A SYNTACTIC FILTER ON PRONOMINAL ANAPHORA FOR SLOT GRAMMAR

Shalom Lappin and Michael McCord

IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
E-mail: Lappin/McCord@yktvmh.bitnet

ABSTRACT

We propose a syntactic filter for identifying non-coreferential pronoun-NP pairs within a sentence. The filter applies to the output of a Slot Grammar parser and is formulated in terms of the head-argument structures which the parser generates. It handles control and unbounded dependency constructions without empty categories or binding chains, by virtue of the unificational nature of the parser. The filter provides constraints for a discourse semantics system, reducing the search domain to which the inference rules of the system's anaphora resolution component apply.

1. INTRODUCTION

In this paper we present an implemented algorithm which filters intra-sentential relations of referential dependence between pronouns and putative NP antecedents (both full and pronominal NP's) for the syntactic representations provided by an English Slot Grammar parser (McCord 1989b). For each parse of a sentence, the algorithm provides a list of pronoun-NP pairs where referential dependence of the first element on the second is excluded by syntactic constraints. The coverage of the filter has roughly the same extension as conditions B and C of Chomsky's (1981, 1986) binding theory. However, the formulation of the algorithm is significantly different from the conditions of the binding theory, and from proposed implementations of its conditions. In particular, the filter formulates constraints on pronominal anaphora in terms of the head-argument structures provided by Slot Grammar syntactic representations rather than the configurational tree relations, particularly c-command, on which the binding theory relies. As a result, the statements of the algorithm apply straightforwardly, and without special provision, to a wide variety of constructions which recently proposed implementations of the binding theory do not handle without additional devices. Like the Slot Grammar whose input it applies to, the algorithm runs in Prolog, and it is stated in essentially declarative terms.

In Section 2 we give a brief description of Slot Grammar, and the parser we are employing. The syntactic filter is presented in Section 3, first through a statement of six constraints, each of which is sufficient to rule out coreference, then through a detailed description of the algorithm which implements these constraints. We illustrate the algorithm with examples of the lists of non-coreferential pairs which it provides for particular parses. In Section 4 we compare our approach to other proposals for syntactic filtering of pronominal anaphora which have appeared in the literature. We discuss Hobbs' algorithm, and we take up two recent implementations of the binding theory. Finally, in Section 5 we discuss the integration of our filter into other systems of anaphora resolution. We indicate how it can be combined with a VP anaphora algorithm which we have recently completed. We also outline the incorporation of our algorithm into **LODUS** (Bernth 1989), a system for discourse representation.

2. SLOT GRAMMAR

The original work on Slot Grammar was done around 1976-78 and appeared in (McCord 1980). Recently, a new version (McCord 1989b) was developed in a logic programming framework, in connection with the machine translation system **LMT** (McCord 1989a,c,d).

Slot Grammar is lexicalist and is dependency-oriented. Every phrase has a head word (with a given word sense and morphosyntactic features). The constituents of a phrase besides the head word (also called the *modifiers* of the head) are obtained by "filling" *slots* associated with the head. Slots are symbols like `subj`, `obj` and `iobj` representing grammatical relations, and are associated with a word (sense) in two ways. The lexical entry for the word specifies a set of *complement* slots (corresponding to arguments of the word sense in logical form); and the grammar specifies a set of *adjunct* slots for each part of

speech. A complement slot can be filled at most once, and an adjunct slot can by default be filled any number of times.

The phenomena treated by augmented phrase structure rules in some grammatical systems are treated modularly by several different types of rules in Slot Grammar. The most important type of rule is the *(slot) filler* rule, which gives conditions (expressed largely through unification) on the filler phrase and its relations to the higher phrase.

Filler rules are stated (normally) without reference to conditions on *order* among constituents. But there are separately stated *ordering rules*.[1] *Slot/head* ordering rules state conditions on the position (left or right) of the slot (filler) relative to the head word. *Slot/slot* ordering rules place conditions on the relative left-to-right order of (the fillers of) two slots.

