The term *quantifier* will usually refer to the determiner with variable and restriction.

[2]This paper is in Norwegian, I'm afraid. An English overview of the work is included in Fenstad, Langholm and Vestre (89), but the details of the scoping algorithm are not described there.

Treating quantifiers in this way, it is easy to rule out the "structurally impossible" scopings mentioned above because the formulas corresponding to the "impossible scopings" will contain free variables. For instance, in sentence (2), the variable of "a company" (say, y) will also occur in the restrictor of "every representative". So in order to avoid an unbound occurrence of that variable, "a company" must either have wider scope than "every representative" or be bound inside its restrictor.

The algorithm presupposes that a few *access* functions are included for the type of input structure[3] used. Further, a few *constructor* functions must be included to define the format of the logical forms generated.

The role of the main access function, *get-quants,* is to pick out the parts of the input structure that are quantifiers, and to return them as a list, where the list order gives the *default* quantification order. There are almost no limits to what kinds of input structures that may be used, but the quantifiers that are returned by the access functions must contain their restrictors as a substructure. Of course, using input structures that already contain such lists of quantifiers as substructures will make the implementation of *get-quants* almost trivial.

In the following, I will give some rather informal descriptions of the main functions involved. The algorithm has been implemented in Common Lisp.

AN OUTSIDE-IN ALGORITHM

The usual way to generate scopings is to do it inside-out: Quantifiers of a subformula are either applied to the subformula or *lifted* to be applied at a higher level.

On the approach presented here, generation is done outside-in, i.e. by first choosing the *outermost* quantifier of the formula to be generated. The motivation behind this unorthodox move is rather pragmatic: It makes it possible, as we shall see below, to implement non-redundancy and sorting in an easy and understandable way. It is also easy to treat examples like the following, presented by Hobbs & Shieber (87):

(4) *Every man I know a child of has arrived*

where "a child of..." cannot be scoped outside of "Every man", since it (presumably) contains a variable that "Every man" binds. Building formulas outside-in, it is trivial to check that a formula only contains variables that are already bound.

[3]The input structure will typically be output from a parser.

There may be other good reasons for choosing an outside-in approach; e.g. if anaphora resolution is going to be integrated into the algorithm, or if scope generation is to be done *incrementally*: Usually, the first NP of a sentence contains the quantifier that by default has the widest scope, so an outside-in algorithm is just the right match for an incremental parser.

The outside-in generation works in this way:

1. Select one of the quantifiers returned by *get-quants*.

2. Generate all possible restrictions of this quantifier by recursively scoping the restrictions.

3. Recursively generate all possible scopes of the quantifier by applying the scoping function to the input structure *with the selected quantifier (and thereby the quantifiers in its restriction) removed.* Note that *get-quants* is called anew for each subscoping, but it will only find quantifiers which have not yet been applied.

4. Finally, construct a set of formulas by combining the quantifier with all the possible restrictions and scopes.

THE BASIC ALGORITHM

I will not formulate a precise definition of the algorithm in some formal programming language, but I will in the following give a half-formal definition of the main functions of the algorithm as it works in its basic version, i.e. *with* neither removal of logical redundancy nor ordering of scopings integrated into the algorithm:

The main function is *scopings* which takes an input form of (almost) any format and returns a set of scoped formulas:

```
scopings(form) =
{ build-main(form) },    if form is quantifier free
{ build-quant(q,r,s) | q ∈ get-quants(form),
                       r ∈ scope-restrictions(q),
                       s ∈ scopings(form(get-var(q)/q)) }
                    otherwise
```

where *form(get-var(q)/q)* means *form* with *get-var(q)* substituted for *q*. The purpose of this substitution is to mark the quantifier as "already bound" by replacing it with the variable it binds. The variable is then used by *build-main* in the main formula.

The function *scope-restrictions* is defined by

```
scope-restrictions(quant) =
combine-restrictions({ scopings(r) :
                       r ∈ get-restrictions(q)})
```

where the role of *combine-restrictions* is to combine scopings when there are several restrictions to a quantifier, e.g. both a relative clause and a prepositional phrase. Roughly, *combine-restrictions* works by using the application-defined function *build-conjunction* to conjoin one element from each of the sets in its argument set.

