Östen Dahl
Univ of Stockholm

THE INTERPRETATION OF BOUND PRONOUNS

This paper is a report on work in progress with the aim of simulating on a computer some aspects of the process of understanding sentences, more specifically the interpretation of so-called **bound pronouns.** The work connects directly to some earlier papers of mine (Dahl 1983a and 1983b), where those problems were discussed from a theoretical point of view.

A bound pronoun is, roughly speaking, a pronoun which behaves analogously to a bound variable in logic. There are at least two kinds of criteria for regarding a pronoun as bound: (i) syntactic criteria - some kinds of pronouns, such as reflexives, reciprocals and so-called logophorics, must find their antecedents in syntactically defined domains, (ii) semantic criteria - some pronouns cannot be assigned referents 'in the world' but can be understood only if regarded as referentially dependent on their antecedents: this concerns e.g. pronouns bound by quantified NPs and wh-phrases (e.g. himself in Nobody likes himself). Although the classes of pronouns delimited by these criteria are not quite identical, they overlap to such an extent that in most cases, they can be regarded as equivalent.

In my earlier papers, I have discussed some cases of bound pronouns which are troublesome for the current theories that account for bound pronouns by translating them into some kind of logical notation using bound variable or equivalent devices. Those cases include:

(i) 'sloppy identity' cases (as in John loves his wife and so does Bill, where Bill may be understood to love either his own or John's wife)

(ii) 'dislocated bound pronouns', that is pronouns that have

The interpretation of bound pronouns
Östen Dahl
Proceedings of NODALIDA 1985, pages 49-57

been 'moved' (presupposing a transformational analysis) out of the scope of their binders, e.g. Himself, everyone despises, and in particular among those

(iii) pronouns with 'relational' (Engdahl 1985) or 'second-order' readings, as in a sentence such as The only woman every Englishman admires is his mother, the interpretation of which cannot be rendered without having recourse to second-order logic

The main idea put forward in my papers was that the troublesome cases could be accounted for if the location of the antecedent in the syntactic structure were considered an integral part of a bound pronoun's interpretation.

So far, two main versions of the pronoun interpretation program have been developed. In Dahl 1985, I report an attempt to construct a syntactic parser to be used on an 8-bit microcomputer, written in the LISP dialect muLISP. The first version of the pronoun interpretation program (henceforth 'Version 1') used a somewhat more elaborate version of this parser as its base, that is, the semantic part of the program took syntactically analysed sentences as its input and assigned referents to the noun phrases in them. In addition, the program had what can be called rudimentary conversational competence: the sentences processed were compared with a database and depending on the type of sentence, the appropriate action was taken: in the case of interrogative sentences, an answer was given, in the case of a declarative sentence, the proposition was added to the database. The reference assignment process in Version 1 worked in an top-down fashion, assigning referents first to the highest NPs in the syntactic structure. Extensive use was made of temporary registers, where processed NPs (with identified referents) were stored so as to be retrieved later on when needed as antecedents of pronouns. When an antecedent was found for a pronoun, both the referent and the location (that is, where it was found in the registers) of the antecedent was stored on the property list of the pronoun. This information was then exploited by the mechanisms used in the

'troublesome cases' listed above. Version 1 was thus able to handle both sloppy identity and at least some 'relational questions', e.g. the following:

(1) Whom does every man love, his wife or Mary?

However, Version 1 was rather slow, with processing times up to half a minute for processing some sentences (this would include both syntactic parsing, reference assignment, comparison with the database and appropriate reaction). There were several reasons for that, including inherent limitations in the hardware and software used. Of a more direct linguistic relevance, however, were the following circumstances: The syntactic and semantic components of the systems were wholly autonomous from each other, and indeed worked in rather different fashions: the syntactic parser was strictly bottom-up, systematically taking into considerations all possible analyses of the sentences, whereas the semantics, as has already been pointed out, worked from the top down, and with the principle of always choosing the first possible alternative. When I considered the slowness of Version 1 and also realized that what the semantic part of it did was largely repeating the syntactic analysis of the sentence, it appeared to me that it might be fruitful to try and build a system where syntactic and semantic analysis would be done in an integrated fashion. This, however, put stronger demands on the parsing mechanism, since it required a more intelligent way of handling structural ambiguities.