A slot is *obligatory* (not *optional*) if it must be filled, either in the current phrase or in a raised position through left movement or coordination. Adjunct slots are always optional. Complement slots are optional by default, but they may be specified to be obligatory in a particular lexical entry, or they may be so specified in the grammar by *obligatory slot rules*. Such rules may be unconditional or be conditional on the characteristics of the higher phrase. They also may specify that a slot is obligatory *relative* to the filling of another slot. For example, the direct object slot in English may be declared obligatory on the condition that the indirect object slot is filled by a noun phrase.

One aim of Slot Grammar is to develop a powerful language-independent module, a "shell", which can be used together with language-dependent modules, reducing the effort of writing grammars for new languages. The Slot Grammar shell module includes the parser, which is a bottom-up chart parser. It also includes most of the treatment of coordination, unbounded dependencies, controlled subjects, and punctuation. And the shell contains a system for evaluating parses, extending Heidorn's (1982) parse metric, which is used not only for ranking final parses but also for pruning away unlikely partial analyses *during* parsing, thus reducing the problem of parse space explosion. Parse evaluation expresses preferences for close attachment, for choice of complements over adjuncts, and for parallelism in coordination.

Although the shell contains *most* of the treatment of the above phenomena (coordination, etc.), a small part of their treatment is necessarily language-dependent. A (language-specific) grammar can include for instance (1) rules for coordinating feature structures that override the defaults in the shell; (2) declarations of slots (called *extraposer* slots) that allow left extraposition of other slots out of their fillers; (3) language-specific rules for punctuation that override defaults; and (4) language-specific controls over parse evaluation that override defaults.

Currently, Slot Grammars are being developed for English (**ESG**) by McCord, for Danish (**DSG**) by Arendse Bernth, and for German (**GSG**) by Ulrike Schwall. **ESG** uses the **UDICT** lexicon (Byrd 1983, Klavans and Wacholder 1989) having over 60,000 lemmas, with an interface that produces slot frames. The filter algorithm has so far been successfully tested with **ESG** and **GSG**. (The adaptation to German was done by Ulrike Schwall.)

The algorithm applies in a second pass to the parse output, so the important thing in the remainder of this section is to describe Slot Grammar syntactic analysis structures.

A syntactic structure is a tree; each node of the tree represents a phrase in the sentence and has a unique head word. Formally, a phrase is represented by a term

```
phrase(X,H,Sense,Features,
       SlotFrame,Ext,Mods),
```

where the components are as follows: (1) `X` is a logical variable called the *marker* of the phrase. Unifications of the marker play a crucial role in the filter algorithm. (2) `H` is an integer representing the position of the head word of the phrase. This integer identifies the phrase uniquely, and is used in the filter algorithm as the way of referring to phrases. (3) `Sense` is the word sense of the head word. (4) `Features` is the feature structure of the head word *and* of the phrase. It is a logic term (not an attribute-value list), which is generally rather sparse in information, showing mainly the part of speech and inflectional features of the head word. (5) `SlotFrame` is the list of complement slots, each slot being in the internal form `slot(Slot,Ob,X)`, where `Slot` is the slot name, `Ob` shows whether it is an obligatory form of `Slot`, and `X` is the *slot marker*. The slot marker is unified (essentially) with the marker of the filler phrase when the slot is filled, even remotely, as in left movement or coordination. Such unifications are important for the filter algorithm. (6) `Ext` is the list of slots that have been *extraposed* or *raised* to the level of the current phrase. (7) The last component `Mods` represents the modifiers (daughters) of the phrase, and is of the form `mods(LMods,RMods)` where `LMods` and `RMods` are

