A Quantifier Scoping Algorithm without A Free Variable Constraint

Ian Lewin
Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
email: il@aipna.ed.ac.uk

Abstract

Three recent demands on quantifier scoping algorithms have been that they should be explicitly stated, they should be sound and complete with respect to the input sentence [Hobbs and Shieber 1987] and they should not employ a 'Free Variable Constraint' [Pereira 1989]. The first demand is for good academic practice. The second is to ask for an algorithm that generates all and only the possible scopings of a sentence. The third demand is for an algorithm that avoids appealing to the syntax of logical form in order to determine possible scopings. I present a modified version of [Hobbs and Shieber 1987], which simplifies its operation, and can be considered sound and complete, depending on what interpretations of English sentences are deemed possible. Finally, any doubts concerning the use of logical form syntax are avoided.

1 Introduction

[Hobbs and Shieber 1987] presented an algorithm to generate quantifier scopings from a representation of "predicate-argument relations and the relations of grammatical subordination" (pg 49). This representation is successively modified by a recursive algorithm until all the quantifiers present in the input have been dealt with and given scope over some part of the output. A sample input representation is,

1. loves(<a x woman(x)> <every y man(y)>)

where representations of quantified noun phrases, called complex terms, are left as arguments to the verb. A sample output is

2. (a x woman(x) (every y man(y) loves(x,y)))

which uses a four-part quantifier notation, and in which no complex terms are present. In converting 1) into 2) the recursive procedure may be called upon representations of intermediate format, eg

(a x woman(x) loves(x <every y man(y)>))

where a four part quantifier phrase has an embedded complex term.

The algorithm is claimed to be more successful than previous accounts in dealing with complex noun phrases such as "every representative of a company" and in coping with certain 'opaque' predicates such as negation. [1]

Two properties of an algorithm which Hobbs and Shieber (H&S) approve of are completeness and soundness. An algorithm with these properties might be used as a benchmark for other algorithms designed for efficiency or the use of heuristics governing the plausibility of the various interpretations. Unfortunately, demonstrating that H&S's algorithm is sound requires a semantics for the input language and the intermediate forms. That is not straightforward.

I present a modified algorithm which avoids such intermediate forms. The input to the algorithm consists of English syntax. The steps of the algorithm retrace steps through a truth definition for the input language. Clearly, the algorithm is sound and complete with respect to that. The algorithm is also sound and complete with respect to English, if you agree that the input

[1] They acknowledge [Keller 1986] as a similar solution

language fairly represents the actual language of English speakers. Furthermore, the algorithm is somewhat simpler than H&S's algorithm and makes no appeal to logical syntax. There is a Prolog implementation of the algorithm.

2 Quantification in Logic

Semantic theories generally recurse over the syntax of the object language. For example, following the procedure and notation of [Tennant 1978], [2] we say that

g satisfies "($\forall$ x f(x))"
iff for every α, g($x \rightarrow \alpha$) satisfies "f(x)"

Thus, the satisfaction of "($\forall$ x f(x))" is given in terms of the satisfaction of formulae of the form "f(x)". Truth is defined as satisfaction by the null assignment, N. Given the following axiom

g($x \rightarrow \alpha$) satisfies "f(x)" iff f'(α)

then we can produce the following proof

"($\forall$ x f(x))" is true
iff N satisfies "($\forall$ x f(x))"
iff for every α, N($x \rightarrow \alpha$) satisfies "f(x)"
iff for every α, f'(α)

Finally, formalising our meta-language gives

"($\forall$ x f(x))" is true iff ($\forall$ α f'(α))

This idea can be extended to structurally ambiguous sentences of English. Suppose C is some environment containing a complex term such as "<a y woman(y)>", then

g satisfies C(<all y woman(y)>)
if (All α g($y \rightarrow \alpha$) satisfies "woman(y)"
g($y \rightarrow \alpha$) satisfies C(y))

Here, C(y) indicates the environment C(<a y woman(y)>) with y replacing the complex term. The extension involves two key changes. First, we employ a four part notation in the meta-language. Let us say that (All x f(x) g(x)) abbreviates the English: for every object x such that f(x) holds, g(x) also holds. Secondly, we use a simple conditional rather than a bi-conditional in the rule. The reason for this is simply that an ambiguous sentence such as 1) is true in either of two conditions. The theory will predict

"(loves <a x woman(x)> <every y man(y)>)"
is true if
(a α woman'(α) (every β man'(β) loves'(α,β))

and also that
"(loves <a x woman(x)> <every y man(y)>)"
is true if
(every α man'(α) (a β woman'(β) loves'(α,β))

We ensure 1) is not true in any other conditions by adopting a general exclusion clause that a sentence is not true except in virtue of the clauses of the given theory.