Version 2, then, has been designed to meet these demands. It is written in the MS-DOS version of muLISP and has been run on several kinds of IBM compatible computers. It has not yet been developed as fully as Version 1 (the 'conversational' part has not been implemented, for instance) but its performance is significantly better than that of Version 1, partly due to better hardware but also due to a more efficient structure of the parsing mechanism. Thus, the parsing time (including reference assignment) is about 20 milliseconds per word on an IBM AT computer. Perhaps the main advantage is that this time

is more or less linear, whereas the parsing time per word in Version 1 grew very rapidly with the length of sentences.

The syntactic analysis in Version 2 is done according to the following principles:

(i) the output is a LISP structure which can be characterized as an 'almost unlabelled bracketing', that is, with very few exceptions, the syntactic category of a constituent (which has the form of a list) is not explicitly marked but has to be deduced from the lexical category of its 'head', that is the first member (CAR) of the list

(ii) parsing is done from left to right in a more or less deterministic way

(iii) the fact that the category of a constituent is in general known when you have identified its head or its first word (which is often the same thing) is systematically exploited in predicting what comes next

(iv) backtracking is made by a systematic use of local parameters of LISP functions: every time a new word is parsed a call is made to a function and the partial structure built so far is passed to that function as a parameter - if the continued parse does not succeed, one automatically returns to the previous state

(v) at any point in the parsing process, the partial analysis arrived at so far is represented as a single stack of 'active constituents' (called the ACTIVESTACK), that is, constituents that have not yet been finished. To show what the parsing of a sentence may look like, we show the successive stages of the parsing of (2) in (3).

(2) John believes that Mary loves Bill

(3)

```
(expression to be parsed:)                                    (ACTIVESTACK:)

1: John believes that Mary loves Bill                                    NIL
2: believes that Mary loves Bill                                ((John) VP (S))
3: that Mary loves Bill                          (NP (believe -s) (S (John)))
4: Mary loves Bill                    (S (that) (believe -s) (S (John)))
5: loves Bill              ((Mary) VP (S) (that)(believe -s) (S (John)))
6: Bill          (NP (love -s) (S (Mary)) (that) (believe -s) (S (John)))
7: NIL       ((Bill) (love -s) (S (Mary)) (that) (believe -s) (S (John)))

(close all constituents)

(S (John) (believe -s (that (S (Mary) (love -s (Bill))))))
```

The assignment of referents to NPs is done during the syntactic analysis, more specifically, when the noun phrase in question is 'closed', i.e. moved off the stack of active constituents. When a referent is assigned to a noun phrase, a 'dotted pair' representing the referent is added to the list which represents the constituent in the structure. At present, the system can handle three kinds of NPs: proper names, bound pronouns, and NPs with a possessive in the determiner slot. For proper nouns, the assignment process is trivial: the proper name itself is used as a reference indicator. Thus, the LISP expression to the left of the arrow is converted into the one to the right of the arrow:

(4) (John) ----> (John (REF. John))

Some people may be disturbed by this rather vacuous process: the point here is that since we are not directly concerned with how proper names are interpreted we do not want to introduce any complications here. Of course, we could easily plug in a

routine that puts in a referential index or the like.

For NPs with a possessive determiner, the principle is also slightly ad hoc: the referent of the possessive expression is first determined, then the property list of that referent is examined to see if there is some property which coincides with the head noun of the NP: in that case, the value of that property becomes the referent of the whole NP. For instance, if we have the NP John's wife and we find the item (wife.Mary) on John's property list, then the referent of John's wife is taken to be Mary.

The most interesting part of the referent assignment procedure is that which assigns antecedents and referents to bound pronouns. The assumption is that the antecedent of a bound pronoun is to be found among the NPs that c-command it. According to the current definition, a node x c-commands a node y if and only if the node that immediately dominates x also dominates y. In the present system, the c-commanders of an NP that is being 'closed' are always precisely those NPs that are immediate constituents of the members of the ACTIVESTACK. For instance, when the NP Bill in (2) above is closed, the ACTIVESTACK looks as follows:

(5) (love -s) (S (Mary)) (that) (believe -s) (S (John)))

The c-commanders in (5) are thus Mary and John.

This makes it possible to formulate a relatively simple algorithm for finding the possible antecedents. In addition to assigning a referent to a pronoun, the algorithm also stores the distance (in nodes) between the pronoun and its antecedent. The point of this will become clear later.