[1] The distinction between slot filler rules and ordering constraints parallels the difference between Immediate Dominance Rules and Linear Precedence Rules in GPSG. See Gazdar et al (1985) for a characterization of ID and LP rules in GPSG. See (McCord 1989b) for more discussion of the relation of Slot Grammar to other systems.

```
Who did John say wanted to try to find him?

┌──────── subj(n)              who(X2)                noun
■──────── top                  do1(X1,X3,X4)          verb
└──────── subj(n)              John(X3)               noun
└──┬──── auxcmp(inf(bare))    say(X4,X3,X9,u)        verb
   └──┬──── obj(fin)          want(X9,X2,X2,X12)     verb
      ┌──── preinf            preinf(X12)            preinf
      └──┬──── comp(en|inf|ing) try(X12,X2,X13)      verb
         ┌──── preinf         preinf(X13)            preinf
         └──┬──── obj(inf)    find(X13,X2,X14,u,u)   verb
            └──── obj(fin)    he(X14)                noun
```

Figure 1.

the lists of left modifiers and right modifiers, respectively. Each member of a modifier list is of the form `Slot:Phrase` where `Slot` is a slot and `Phrase` is a `phrase` which fills `Slot`. Modifier lists reflect surface order, and a given slot may appear more than once (if it is an adjunct). Thus modifier lists are not attribute-value lists.

In Figure 1, a sample parse tree is shown, displayed by a procedure that uses only one line per node and exhibits tree structure lines on the left. In this display, each line (representing a node) shows (1) the tree connection lines, (2) the slot filled by the node, (3) the *word sense predication*, and (4) the feature structure. The feature structure is abbreviated here by a display option, showing only the part of speech. The word sense predication consists of the sense name of the head word with the following arguments. The first argument is the marker variable for the phrase (node) itself; it is like an event or state variable for verbs. The remaining arguments are the marker variables of the slots in the complement slot frame (`u` signifies "unbound"). As can be seen in the display, the complement arguments are unified with the marker variables of the filler complement phrases. Note that in the example the marker `X2` of the 'who' phrase is unified with the subject variables of 'want', 'try', and 'find'. (There are also some unifications created by adjunct slot filling, which will not be described here.)

For the operation of the filter algorithm, there is a preliminary step in which pertinent information about the parse tree is represented in a manner more convenient for the algorithm. As indicated above, nodes (phrases) themselves are represented by the word numbers of their head words. Properties of phrases and relations between them are represented by unit clauses (predications) involving these integers (and other data), which are asserted into the Prolog workspace. Because of this "dispersed" representation with a collection of unit clauses, the original `phrase` structure for the whole tree is first grounded (variables are bound to unique constants) before the unit clauses are created.

As an example for this clausal representation, the clause `hasarg(P,X)` says that phrase `P` has `X` one of its arguments; *i.e.*, `X` is the slot marker variable for one of the complement slots of `P`. For the above sample parse, then, we would get clauses

```
hasarg(5,'X2').   hasarg(5,'X12').
```

as information about the 'want' node (5).

As another example, the clause `phmarker(P,X)` is added when phrase `P` has marker `X`. Thus for the above sample, we would get the unit clause

```
phmarker(1,'X2').
```

An important predicate for the filter algorithm is `argm`, defined by

```
argm(P,Q) ← phmarker(P,X) &
            hasarg(Q,X).
```

This says that phrase `P` is an argument of phrase `Q`. This includes remote arguments and controlled subjects, because of the unifications of marker variables performed by the Slot Grammar parser. Thus for the above parse, we would get

```
argm(1,5).  argm(1,7).  argm(1,9).
```

showing that 'who' is an argument of 'want', 'try', and 'find'.

