An Algorithm for Generating Quantifiers

Norman Creaney

Faculty of Informatics
University of Ulster at Coleraine
N Ireland
BT52 1SA

E-mail: n.creaney@ulst.ac.uk
Tel: +44 (0)1265 324502
Fax: +44 (0)1265 324916

Abstract:

Quantifiers, and their associated scoping phenomena are ubiquitous in English and other natural languages, and a great deal of attention has been paid to their treatment in the context of natural language analysis. Rather less attention, however, has been paid to their treatment in the context of language generation. This paper describes an algorithm for generating quantifiers in English sentences which describe small models containing collections of individuals which are inter-related in various ways. The input to the algorithm is, i) a model represented as a collection of facts and ii) an abstract description of the target sentence with gaps where the quantifiers should be.

Keywords: *lexical choice, realisation, quantifiers*

An Algorithm for Generating Quantifiers

Quantifiers, and their associated scoping phenomena are ubiquitous in English and other natural languages, and a great deal of attention has been paid to their treatment in the context of natural language analysis (Alshawi 1990, Creaney 1995, Grosz et al. 1987, Hobbs and Shieber 1987, Park 1995, Saint-Dizier 1984). Rather less attention, however, has been paid to their treatment in the context of language generation. This paper describes an algorithm for generating quantifiers in English sentences which describe small models containing collections of individuals which are inter-related in various ways.

A model is represented as a collection of facts like the following:

1
```
rep( r1 ).   sample( s1 ).   saw( r1, s1 ).
rep( r2 ).   sample( s2 ).   saw( r2, s1 ).
rep( r3 ).   sample( s3 ).   saw( r2, s2 ).
rep( r4 ).                   saw( r3, s2 ).
                             saw( r3, s3 ).
                             saw( r4, s1 ).
                             saw( r4, s2 ).
                             saw( r4, s3 ).
```

In model (1) there are four representatives and three samples and some of the representatives saw some of the samples. The algorithm generates suitable quantifiers to complete sentences of the form:

Q_R *representative(s) saw* Q_S *sample(s)*

where Q_R and Q_S can be arbitrary quantifiers like; *some*, *two*, *all*, *both*, *one of the*, *most*, etc. The algorithm also handles models containing more relationships than model (1) to generate sentences of the form:

Q_R *representative(s) of* Q_C *company(s) saw* Q_S *sample(s)*

Q_R *representative(s) saw* Q_S *sample(s) of* Q_P *product(s)*

Q_R *representative(s) of* Q_C *company(s) saw* Q_S *sample(s) of* Q_P *product(s)*

One of the most striking things about the problem is that there are generally a great many sentences which provide reasonable descriptions of any given model. For example, the following are all acceptable for model (1).

2 *Every representative saw a sample*

3 *Every representative saw at least one sample*

4 *Most representatives saw at least two samples*

5 *A representative saw every sample*

6 *At least one representative saw every sample*

7 *At least two representatives saw most samples*

It turns out that there are three distinct sources of this variation and they correspond to three different kinds of choices which are made in the generation algorithm. They are:

- quantifier scoping choices
- choice of focus sets
- choice of individual quantifiers constrained by the above two choices

A great deal has been written about the quantifier scoping problem for natural language analysis (Hobbs & Shieber 1987) and much of this is applicable to the generation problem in the sense that any particular description must assume some particular quantifier scoping arrangement. For example, sentence (2) assumes that "*Every representative*" has wide scope while sentence (5) assumes that "*every sample*" has wide scope, and the sentences are only satisfied in the model under these assumptions. In fact sentences (2), (3) and (4) all assume wide scope for "*representative*" while sentences (5), (6) and (7) all assume wide scope for "*sample*". The generation algorithm, of necessity, incorporates quantifier scoping.

The concept of a *focus set* has no correlate in the language analysis literature although it is similar to what Barwise and Cooper (Barwise and Cooper 1981) call a *witness set*. It has to do with choosing some particular subset of the model on which to base the description. Sentence (2) talks about all three representatives while sentence (4) talks only about the subset {r2,r3,r4}. This subset stands in the realtion *most* to the entire set of representatives {r1,r2,r3,r4} and is the *focus set* for R in sentence (4). The concept of a focus set will be made more precise below where dependency functions are discussed.