3 Comparison and Illustration

The primitive operation of our algorithm will be to apply a complex term to a formula containing it, e.g. to apply <q x r(x)> to p(<q x r(x)>). The result of application is a new four part quantifier expression whose first two parts are q and x, whose third part is the result of recursing on r(x) and whose fourth part is the result of recursing on p(x) (the formula itself with the complex term replaced by the variable it restricts).
For example, by choosing <a x woman(x)> first in 1), the algorithm will construct a new expression derived from "a", "x" and recursions on "woman(x)" and "loves(x <every y man(y)>)". The first recursion will result in woman(x). The second will build yet another term from "every", "y" and further recursion on "man(y)" and "loves(x,y)". The final result will be

(a x woman(x) (every y man(y) loves(x,y)))

Clearly, by choosing <every y man(y)> first, the alternative reading of the sentence would have been produced. Quantifiers chosen earlier receive wider scope. We work our way through the formula outside-in. [Woods 1968] explained the advantages of a top-down treatment of quantified noun phrases.

[2] We assume g is an assignment from variables to objects dealing with all variables required. g($x \rightarrow \alpha$) is g modified so that x is assigned to α. Greek letters are reserved for meta-language variables.

The basic operation of H&S is similar. An application builds a four part term whose first two parts are q and x, whose third part is r(x) and whose fourth part is the formula with x replacing <q x r(x)>). The result is then recursed upon in order to deal with other complex terms in the formula.
Now consider complex noun phrases such as "every representative of a company". These are success cases for H&S. The new algorithm deals with them without alteration. For example [3]

3. arrived(<every x
and(rep(x),
of(x, <a y company$y$(y)>)>)

We allow "every" to take wide scope as follows. First, we construct a new term from "every","x" and recursions on "arrived(x)" and "and(rep(x), of(x, <a y company(y)>))". The recursion on "arrived(x)" simply produces "arrived(x)". The recursion on

"and(rep(x), of(x, <a y company(y)>))"

will lead us to construct a new term from "a", "y" and the results of recursions on "company(y)" and "and(rep(x),of(x,y))". These last two recursions are again simple cases, [4] resulting in

(a y company(y)
and(rep(x), of(x,y)))

for "and(rep(x), of(x, <a y company(y)>))". With this result, we can complete our analysis of 3 itself.

(every x
(a y company(y)
and(rep(x), of(x,y)))
arrived(x))

for the whole input.
In comparison, H&S use a much more complex mechanism. They do this because otherwise dealing with <a y company(y)> first results in

(a y company(y)
arrived(<every x and(rep(x), of(x,y))>)

and recursion on this produces

(every x and(rep(x), of(x,y))
(a y company(y) arrived(x)))

which is not the required reading of the sentence. It also contains a free variable. H&S therefore forbid the algorithm to apply complex terms which are embedded within other complex terms. Also, the restrictions of complex terms are recursively scoped with a flag set so that this call of the procedure returns partial results (still containing complex terms), as well as full results.

4 Negation

There are two readings of the sentence

4. Everyone isn't here

depending on whether "not" or "every" takes wider scope. In ordinary logic we have

"not(p)" is true
iff it is not the case that "p" is true

Suppose C is an environment containing an occurrence of "not", then

g satisfies C(..not..)
if it is not the case that g satisfies C(.. ..)

Here the formula on the right-hand-side is just that on the left, with the occurrence of "not" removed. The ambiguity in 4) arises in exactly the same manner as quantifier scope ambiguities. Using one rule (negation) before another (quantification) leads to wider scope for the first application.
In contrast, H&S analyse 4 syntactically as

not(here(<every x person(x)>))

and mark "not" as being opaque in its only argument. The rule for opaque arguments allows them to be scoped first thus giving H&S the narrow scope "every" reading.