3. THE FILTER

The Filter Algorithm

```
A.      nonrefdep(P,Q) ← refpair(P,Q) & ncorefpair(P,Q).
A.1.    refpair(P,Q) ← pron(P) & noun(Q) & P=/Q.
B.      ncorefpair(P,Q) ← nonagr(P,Q) &/.
B.1.    nonagr(P,Q) ← numdif(P,Q) | typedif(P,Q) | persdif(P,Q).
C.      ncorefpair(P,Q) ← proncom(P,Q) &/.
C.1.    proncom(P,Q) ←
a.         argm(P,H) &
b.         (argm(Q,H) &/ |
c.            ¬pron(Q) &
d.            cont(Q,H) &
e.            (¬subclcont(Q,T) | gt(Q,P)) &
f.            (¬det(Q) | gt(Q,P))).
C.2.    cont_i(P,Q) ← argm(P,Q) | adjunct(P,Q).
C.2.1.  cont(P,Q) ← cont_i(P,Q).
C.2.2.  cont(P,Q) ← cont_i(P,R) & R=/Q & cont(R,Q).
C.3.    subclcont(P,Q) ← subconj(Q) & cont(P,Q).
D.      ncorefpair(P,Q) ← prepcom(Q,P) &/.
D.1.    prepcom(Q,P) ← argm(Q,H) & adjunct(R,H) & prep(R) & argm(P,R).
E.      ncorefpair(P,Q) ← npcom(P,Q) &/.
E.1.    npcom(Q,P) ← adjunct(Q,H) & noun(H) &
                     (argm(P,H) |
                       adjunct(R,H) & prep(R) & argm(P,R)).
F.      ncorefpair(P,Q) ← nppcom(P,Q) &/.
F.1     nppcom(P,Q) ← adjunct(P,H) & noun(H) &
                          ¬pron(Q) & cont(Q,H).
```

Figure 2.

In preparation for stating the six constraints, we adopt the following definitions. The *agreement features* of an NP are its number, person and gender features. We will say that a phrase P is in the *argument domain* of a phrase N iff P an N are both arguments of the same head. We will also say that P is in the *adjunct domain* of N iff N is an argument of a head H, P is the object of a preposition PREP, and PREP is an adjunct of H. P is in the *NP domain* of N iff N is the determiner of a noun Q and (i) P is an argument of Q, or (ii) P is the object of a preposition PREP and Prep is an adjunct of Q. The six constraints are as follows. A pronoun P is not coreferential with a noun phrase N if any of the following conditions holds.

I. P and N have incompatible agreement features.
II. P is in the argument domain of N.
III. P is in the adjunct domain of N.
IV. P is an argument of a head H, N is not a pronoun, and N is contained in H.
V. P is in the NP domain of N.
VI. P is the determiner of a noun Q, and N is contained in Q.

The algorithm which implements I-VI defines a predicate `nonrefdep(P,Q)` which is satisfied by a pair whose first element is a pronoun and whose second element is an NP on which the pronoun cannot be taken as referentially dependent, by virtue of the syntactic relation between them. The main clauses of the algorithm are shown in Figure 2.

Rule A specifies that the main goal `nonrefdep(P,Q)` is satisfied by `<P,Q>` if this pair is a referential pair (`refpair(P,Q)`) and a non-coreferential pair (`ncorefpair(P,Q)`). A.1 defines a refpair `<P,Q>` as one in which `P` is a pronoun, `Q` is a noun (either pronominal or non-pronominal), and `P` and `Q` are distinct. Rules B, C, D, E, and F provide a disjunctive statement of the conditions under which the non-coreference goal `ncorefpair(P,Q)` is satisfied, and so constitute the core of the algorithm. Each of these rules concludes with a cut to prevent unnecessary backtracking which could generate looping.

Rule B, together with B.1, identifies the conditions under which constraint I holds. In the following example sentences, the pairs consisting of the second and the first coindexed expressions in 1a-c (and in 1c also the pair <'I','she'>) satisfy `nonrefdep(P,Q)` by virtue of rule B.

1a. John$_i$ said that they$_i$ came.

b. The woman$_i$ said that he$_i$ is funny.
c. I$_i$ believe that she$_i$ is competent.

The algorithm identifies <'they','John'> as a nonrefdep pair in 1a, which entails that 'they' cannot be taken as coreferential with 'John'. However, (the referent of) 'John' could of course be part of the reference set of 'they', and in suitable discourses **LODUS** could identify this possibility.