In muLISP formalism the main antecendent-finding function looks as follows (some irrelevant details have been left out):

(6)

```
(DEFUN ANTECEDENT (LAMBDA (X Y XNP XNODE DIST)
  (SETQ Y (CDR ACTIVESTACK))   Define Y as the ACTIVESTACK minus the NP
                                                    under consideration.
  (SETQ DIST 0)                                Set the variable DIST to 0.
  (LOOP                     Repeat until Y is empty or antecedent is
                                                                  found:
    ((NULL Y) NIL)
    (SETQ XNODE (POP Y))                  Set XNODE to next member of Y.
    (SETQ XNP (FIRSTNP XNODE)) Find the first NP in XNODE: call it XNP.
    ((AND
        (MEMBER (CAR X) REFLPROLIST)    If the pronoun is reflexive and
        (EQ (CAT XNODE) S) )                  XNODE is a sentence, then
      (PUT X 'ANTEC-DIST DIST)  set the antecedent-distance to DIST and
      (AGREE X XNP) )        the antecedent to XNP, if it agrees with
                                                    the pronoun, else to NIL,
    ((AND                              (if the pronoun is non-reflexive:)
        (AGREE X XNP)                 if XNP agrees with the pronoun then
        (NOT (AND                   unless the pronoun is non-possessive
         (NOT (POSSESSIVE X))                                        and
         (EQ (GET XNP REF) (GET (SUBJECT) REF)) )) )             XNP is
                                                     coreferent with the
                                                   subject of the sentence,
      (PUT X 'ANTEC-DIST DIST)then set the antecedent-distance to DIST
      XNP )                                        and the antecedent to XNP.
      XNP )
        (SETQ DIST (ADD1 DIST))  ) ) ))                  Add 1 to DIST.
```

This is certainly a simplified rule: for instance, it assumes that the antecedent of a reflexive is always the subject. However, in most simple cases, it assigns the closest possible antecedent to any bound pronoun.

Let us now have a closer look at the antecedent distance parameter. Its function is to define the location of the antecedent of a pronoun: this information is above all useful when the stored interpretation of the constituent which

contains the pronoun is retrieved later on. We shall illustrate what this means by looking at the way in which the program handles 'sloppy identity'. Consider the again the example from the beginning of the paper:

(7) John loves his wife and so does Bill

At present, the program is only able to handle a somewhat unidiomatic paraphrase of (7):

(8) John loves his wife and Bill too.

Basically, the following is what happens when (8) is interpreted by the system: First, the clause John loves his wife is parsed. Assuming that the system knows that Mary is John's wife, it will assign Mary as a referent to his wife. Then, the reduced clause Bill too is parsed. After the subject NP Bill the system expects a verb phrase: it takes the particle too as a signal of an elliptical VP. Every time a VP is parsed, it becomes the value of the variable LASTVP: in this case, LASTVP is loves his wife. The parsed version of this expression is now copied into the place where the VP should occur in the elliptical sentence. When this happens, the NP his wife is again subjected to the reference assignment process - however, since it is the second time, the antecedent is found not by the function ANTECEDENT but by another called FIND-ANTECEDENT-AGAIN. This function looks at the antecedent distance associated with the pronoun his and tries to find the NP at the corresponding place in the tree. In this case, it is Bill, so the referent of his wife is now taken to be Bill's wife.

Version 2 has not yet been developed so far that it can take care of the other problematic cases of bound pronouns, but in principle similar mechanisms as the one mentioned should be sufficient to solve the problems, as was demonstrated by Version 1. The point is that the antecedent distance parameter approach is inherently more powerful than the common way of displaying coreference relations, viz. by referential indices or multiple occurrences of the same variable letter, in that it

has a meaningful interpretation also out of context.

The above account has been lacking in explicitness in various ways. There are two reasons for this: the rather early stage of development of the program and the limited space available. The long-range aim of the undertaking is to provide a small yet powerful 'module' for processing natural languages sentences and texts, where the pronoun interpretation mechanism will only be a small part. Hopefully, the work on the 'module' will be possible to shed light on some questions of general theoretical interest.

REFERENCES

Dahl, Ö. 1983a. On the nature of bound pronouns. PILUS 48, Dept. of Linguistics, Univ. of Stockholm.

Dahl, Ö. 1983b. Bound pronouns in an integrated process model. In F. Karlsson, ed., Papers from the Seventh Scandinavian Conference of Linguistics. University of Helsinki, Dept. of General Linguistics.

Dahl, Ö. 1985. Syntactic Parsing on a Microcomputer. In S. Bäckman and G. Kjellmer, eds., Papers on Language and Literature presented to Alvar Ellegård and Erik Frykman. Gothenburg Studies in English 60. Göteborg: Acta Universitatis Gothoburgensis.

Engdahl, E. 1985. The Syntax and Semantics of Questions with Special Reference to Swedish. Dordrecht: Reidel.