This is the whole algorithm in its most basic version[4], provided of course, that the functions *build-main, build-quant, build-conjunction, get-quants, get-var* and *get-restrictions* are defined. These may be defined to fit almost any kind of input and output structure[5]

REMOVING LOGICAL REDUNDANCY

We now turn to the enhancements which are the main concern of this paper. We first look at the most important, the removal of logically redundant scopings. To give a precise formulation of the kind of logical redundancy that we want to avoid, we first need some definitions:

Definition

A determiner DET is *scope-commutative* if (for all suitable formulas) the following is equivalent:

(1) $DET(x, R_1(x), DET(y, R_2(y), S(x, y)))$
(2) $DET(y, R_2(y), DET(x, R_1(x), S(x, y)))$

A determiner DET is *restrictor-commutative* if (for all suitable formulas) the following is equivalent:

(1) $DET(x, R_1(x) \& DET(y, R_2(y), S_2(x, y)), S_1(x))$
(2) $DET(y, R_2(y), DET(x, R_1(x) \& S_2(x, y), S_1(x)))$

[4]In this basic version, the algorithm does exactly what the algorithm of Hobbs & Shieber (87) does when "opaque operators" are left out.

[5]In the actual Common Lisp implementation, substitution of variables for quantifiers is done by destructive list manipulation. This means that quantifiers must be cons-cells, and that the occurrence of a quantifier in the list returned by *get-quants(form)* must *share* with the occurrence of the same quantifier in *form*.

It is easily seen that both existential and universal determiners are scope-commutative, and that existential, but not universal, determiners are restrictor-commutative. In natural language, this means that e.g. *A representative of a company arrived* is not ambiguous, in contrast to *Every representative of every company arrived.* Typical generalized quantifiers like *most* are neither restrictor-commutative nor scope-commutative[6].

Since quantifiers are selected outside-in, it is now easy to equip the algorithm with a mechanism to remove redundant scopings:

> If the surrounding quantifier had a scope-commutative determiner, quantifiers with the same determiner *and which precede the surrounding quantifier in the default ordering* are not selected.

For example, this means that in *Every man loves every woman,* "every man" has to be selected before "every woman". The algorithm will also try "every woman" as the first quantifier, but will then discard that alternative because "every man" cannot be selected in the next step - it precedes "every woman" in the default ordering. For more complex sentences, this discarding may give a significant time saving, which will be discussed below.

The algorithm also takes care of the restrictor-commutativity of existential determiners by using the same technique of comparing with the surrounding quantifier when restrictions on quantifiers are recursively scoped.

PARTIALLY ORDERING THE SCOPINGS

Generating outside-in, one has a "global" view of the generation process, which may be an advantage when trying to integrate ordering of scoping according to preference into the algorithm. As an example, the implemented algorithm provides a very simple kind of preference ordering: A scoping is considered "better" than another scoping if the number of quantifiers occurring in a non-default position is lower.

It is supposed that the input comes with a default ordering, and that the application-specific function *get-quants* takes care of this. This default order may reflect several heuristics for scope generation; e.g. that the *of*-complements of NPs usually take scope over the whole NP (and thus should be lifted by default).

The trick is now to assign a "penalty" number to every sub-scoping. Every time several quantifiers can be chosen at a given step, the penalty is increased by 1 if a quantifier different from the default one is chosen. And every time a quantifier is constructed, its penalty is set to the sum of the penalties of the restrictor and scope subformulas. Thus, the penalty counts the number of quantifier displacements (compared to the default scoping). The main function of the Common Lisp implementation thus looks like this[7]:

```
(defun scopings (form)
    (let ((qlist (get-quants form)))
        (if qlist
            (prefer (use-quant (car qlist) form)
                    (use-quants (cdr qlist) form))
            (list (cons 0 (build-main form))))))
```

Here *prefer* is a function which increases the penalty of each of the scopings in its second list, and calls *merge-scopings* on the two lists. *Merge-scopings* merges the two lists with the penalty as ordering criterion. This function is used whenever needed by the algorithm, such that one never needs to reorder the scoping list. From the last function-call above, one can also see how the coding of penalties is done: Atomic formulas are marked with a zero in their *car*. This number is later removed, the penalty is always stored only in the *car* of the whole scoped formula.