Rule C states that <P,Q> is a non-coreferential pair if it satisfies the proncom(P,Q) predicate. This holds under two conditions, corresponding to disjuncts C.1.a-b and C.1.a,c-f. The first condition specifies that the pronoun P and its putative antecedent Q are both arguments of the same phrasal head, and so implements constraint II. This rules out referential dependence in 2a-b.

2a. Mary$_i$ likes her$_i$.
b. She$_i$ likes her$_i$.

Given the fact that Slot Grammar unifies the argument and adjunct variables of a head with the phrases which fill these variable positions, it will also exclude coreference in cases of control and unbounded dependency, as in 3a-c.

3a. John$_i$ seems to want to see him$_i$.
b. Which man$_i$ did he$_i$ see?
c. This is the girl$_i$ John said she$_i$ saw.

The second disjunct C.1.a,c-f covers cases in which the pronoun is an argument which is higher up in the head-argument structure of the sentence than a non-pronominal noun. This disjunct corresponds to condition IV. C.2-C.2.2 provide a recursive definition of containment within a phrase. This definition uses the relation of immediate containment, cont_i(P,Q), as the base of the recursion, where cont_i(P,Q) holds if Q is either an argument or an adjunct (modifier or determiner) of a head Q. The second disjunct blocks coreference in 4a-c.

4a. He$_i$ believes that the man$_i$ is amusing.
b. Who$_i$ did he$_i$ say John$_i$ kissed?
c. This is the man$_i$ he$_i$ said John$_i$ wrote about.

The wh-phrase in 4b and the head noun of the relative clause in 4c unify with variables in positions contained within the phrase (more precisely, the verb which heads the phrase) of which the pronoun is an argument. Therefore, the algorithm identifies these nouns as impossible antecedents of the pronoun.

The two final conditions of the second disjunct, C.1.e and C.1.f, describe cases in which the antecedent of a pronoun is contained in a preceding adjunct clause, and cases in which the antecedent is the determiner of an NP which precedes a pronoun, respectively. These clauses prevent such structures from satisfying the non-coreference goal, and so permit referential dependence in 5a-b.

5a. After John$_i$ sang, he$_i$ danced.
b. John$_i$'s mother likes him$_i$.

Notice that because a determiner is an adjunct of an NP and not an argument of the verb of which the NP is an argument, rule C.1 also permits coreference in 6.

6. His$_i$ mother likes John$_i$.

However, C.1.a,c-e correctly excludes referential dependence in 7, where the pronoun is an argument which is higher than a noun adjunct.

7. He$_i$ likes John$_i$'s mother.

The algorithm permits backwards anaphora in cases like 8, where the pronoun is not an argument of a phrase H to which its antecedent Q bears the cont(Q,H) relation.

8. After he$_i$ sang, John $_i$ danced.

D-D.1 block coreference between an NP which is the argument of a head H, and a pronoun that is the object of a preposition heading a PP adjunct of H, as in 9a-c. These rules implement constraint III.

9a. Sam$_i$ spoke about him$_i$.
b. She$_i$ sat near her$_i$.
c. Who$_i$ did he$_i$ ask for?

Finally, E-E.1 and F realize conditions V and VI, respectively, in NP internal non-coreference cases like 10a-c.

10a. His$_i$ portrait of John$_i$ is interesting.
b. John$_i$'s portrait of him$_i$ is interesting.
c. His$_i$ description of the portrait by John$_i$ is interesting.

Let us look at three examples of actual lists of pairs satisfying the nonrefdep predicate which the algorithm generates for particular parse trees of Slot Grammar. The items in each pair are identified by their words and word numbers, corresponding to their sequential position in the string.