Inputs to the algorithm

The inputs to the algorithm are:

- a model like (1)
- a predicate argument structure for the target sentence

A predicate argument structure (PAS) is essentially an unscoped logical form of the form taken as input to Hobs and Shieber's algorithm (Hobbs & Shieber 1987). It makes explicit the relationships between predicates and there arguments but does not express any quantifier scope relationships. Sentence (2) has the following PAS:

saw(every(R,rep(R)), a(S,sample(S)))

and sentence (8a) has the following one.

saw(every(R, rep(R)∧of(R,a(C,com(C))),
some(S, sample(S)))

Each variable in a PAS has a quantifier and a *restriction* which restricts the values which it may take.

- S's restriction is sample(S) in both cases
- R's restriction is rep(R) in the first PAS and rep(R)∧of(R,a(C,company(C)) in the second

Since the algorithm generates quantifiers its input PASs are not exactly like these. Instead they have gaps where quantifiers should be:

saw(Q_1(R, rep(R)), Q_2(S, sample(S)))

saw(Q_3(R, rep(R) ∧ of(R, Q_4(C,com(C))),
Q_5(S, sample(S)))

and the purpose of the algorithm is to assign values to the Q_i's given some suitable model. It works by processing the PAS recursively and non-deterministically selecting quantifier scopings and focus sets at each level. Quantifiers are then generated based on the cosen focus set.

Quantifier scoping

Since the particular scoping framework underlying the generation algorithm is novel a brief explanation is appropriate. The orthodox approach to quantifier scoping is embodied in Hobbs and Sheiber's algorithm and it permits all permutations of quantifiers such that there are no unbound variables in the resulting logical form. For example, Hobbs and Shieber's algorithm produces the scopings (8b-f) for sentence (8a)[1]:

8 a *Every representative of a company saw some samples*
b R > C > S
c C > R > S
d S > R > C
e C > S > R
f S > C > R

where R > C > S indicates that "*Every representative*" outscopes "*a company*" which in turn outscopes "*some samples*". The missing permutation, R > S > C, is not permitted because it violates what has become known as *the unbound variable constraint.*

The scoping framework which underlies the generation algorithm recognises fewer scopings than (8). The relative scope of two quantifiers is only considered for variables

[1] We adopt the convention of naming variables with the first letter of the head noun with which they are associated (R=representative, C=company, S=sample) and using the symbol '>' to denote relative scope.

which are arguments to the same predicate. For example, R must be scoped relative to C because they are both arguments to the predicate *of*. The possibilities are R > C and C > R. Similarly, R and S are arguments to the predicate *saw* and may be scoped either R > S or S > R. The relative scoping of C and S is never considered directly because they do not participate directly in any single predication in the sentence. They may however end up with a relative scoping as a result of taking the transitive closure of other scoping relationships. For example, if it is decided that C > R and R > S, then clearly, since scope is transitive, C > R. In this framework the following scopings are allowed for (8a).

9 a R > S, R > C
  b S > R > C
  c C > R > S
  d S > R, C > R

It is clear from (8) and (9) that this framework undergenerates in comparison with Hobbs and Shieber's. However, in the context of language generation, undergeneration is not necessarily a serious problem, provided that there is the ability to adequately describe any model. In fact there is an argument to be made in favour of a scoping framework which undergenerates with respect to Hobbs and Shieber's as a general approach to quantifier scoping (Park 1995). This is the subject of a future paper.

Dependency functions, partitions and focus sets

Each variable in a PAS has a *candidate set* which is defined by its restriction and the model under consideration.

***Definition 1**: candidate set*
A variables ***candidate set*** *is the set of individuals from the model which satisfy the variables restriction.*

For the PAS:

saw(every(R,rep(R)), a(S,sample(S)))

and model (1), R's candidate set is {r1,r2,r3,r4} and S's is {s1,s2,s3}.

When we say that "*Every representative*" has wide scope we are saying that there is a function which maps R's candidate set onto the power set of S's candidate set. This *dependency function* is computed from model (1) and is exhaustively listed in (10) below.

10 saw: {r1,r2,r3,r4} → power({s1,s2,s3})
  r1 → { s1 }
  r2 → { s1, s2 }
  r3 → { s2, s3 }
  r4 → { s1, s2, s3 }

Alternatively, if S is given wide scope the following dependency function is computed.

11 saw: {s1,s2,s3} → power({r1,r2,r3,r4})
  s1 → { r1, r2, r4 }
  s2 → { r2, r3, r4 }
  s3 → { r3, r4 }

Focus sets were discussed briefly above and are made more precise now in the context of dependency function partitions.