SCOPE OF RELATIVE CLAUSE QUANTIFIERS

Whether it is a general constraint on English may be questionable, but at least for practical purposes it seems reasonable to assume that *no other quantifiers than the existential quantifier may be extracted out of a relative clause.*

The algorithm makes it easy to implement such a constraint. Since the quantifiers that can be used at a given step are given by the application-defined function *get-quants*, it is easy for any implementation of *get-quants* to filter out all non-existential quantifiers when looking for quantifiers inside a relative clause. Here some of the burden is put on the grammar: The parts of the input structures that correspond to relative clauses must be marked to be distinguishable from e.g. PP complements[8].

[6]To prove non-scope-commutativity of *most*, construct an actual example where *Most men love most women* holds, but *Most women are loved by most men* does not hold (with the default scopings)!

[7]For clarity, the mechanism for removing logical redundancy is left out here.

[8]One could also put *all* the burden on the grammar, if one wanted the structures to contain the quantifier list as a

THE NUMBER OF SCOPINGS

Hobbs and Shieber (87) point out that just by avoiding those scopings that are structurally impossible, the number of scopings generated is significantly lower than *n!*. For the following sentence, the reduction is from 8! = 40320 to "only" 2988:

(5) *A representative of a department of a company gave a friend of a director of a company a sample of a product.*

Of course, the sentence has only one "real" scoping! Since the algorithm presented here avoids logical non-redundancy by looking at the default order already when a quantifier is selected for the generation of a subformula, the gain for sentences like (5) is tremendous[9].

The above suggests that complexity for scoping algorithms is a function of both the number of quantifiers in the input, and of the structure of the input. The highest number of scopings is obtained when the input contains *n* quantifiers, *none of which are contained in a restriction to one of the others*. An example of this is *Most women give most men a flower*. In such cases, no quantifier permutations can be sorted out on structural grounds, so the number of scopings is *n!*.

For more complex sentences, the picture is fairly complex. The easiest task is to look at the case where the lowest number of scopings are obtained (disregarding logical redundancy), *when all quantifiers are nested inside each other*, e.g.

(6) *Most representatives of most departments of most companies of most cities sighed.*

It is easy to see that if N is the function that counts the number of scopings in such a sentence, then

$$N(n) = \sum_{k=1}^{n} N(n-k)N(k-1)$$

Here $N(n-k)N(k-1)$ is the number of subscopings generated if quantifier number k is selected as the outermost, the factors are the number of scopings of the restriction and scope of that quantifier, respectively. Of course, $N(0) = 1$.

It can be shown that[10]

$$N(n) = \frac{(2n)!}{n!(n+1)!}$$

Further, estimating by Stirlings formula for *n!*we get the following (rough) estimate:

$$N(n) \approx \frac{4^n}{(n+1)\sqrt{\pi\, n}}$$

The important observation here, is that that the number of scopings of the completely nested sentences no longer is of faculty order, but of "only" exponential order. This gives us a mathematical confirmation of the suspicion that the number of scopings of such sentences is significantly lower than the number of permutations of quantifiers. For sentences which contain two argument NPs and the rest of the quantifiers nested inside each of these, the number of scopings is also $N(n)$. For sentences with three argument NPs, it is somewhat higher, but still of exponential order.

COMPUTATIONAL COMPLEXITY

What is the optimal way to generate (an explicit representation of) the *n!* scopings of the worst case? The absolute lower bound of the time complexity, will necessarily be at least as bad as the lower bound on space complexity. And the absolute lower bound on space complexity is given by the size of an optimally structure-sharing direct representation of the *n!* scopings. Such a representation will only contain one instance of each possible subscoping, but it has to contain all subscopings as substructures. This makes a total of $n + n\cdot(n-1)+\ldots+n!$ subscopings. Factoring out *n!*, we get $n!(1 + 1/1! + 1/2! +\ldots+1/(n-1)!)$. Readers trained in elementary calculus, will recognize the latter sum as the Taylor polynomial of degree *n-1* around 0 of the exponential function, applied to argument 1, i.e. the sum converges to the number *e*. This means that the total number of subscopings – and hence the lower bound on space complexity – is of order *n!*.

Without *any* structure-sharing, the number of subscopings generated will of course be $n\cdot n!$. This is exactly what happens here: The algorithm presented is $O(n^2\cdot n!)$ in time and space (provided that no redundancy occurs). This estimate presupposes that *get-quants* is of order *n* in both time and space, even when less than *n* quantifiers are left (presumably this figure will be better for some im-

substructure. This seems difficult to do with a pure unification grammar, however.