[3] I assume H&S's syntactic analysis

[4] Actually, there is an issue concerning "and", forced on us by H&S's syntactic analysis. The issue is whether quantifiers can be extracted across conjunctions. For present purposes, I assume they can - indeed, that the recursive rule for "and" only applies when the environments C and D in "and(C,D)" contain no complex terms.

This use of the term "opaque" is somewhat non-standard since "not" is not usually considered to be opaque.

5 Pronouns

Introducing complex noun phrase led to increased complexity in the H&S algorithm. The introduction of structure such as

5. Every man saw a picture of himself

where "him" is bound by "every man" leads to yet more. Take the representation of 5. as

6. saw(<every x man(x)>,<a y picture(y,x)>)

Applying <every x man(x)> first, via the H&S mechanism, gives

(every x man(x) saw(x,<a y picture(y,x)>))

Application of <a y picture(y,x)> would now lead to "x" being free. H&S prevent this by stipulating that a complex term is applicable only if all free variables in the term are also free in the containing formula. [Pereira 1989] calls this 'The Free Variable Constraint' and complains of an appeal to logical syntax.
Our own methodology avoids this. First, note that 6) is supposed to be a purely syntactic structure. The occurrence of "x" in "picture(y,x)" represents the pronoun "himself", and the fact that "x" also occurs in "<every x man(x)>" represents the grammatical relation holding between "<every x man(x)>" and "himself". Coindexing is used here just to indicate certain grammatical relations. [5] The following notation is clearer.

saw(<every x man(x)>,
<a y picture(y,himself-x)>)

Now, we alter our quantification rule so that if C is an environment containing <q x r(x)>, our new term is constructed from "q", "x", and recursions on "r(x)" and C where <q x r(x)> and all embedded coindexed reflexives are replaced by "x".
Suppose we choose to apply

<a y picture(y,himself-x)>

first to 6). Then we construct our result from "a", "y" and recursions on both "saw(<every x man(x)>,y)" and "picture(y,himself-x)" ; the final recursion cannot proceed however, for we have no rule to interpret a reflexive in this position. There is no appeal to logical syntax, only English syntax.
The same holds true of non-reflexives as in

7. Every man saw a friend of his

where "every man" and "his" are co-indexed. [6]

6 Summary and Conclusion

A modification to the algorithm of [Hobbs and Shieber 1987] based on a hint from standard logical theory has led to a simpler algorithm, and one which makes no illegitimate appeal to the syntax of logical form. The algorithm is sound and complete with respect to the input language since it retraces the semantic definition of that language. The degree to which it matches our intuitions concerning English determines how good a contribution to natural language processing it is.

Acknowledgements

Helpful comments on this work have been made by Robin Cooper, Graeme Ritchie and audiences at both the A.I. Natural Language Group and the Cognitive Science Formal Semantics Workshop in Edinburgh.
The research was supported by SERC research studentship no. 88304590

References

[1] Hobbs J.R., and Shieber S.M., 1987, 'An Algorithm for Generating Quantifier Scopings' *Computational Linguistics* **13**,numbers 1-2, January-June 1987.

[5] This is one area where H&S's analysis is difficult to follow - what *is* the role of variables in the input and intermediate forms ?

[6] There are other uses of pronouns not treated in the version of the algorithm given here. For example, there is a possible deictic use of "his" in 7). Nor do we account for "donkey" pronouns such as *Every woman who saw a man disliked him.*

[2] Keller W., 1986, 'Nested Cooper Storage' *Natural Language Parsing and Linguistic Theory* edited by U.Reyle and C.Rohrer 432-447, Studies in Linguistics and Philosophy volume 35, Dordrecht Reidel, Dordrecht.

[3] Pereira F.C.N., 1989 'A Calculus for Semantic Composition and Scoping' *27th Annual Meeting of ACL* 26-29 June 1989 Vancouver, British Columbia, 152-160.

[4] Tennant N.W., 1978, *Natural Logic* Edinburgh University Press

[5] Woods W.A., 1968, Procedural Semantics for a Question Answering Machine *AFIPS Natl. Comput. Conf. Expo., Conference Proceedings* **33**, 457-471.