Any dependency function can be partitioned by choosing a arbitrary subset of the mappings it contains as its *focus*, the remainder being its *complement*. Of course, the domains and ranges of these sub-functions are appropriately adjusted.

The partitions (12a-c) are among the possible partitions of dependency function (10).

12 a focus: saw: {r1} → power({s1})
  r1 → { s1 }

  compt: saw: {r2,r3,r4} →
    power({s1,s2,s3})
  r2 → { s1, s2 }
  r3 → { s2, s3 }
  r4 → { s1, s2, s3 }

  b focus: saw: {r2,r3} → power({s1,s2,s3})
  r2 → { s1, s2 }
  r3 → { s2, s3 }

  compt: saw: {r1,r4} → power({s1,s2,s3})
  r1 → { s1 }
  r4 → { s1, s2, s3 }

c focus:saw:{r2,r3,r4} → power({s1,s2,s3})
r2 → { s1, s2 }
r3 → { s2, s3 }
r4 → { s1, s2, s3 }

compt: saw: {r1} → power({s1})
r1 → { s1 }

***Definition 2**: focus set*
*The **focus set** for a variable, given a particular partition is either:*
- *the domain of the partition's focus*
- *the union of the range of the partition's focus*

depending on the variable of interest.

For example, (12a-c) define the following candidate sets for R and S.

{ r1 }	{ s1 }
{ r2, r3 }	{ s1, s2, s3 }
{ r2, r3, r4 }	{ s1, s2, s3 }

It is useful to note that a variable's candidate set is related to an unpartitioned dependency function in exactly the same way that its focus set is related to the focus of the partitioned function. These relationships are illustrated in appendix 1.

Individual quantifiers are selected for generation on the basis of dependency function partitions. For example, the descriptions (13a-c) are licensed by the partitions (12a-c) respectively.

13 a i *A representative saw a sample*
ii *Exactly one representative saw exactly one sample*

b i *At least half the representatives saw exactly two samples*
ii *Exactly two representatives saw exactly two samples*
iii *Two representatives saw most samples*

c i *Exactly three representatives saw at least two samples*
ii *Three representatives saw some samples*

The mapping from partitioned dependency function to quantifiers is non-deterministic as (13) shows. For instance, partition (12b) gives, at least, the three sentences (13bi,ii,iii).

Not all sentences provide equally good descriptions of the model but they are all true in it. For example, (13ai) is true in (1), assuming "*a*" means *at least one*, but is not very informative. Bigger focus sets tend to give more information and sound more natural however the generation algorithm is concerned only with presenting alternatives and not with selecting between them.

Generating quantifiers

The process of generating quantifiers takes place after a scoping has been chosen and a dependency function has been constructed and partitioned, so that all decisions are made in the context of a particular partitioning of a particular dependency function.

Generation consists of going through the list of all possible quantifiers and checking whether or not each one is consistent with the appropriate variable in the current dependency function partition. Those which are consistent are then generated and those which are inconsistent are rejected. To check the consistency of a particular quantifier with a particular variable it is first necessary to compute the variable's candidate set, focus set, and *focus maximum* and *focus minimum*.

***Definition 3**: Focus maximum and minimum*
For a variable with wide scope the focus maximum (Fmax) and focus minimum (Fmin) are the same. They are simply the size of the focus set or, equivalently, the number of mappings in the focus of the dependency function.

e.g. R in (12a): Fmax=Fmin= |{r1}| = 1
R in (12c): Fmax=Fmin= |{r2,r3,r4}| = 3

For a variable with narrow scope the focus maximum (Fmax) is equal to the size of the

biggest member of the range of the focus of the dependency function.

e.g. S in (12a): Fmax = max(|{s1}|) = 1
S in (12c): Fmax=
max(|{s1,s2}|,|{s2,s3}|,|{s1,s2,s3}|)
= 3

The focus minimum is defined along the same lines as he focus maximum except that the minimum set size is taken.

e.g. S in (12a): Fmin= min(|{s1}|) = 1
S in (12c): Fmin=
min(|{s1,s2}|,|{s2,s3}|,|{s1,s2,s3}|)
=2

The checking procedure varies according to the type of quantifier under consideration where quantifiers are classified as one of three types *monotone increasing*, *monotone decreasing* or *cardinal* (Barwise and Cooper 1981).