When the sentence `Who did John say wanted to try to find him?` is given to the system, the parse is as shown in Figure 1 above, and the output of the filter is:

```
Noncoref pairs:
he.10 - who.1
```

```
Coreference analysis time = 11 msec.
```

Thus < 'him','who' > is identified as a non-coreferential pair, while coreference between 'John' and 'him' is allowed.

In Figure 3, the algorithm correctly lists < 'him','Bill' > (6-3) as a non-coreferential pair, while permitting 'him' to take 'John' as an antecedent. In Figure 4, it correctly excludes coreference between 'him' and 'John' (he.6-John.1), and allows 'him' to be referentially dependent upon 'Bill'.

```
John expected Bill to impress him.

   subj(n)    John(X3)              noun
   top        expect(X1,X3,X4,X5)   verb
   obj        Bill(X4)              noun
   preinf     preinf(X5)            preinf
   comp(inf)  impress(X5,X4,X6)     verb
   obj        he(X6)                noun

Noncoref pairs:
he.6 - Bill.3
Coreference analysis time = 5 msec.
```

Figure 3.

```
John lectured Bill to impress him.

   subj(n)  John(X3)              noun
   top      lecture(X1,X3,X4)     verb
   obj      Bill(X4)              noun
   preinf   preinf(X5)            preinf
   vnfvp    impress(X5,X3,X6)     verb
   obj      he(X6)                noun

Noncoref pairs:
he.6 - John.1
Coreference analysis time = 5 msec.
```

Figure 4.

It makes this distinction by virtue of the differences between the roles of the two infinitival clauses in these sentences. In Figure 3, the infinitival clause is a complement of 'expected', and this verb is marked for object control of the complement clause subject. However, in Figure 4, the infinitival clause is an adjunct of 'lectured' and requires matrix subject control.

4. EXISTING PROPOSALS FOR CONSTRAINING PRONOMINAL ANAPHORA

We will discuss three suggestions which have been made in the computational literature for syntactically constraining the relationship between a pronoun and its set of possible antecedents intra-sententially. The first is Hobbs' (1978) Algorithm, which performs a breadth-first, left-to-right search of the tree containing the pronoun for possible antecedents. The search is restricted to paths above the first NP or S node containing the pronoun, and so the pronoun cannot be bound by an antecedent in its minimal governing category. If no antecedents are found within the same tree as the pronoun, the trees of the previous sentences in the text are searched in order of proximity. There are two main difficulties with this approach. First, it cannot be applied to cases of control in infinitival clauses, like those given in Figures 3 and 4, or to unbounded dependencies, like those in Figure 1 and in examples 3b-c and 4b-c, without significant modification.

Second, the algorithm is inefficient in design and violates modularity by virtue of the fact that it computes both intra-sentential constraints on pronominal anaphora and inter-sentential antecedent possibilities each time it is invoked for a new pronoun in a tree. Our system computes the set of pronoun-NP pairs for which coreference is syntactically excluded in a single pass on a parse tree. This set provides the input to a semantic-pragmatic discourse module which determines anaphora by inference and preference rules.

The other two proposals are presented in Correa (1988), and in Ingria and Stallard (1989). Both of these models are implementations of Chomsky's Binding theory which make use of Government Binding type parsers. They employ essentially the same strategy. This involves computing the set of possible antecedents of an anaphor as the NP's which c-command the anaphor within a minimal domain (its minimal governing category).[2] The minimal domain of an NP is characterized as the first S, or the first NP without a possessive subject, in which it is contained. The possible intra-sentential antecedents of a pronoun are the set of NP's in the tree which are not included within this minimal domain.

[2] See Reinhart (1976) and (1983) for alternative definitions of c-command, and discussions of the role of this relation in determining the possibilities of anaphora. See Lappin (1985) for additional discussion of the connection between c-command and distinct varieties of pronominal anaphora. See Chomsky (1981), (1986a) and (1986b) for alternative definitions of the notion 'government' and 'minimal governing category'.