[9]For this particular sentence, the single scoping is generated in less than 1/200 of the time required to generate the 2988 scopings of the same sentence with 'most' substituted for 'a'.

[10]See e.g. Jacobson (51), p. 19.

plementations of *get-quants*). By comparison, the Hobbs & Shieber algorithm is O(*n!*), by using optimal structure sharing.

Does this mean that the outside-in approach should be rejected? Note that we above only considered the non-nested case. In the nested case, the algorithm presented here gains somewhat, while the Hobbs&Shieber algorithm loses somewhat. In both cases, scoping of restrictions has to be redone for every new application of the quantifier they restrict. This means that in the general case, the Hobbs & Shieber algorithm no longer provides optimal structure sharing, while the algorithm presented here provides a modest structure sharing. Now, both algorithms can of course be equipped with a hash table (or even a plain array) for storing sets of subscopings (by the quantifiers left to be bound). This has been successfully tried out with the algorithm presented here. It brings the complexity down to the optimal: O(*n!*) in the worst case, and similarly to $O(4^n n^{-3/2})$ in the completely nested case. So, there is, at least in theory, nothing to be lost in efficiency by using an outside-in algorithm.

THE SINGLE-SCOPING CASE

What about the promised reduction of complexity due to redundancy checking? We consider the case where a sentence contains *n* un-nested existential quantifiers. Then the complexity is given by the number of times the algorithm tries to generate a subscoping, multiplied by the complexity of *get-quants*. When quantifier number *k* is selected as the outermost, *n-k* quantifiers are left applicable in the resulting recursive call to the algorithm. Let *S* be the function that counts the number of subscopings considered. We have:

$$S(n) = 1 + \sum_{k=1}^{n} S(n-k) = 2^n - 1$$

Thus, in the single-scoping case the algorithm is $O(n \cdot 2^n)$ for input with un-nested quantifiers (and even lower for nested quantifiers).

Although the savings will be somewhat less spectacular for sentences with more than 1 scoping, this nevertheless shows that removing logical redundancy not only is of its own right, but also gives a significant reduction of the complexity of the algorithm.

MODULAR THEORIES OF LINGUISTICS

The algorithm presented here is related to the work of Johnson & Kay (90) by its *modular* nature. As mentioned, the interface with the syntax (parse output) is through a small set of access functions (*get-quants, get-restrictions, get-var,* and *quant-type*) and the interface with the semantics (the output of the algorithm) is through a small set of constructor functions (*build-conjuction, build-main* and *build-quant*). The implementation thus is a convenient "software glue" which allows a high degree of freedom in the choice of both syntactic and semantic framework.

This approach is not as "nice" as that of Johnson & Kay (90) or Halvorsen & Kaplan (88), and may on such grounds be rejected as a *theory* of the syntactic/semantic interface. But the question is whether it is possible to state any relationship between syntax and semantics which satisfies my four initial requirements (non-redundancy, ordering, special treatment of sub-clauses and modularity), and which still is "beautiful" or "simple" according to some standard.

REFERENCES

Fenstad, J.E., Langholm, T. and Vestre, E. (1989): *Representations and Interpretations.* Cosmos Report no. 09, Department of Mathematics, University of Oslo.

Halvorsen, P.K. and Kaplan, R.M. (1988): *Projections and Semantic Description in Lexical-Functional Grammar,* Proceedings of FGCS'88, Tokyo, Japan. Tokyo: Institute for New Generation Systems; 1988; Volume 3: 1116-1122.

Hobbs, J.R. and Shieber, S.M. (1987): *An Algorithm for Generating Quantifier Scope.* Computational Linguistics, Volume 13, Numbers 1-2, January-June 1987.

Jacobson, N. (1951): *Lectures in Abstract Algebra.* D. van Nostrand Comp. Ltd., New York.

Johnson, M. and Kay, M. (1990): *Semantic Abstraction and Anaphora.* Proceedings of COLING 90.

Pereira, F.C.N. and Shieber, S.M. (1987): *Prolog and Natural-language Analysis.* CSLI Lecture Notes No. 10, CSLI, Stanford.

Vestre, E. (1987): *Representasjon av direkte spørsmål,* Cand. Scient. thesis (unpublished, in norwegian)