- Monotone increasing quantifiers are those with an *at least N* interpretation. They include; *a*, *some_sing*, *some_plur*, *the*, *both*, *many*, *at least four*
- Monotone decreasing quantifiers are ones with an *at most N* interpretation. They include; *no*, *few*, *at most three*, *less than three quarters*
- Cardinal quantifiers are of the form *exactly N*

The check for monotone increasing quantifiers is simplest. The acceptability of each quantifier is as defined by a call to the following Prolog goal:

```
?- q_inc( Fmin, Nc, QUANT ).
```

where; Fmin = the focus minimum, Nc = |candidate set|, and the q_inc/3 relation is defined along the following lines.

```
14 % q_inc( +N1, +N2, ?Q ) defines Q as
   % - "at least N1 out of a possible N2"
   % e.g. "a man" means
   % - at least 1 man out of any number
   % "some men" means
   % - at least 2 men out of any number
   % "both men" means
   % - at least 2 men out of a possible 2

q_inc( 1, _, [a] ).
q_inc( 1, _, [some_sing] ).
q_inc( 1, _, [at,least,one] ).
q_inc( 2, _, [at,least,two] ).

q_inc( N, _, [some_plur] ) :- N > 1.
q_inc( N, M, [most] ) :- M < 2*N.

q_inc( 1, 1, [the] ).
q_inc( 2, 2, [both] ).
q_inc( 3, 3, [all,three] ).
q_inc( N, N, [all] ).
q_inc( N, N, [each] ).
q_inc( N, N, [every] ).
```

For R in (12a) the appropriate call is therefore:

```
?- q_inc( 1, 4, QR ).
```

which returns the following quantifiers: *a*, *some_sing*, *at least one*. Similarly, for S in (12a) the appropriate call is:

```
?- q_inc( 1, 3, QS ).
```

which returns the same set of quantifiers. Hence sentence (13ai) is generated, as is:

Some representatives saw a sample
At least one representative saw a sample
A representative saw some samples

and other similar sentences formed by selecting from the above quantifiers.

For a monotone decreasing quantifier the check depends on whether it is in wide scope position or narrow scope position. In narrow scope position the check is similar to the one for monotone increasing quantifiers except that:

- a different collection of quantifiers is checked - the monotone decreasing ones
- the focus maximum is input rather than the focus minimum.

```
?- q_dec( Fmax, Nc, QUANT ).
```

where; Fmax = the focus maximum, Nc = |candidate set|, and the q_dec/3 relation is defined along the following lines.

```
% q_dec( +N1, +N2, ?Q ) defines Q as
% - "at most N1 out of a possible N2"
% e.g. "no man" means
%    - at most 0 men out of any number
%    "few men" means
%    - at most half of all the men
%    "neither man" means
%    - at most 0 men out of 2

q_dec( 0, _, [no] ).
q_dec( 2, _, [at,most,two] ).

q_dec( N, M, [few] ) :- M < 2*N.

q_dec( 0, 2, [neither] ).
```

The check for monotone decreasing quantifiers in wide scope position is a little bit trickier. For example, to check the consistency of the quantifier *at most two* in "*At most two representatives saw a sample*", assuming R > S, the following checks need to be made.

- There must be a set of at most two of R's who may or may not have seen a sample. This entails checking that R's focus set contains exactly two members.
- All other R's outside this set must certainly not have seen a sample. This entails checking the complement of the dependency function to make sure that the quantifier *a* fails to be consistent with the variable S.

These checks are carried by calls to q_dec/3 an q_inc/3 with appropriate input values.

The check for cardinal quantifiers is defined in terms of two sub checks: one for a monotone increasing quantifier and one for a monotone decreasing. This follows from the observation that *exactly N* meant the same as (*at least N*)∧(*at most N*).

Embedded quantifiers

The preceding discussion concentrated on simple linguistic structures like (2-7) which contain one main verb and noun phrases with no recursive structure. The processing of a more complex structure like:

16 saw(Q_R(R, rep(R) ∧ of(R, Q_C(C,com(C))), Q_S(S, sample(S)))

is done by breaking it down into sub-structures (17) which are processed almost independently.