This approach does sustain modularity by computing the set of possible antecedents for all pronouns within a tree in a single pass operation, prior to the application of inter-sentential search procedures. The main difficulty with the model is that because constraints on pronominal anaphora are stated entirely in terms of configurational relations of tree geometry, specifically, in terms of c-command and minimal dominating S and NP domains, control and unbounded dependency structures can only be handled by additional and fairly complex devices. It is necessary to generate empty categories for PRO and trace in appropriate positions in parse trees. Additional algorithms must be invoked to specify the chains of control (A-binding) for PRO, and operator (A')-binding for trace in order to link these categories to the constituents which bind them. The algorithm which computes possible antecedents for anaphors and pronouns must be formulated so that it identifies the head of such a chain as non-coreferential with a pronoun or anaphor (in the sense of the Binding theory), if any element of the chain is excluded as a possible antecedent.

Neither empty categories nor binding chains are required in our system. In Slot Grammar parse representations, wh-phrases, heads of relative clauses, and NP's which control the subjects of infinitival clauses are unified with the variables corresponding to the roles they bind in argument positions. Therefore, the clauses of the algorithm apply to these constructions directly, and without additional devices or stipulations.[3]

5. THE INTEGRATION OF THE FILTER INTO OTHER SYSTEMS OF ANAPHORA RESOLUTION

We have recently implemented an algorithm for the interpretation of intrasentential VP anaphora structures like those in 11a-c.

11a. John arrived, and Mary did too.
b. Bill read every book which Sam said he did.
c. Max wrote a letter to Bill before Mary did to John.

The VP anaphora algorithm generates a second tree which copies the antecedent verb into the position of the head of the elliptical VP. It also lists the new arguments and adjuncts which the copied verb inherits from its antecedent. We have integrated our filter on pronominal anaphora into this algorithm, so that the filter applies to the interpreted trees which the algorithm generates. consider

12. John likes to him, and Bill does too.

If the filter applies to the parse of 11, it will identify only <'him','John'> as a non-coreferential pair, given that the pair <'him','Bill'> doesn't satisfy any of the conditions of the filter algorithm. However, when the filter is applied to the interpreted VP anaphora tree of 12, the filter algorithm correctly identifies both pronoun-NP pairs, as shown in the VP output of the algorithm for 12 given in Figure 5.

```
John likes him, and Bill does too.

Antecedent Verb-Elliptical Verb Pairs.
like.2 - do1.7

Elliptical Verb-New Argument Pairs.
like.7 - he.3

Interpreted VP anaphora tree.
   ┌─ subj  John(X9)            noun
┌──┼─ lconj like(X8,X9,X10)     verb
│  └─ obj   he(X10)             noun
•──── top   and(X1,X8,X11)      verb
│  ┌─ subj  Bill(X12)           noun
└──┼─ rconj like(X11,X12,X10)   verb
   └─ vadv  too(X11)            adv

Non-Coreferential Pronoun-NP Pairs.
he.3 - John.1, he.3 - Bill.6
Coreference analysis time = 70 msec.
```

Figure 5.

Our filter also provides input to a discourse understanding system, **LODUS**, designed and implemented by A. Bernth, and described in (Bernth 1988, 1989). **LODUS** creates a single discourse structure from the analyses of the Slot Grammar parser for several sentences. It interprets each sentence analysis in the context consisting of the discourse processed so far, together with domain knowledge, and it then embeds it into the discourse structure. The process of interpretation consists in applying rules of inference which encode semantic and pragmatic (know-

[3] In fact, a more complicated algorithm with approximately the same coverage as our filter can be formulated for a parser which produces configurational surface trees without empty categories and binding chains, if the parser provides deep grammatical roles at some level of representation. The first author has implemented such an algorithm for the PEG parser. For a general description of PEG, see Jensen (1986). The current version of PEG provides information on deep grammatical roles by means of second pass rules which apply to the initial parse record structure. The algorithm employs both c-command and reference to deep grammatical roles.

ledge-based) relations among lexical items, and discourse structures. The filter reduces the set of possible antecedents which the anaphora resolution component of **LODUS** considers for pronouns. For example, this component will not consider 'the cat' or 'that' as a possible antecedents for either occurrence of 'it' in the second sentence in 13, but only 'the mouse' in the first sentence of this discourse. This is due to the fact that our filter lists the excluded pairs together with the parse tree of the second sentence.