17 a saw(Q_R(R, ...), Q_S(S, sample(S)))
   b of($Q_{R'}$(R, rep(R)), Q_C(C, com(C)))

The variable R is assigned the quantifier Q_R in (17a) and the quantifier $Q_{R'}$ in (17b) but clearly only one of these will ultimately be generated and some special treatment is required. Thes are called R's *outer* and *inner* quantifiers respectively. PAS (17b) is processed first. A scoping is chosen for R and C and a dependency function is constructed in the normal way but when it comes to partitioning the function and generating a quantifier for C some care must be taken. Some choices of partition and quantifier must be excluded. What is required is that the resulting focus set for R is the set of all representatives who satisfy restriction (17b) under the chosen partition and quantifier. Consider the following dependency function and associated partition.

```
18 a  of: {r1,r2,r3} → power( {c1,c2,c3} )
          r1 → { c1 }
          r2 → { c2 }
          r3 → { c3 }

   b  focus: of: {r1,r2} → power({c1,c2})
                r1 → { c1 }
                r2 → { c2 }

      compt: of: {r3} → power({c3})
                r3 → { b3 }
```

Based on the focus in (18b) the quantifier *exactly one* might be generated for C. The corresponding candidate set for R is {r1,r2} but this is not the set of all representatives who satisfy the restriction since r3 also satisfies it. It is to avoid this anomaly that the following constraint on the acceptability of dependency function partitions in this context.

19 If variable R is in wide scope position in (17b) then $Q_{R'}$ must be of the form *exactly N* but is not generated in the final output.

Constraint (19) restricts the range of acceptable partitions by restricting the range of acceptable inner quantifiers for R. It also specifies R's outer quantifier as the one which is to be finally generated. It guarantees that R's focus set is maximal in the sense that it contains all possible R's which satisfy the restriction and avoids the above anomoly by failing to allow partition (18b). A parallel but different constraint is applied whenever R is in narrow scope position.

20 If variable R is in narrow scope position in (17b) then Q_R must be the quantifier $\forall$ but is not generated in the final output.

The algorithm

The overall strategy is to process a PAS recursively, assigning quantifiers to the most deeply nested structures first. As a variable's restriction is processed the resulting focus set is passed back up to act as the candidate set for the same variable in the embedding structure. An inner quantifier is also returned together with a flag which indicates scoping choices within the restriction. The algorithm is as follows, where the choose construct indicates non-determinism.

```
21 to process_PAS p( Qx(X,RX),Qy(Y,RY) )
     process_RES RX; process_RES RY;
     % returned by calls to process_RES:
     % Xs, Ys: the candidate sets for X&Y
     % Qx', Qy': the inner quants for X&Y
     % ScpX, ScpY: = '>' if head variable
     % is given wide scope within restriction,
     % '<' if it has narrow scope or 'nil' if there
     % are no other variables within the rest'n
     choose a scoping for X and Y;
     construct the dependency function;
     choose a partition;
     choose outer quantifiers Qx" and Qy";
     % must be consistent with consts. 19&20
     if ScpX = > or ScpX = nil then Qx := Qx"
       else Qx := Qx';
     if ScpY = > or ScpX = nil then Qy := Qy"
       else Qy := Qy';
   end process_PAS;
```

```
   to process_RES RX % X is head of phrase
   % this procedure returns: Xs, Qx & ScpX
   case 1: RX = p(Qx(X,RX),Qy(Y,RY))
   % i.e. RX contains an embedded NP
     process_RES RY; % returns Ys Qy' ScpY
     Xs := { X: RX };
     choose a scoping for X and Y;
     % and so assign a value to ScpX
     construct a dependency function;
     choose a partition;
     choose quantifiers Qx and Qy";
     % must be consistent with constraints
     % 19&20 and ScpX & ScpY
   case 2: RX = p(X)
   % i.e. RX contains no embedded structure
     Xs := { X: p(X) }; Qx := nil; ScpX := nil;
   end process_RES;
```

An example

Consider the following model:

```
rep( r1 ).    of( r1, c1 ).    saw( r1, s1 ).
rep( r2 ).    of( r2, c2 ).    saw( r1, s2 ).
rep( r3 ).                     saw( r2, s1 ).
rep( r4 ).    samp( s1 ).      saw( r2, s2 ).
              samp( s2 ).      saw( r3, s1 ).
com( c1 ).                     saw( r4, s2 ).
com( c2 ).
```

and the target PAS (16). According to process_PAS the following restrictions must be processed first.

```
22 a  of( QR(R,rep(R)), QC(C,com(C)) )
   b  samp(S)
```

Starting with (22a):

```
candidate set R = { r1, r2, r3, r4 }
candidate set C = { c1, c2 }
choose scoping R > C
```

the following dependency function is constructed.

```
23 of: {r1,r2,r3,r4} → power( {c1,c2} )
       r1 → { c1 }
       r2 → { c2 }
       r3 → { }
       r4 → { }
```

The following partition is chosen.

24 focus: of: {r1,r2} → power({c1,c2})
r1 → { c1 }
r2 → { c2 }

compt: of: {r3,r4} → power({})
r3 → { }
r4 → { }

The quantifier *a* is chosen for C and the candidate set {r1,r2} is returned.

Processing (22b) is straightforward and consists of returning the value {s1,s2}, the set of all samples. Now both restrictions have been processed and PAS (16) is processed with the candidate sets {r1,r2} and {s1,s2}.

candidate set R = { r1, r2 }
candidate set S = { s1, s2 }
choose scoping R > S

The following dependency function is constructed.

25 saw: {r1,r2} → power({s1,s2})
r1 → { s1, s2 }
r2 → { s1, s2 }

and the following partition is chosen.

26 focus: saw: {r1,r2} → power({s1,s2})
r1 → { s1, s2 }
r2 → { s1, s2 }

compt: saw: { } → power({ })

The quantifiers *every* and *both* are now chosen for R and S respectively giving the following sentence.

27 *Every representative of a company saw both samples*

Of course different scoping and partitioning choices may have generated different quantifiers. Those in (27) are based on the scoping choices R > C and R > S with the partition choice shown in (26).

Conclusion

An algorithm has been described for generating quantifiers in English sentences which describe small models containing collections of individuals which are inter-related in various ways.

The algorithm performs a great deal of search with three levels of non-determinism corresponding to.

- quantifier scoping choices
- choice of focus sets / dependency function partitions
- choice of individual quantifiers constrained by the above two choices

This is not necessarily a problem in the context of language generation where only one solution is sought.

An obvious improvement to the algorithm would be to generate a preferred sentence or to rank the outputs as to how well they describe the model. We are currently looking at how this might be done by incorporating something like the preference heuristics that have been used successfully to select quantifier scopings in natural language analysis (Grosz et al 1987). After the choose scoping step in the algorithm quantifiers can be proposed which are preferred in the given scoping position. These proposed quantifiers are then checked first by q_inc/3 and q_dec/3. The details of this have not yet been worked out.

The description given of the algorithm is based on binary predicates for the sake of brevity and clarity but the generalisation to predicates with three or more arguments is not difficult. For example, sentences of the following form can be generated:

28 *Every boy gave most girls a kiss*

where there is a different kiss for each <boy,girl> pair. The resulting dependency functions are, however, much bigger, and consequently the search space is also.

It is well documented (Webber 1978, Park 1995) that some plural noun phrases are capable of *collective* interpretations which are not sensitive to quantifier scoping. For

example, the sentence "*Three men lifted two boxes*" has an interpretation in which three men combined their efforts in a single act of lifting two boxes. The algorithm does not deal with collective interpretations like this.

References

Alshawi, H, (1990), "*Resolving Quasi Logical Forms.*", Computational Linguistics, 16, 3.

Barwise, J, R Cooper, (1981), *Generalized Quantifiers and Natural Language*, Linguistics and Philosophy, 4, 159-219.

Creaney, N, (1995), *Implementing Scope and Dependency Constrains in a Typed Attribute Logic*, Proceedings of 5th International Workshop on Natural Language Understanding and Logic Programming, Lisbon.

Grosz, BJ, DE Appelt, PA Martin, and FCN Pereira, (1987), *TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces*, Artificilal Intelligence, 32, 173-243.

Hobbs, JR, SM Shieber, (1987), *An Algorithm for Generating Quantifier Scopings*, Computational Linguistics, Vol.3, No.1, 47-63.

Park, J, (1995), *Quantifier Scope and Constituency*, Proceedings ACL-95.

Saint-Dizier, P, (1984). *Handling Quantifier Scoping Ambiguities in a Semantic Representation of Natural Language Sentences.* In Dahl, V. and Saint-Dizier, P. (eds.). Proc. 1st Int. Workshop on Natural Language Understanding and Logic Programming, (1984). North-Holland.

Webber, BL, (1978), *A Formal Approach to Discourse Anaphora*, BBN report no. 3761, Cambridge, MA: Bolt Beranek and Newman Inc.

Appendix 1: terminology diagram