13. The mouse ran in.
The cat that saw it ate it.

Thus, the filter significantly reduces the search space which the anaphora resolution component of **LODUS** must process. The interface between our filter and **LODUS** embodies the sort of modular interaction of syntactic and semantic-pragmatic components which we see as important to the successful operation and efficiency of any anaphora resolution system.

ACKNOWLEDGMENTS

We are grateful to Arendse Bernth, Martin Chodorow, and Wlodek Zadrozny for helpful comments and advice on proposals contained in this paper.

REFERENCES

Bernth, A. (1988) *Computational Discourse Semantics*, Doctoral Dissertation, U. Copenhagen and IBM Research.

Bernth, A. (1989) "Discourse Understanding In Logic", *Proc. North American Conference on Logic Programming*, pp. 755-771, MIT Press.

Byrd, R. J. (1983) "Word Formation in Natural Language Processing Systems," *Proceedings of IJCAI-VIII*, pp. 704-706.

Chomsky, N. (1981) *Lectures on Government and Binding*, Foris, Dordrecht.

Chomsky, N. (1986a) *Knowledge of Language: Its Nature, Origin, and Use*, Praeger, New York.

Chomsky, N. (1986b) *Barriers*, MIT Press, Cambridge, Mass.

Correa, N. (1988) "A Binding Rule for Government-Binding Parsing", *COLING '88*, Budapest, pp. 123-129.

Gazdar, G., E. Klein, G. Pullum, and I. Sag, (1985) *Generalized Phrase Structure Grammar*, Blackwell, Oxford.

Heidorn, G. E. (1982) "Experience with an Easily Computed Metric for Ranking Alternative Parses," *Proceedings of Annual ACL Meeting, 1982*, pp. 82-84.

Hobbs, J. (1978) "Resolving Pronoun References", *Lingua* 44, pp. 311-338.

Ingria, R. and D. Stallard (1989) "A Computational Mechanism for Pronominal Reference", *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, pp. 262-271.

Jensen, K. (1986) "PEG: A Broad-Coverage Computational Syntax of English," Technical Report, IBM T.J. Watson Research Center, Yorktown Heights, NY.

Klavans, J. L. and Wacholder, N. (1989) "Documentation of Features and Attributes in UDICT," Research Report RC14251, IBM T.J. Watson Research Center, Yorktown Heights, N.Y.

Lappin, S. (1985) "Pronominal Binding and Coreference", *Theoretical Linguistics* 12, pp. 241-263.

McCord, M. C. (1980) "Slot Grammars," *Computational Linguistics*, vol. 6, pp. 31-43.

McCord, M. C. (1989a) "Design of **LMT**: A Prolog-based Machine Translation System," *Computational Linguistics*, vol. 15, pp. 33-52.

McCord, M. C. (1989b) "A New Version of Slot Grammar," Research Report RC 14506, IBM Research Division, Yorktown Heights, NY 10598.

McCord, M. C. (1989c) "A New Version of the Machine Translation System **LMT**," to appear in *Proc. International Scientific Symposium on Natural Language and Logic*, Springer Lecture Notes in Computer Science, and in *J. Literary and Linguistic Computing*.

McCord, M. C. (1989d) "**LMT**," *Proceedings of MT Summit II*, pp. 94-99, Deutsche Gesellschaft für Dokumentation, Frankfurt.

Reinhart, T. (1976) *The Syntactic Domain of Anaphora*, Doctoral Dissertation, MIT, Cambridge, Mass.

Reinhart, T. (1983) *Anaphora*, Croom Helm, London.