An Architecture for Anaphora Resolution

Elaine Rich
Susann LuperFoy

MCC, Advanced Computer Architecture Program
3500 West Balcones Center Drive
Austin, TX 78759

ABSTRACT

In this paper, we describe the pronominal anaphora resolution module of Lucy, a portable English understanding system. The design of this module was motivated by the observation that, although there exist many theories of anaphora resolution, no one of these theories is complete. Thus we have implemented a blackboard-like architecture in which individual partial theories can be encoded as separate modules that can interact to propose candidate antecedents and to evaluate each other's proposals.

INTRODUCTION

The Lucy system (Rich, 1987) is a prototype of a portable English front end for knowledge-based systems. The major components of Lucy are a syntax-based parser (Wittenburg, 1986), a semantic translation system, a pronominal anaphora resolution system (which will be described in this paper) and a pragmatic processor. The parser produces as its output a feature graph that describes the syntactic properties of the constituents of the sentence. The semantic translation system produces as its output a list of discourse referents and a set of assertions about them. The job of the anaphora resolution system is to augment this assertion set with additional assertions that describe coreference relations between discourse referents. Figure 1 shows the results of semantic processing and anaphora resolution for the simple discourse, "Dave created a file. He printed it."

A DISTRIBUTED ARCHITECTURE

Designing an anaphora resolution system is difficult because there exists no single, coherent theory upon which to build, even if we restrict our attention to pronominal anaphora (which we will do throughout this paper). There do, however, exist many partial theories,[1] each of which accounts for a subset of the phenomena that influence the use and interpretation of pronominal anaphora. These partial theories range from purely syntactic ones (for example the simple rules of number and gender agreement) to highly semantic and pragmatic ones that account for focusing phenomena. If there were a single, complete theory, then it might be appropriate to implement it. If there were no theories at all, then an *ad hoc* implementation might be the only alternative. But because there are partial theories but not a complete one, we have designed an architecture (patterned after the idea of a blackboard system (Erman, 1981)) that allows for a loosely coupled set of modules, each of which handles a subset of discourse phenomena by implementing a specific partial theory. These modules communicate by proposing candidate antecedents and by evaluating each other's proposals. An oversight module, called the handler, mediates these communications and resolves conflicts among the modules.

All the modules in this system share a common representation of the current discourse. This representation is called the core discourse structure. It includes a list of the discourse referents that have so far been introduced. Associated with each such referent is a set of assertions, including syntactic facts about the use of the referent (e.g., its number and gender and whether or not it is reflexive), semantic facts (such as the ones shown in Figure 1), and anaphoric usage facts such as coextension relations.

A schematic view of the architecture is shown in Figure 2. Each of the ovals in the figure represents an implementation of one of the partial theories of anaphora. Each of these implementations is called a constraint source (CS), because each of them is viewed as imposing a set of constraints on the choice of an antecedent for a pronominal referent.

e1: (create e1) (agent e1 x1) (object e1 x2)	e2: (print e2) (agent e2 x3) (object e2 x4)
x1: (= x1 Dave)	x3:
x2: (file x2)	x4:

(a) The Result of Semantic Processing

x3: (coextensive x3 x1)	x4: (coextensive x4 x2)

(b) Assertions Added by Anaphora Resolution

Figure 1: Processing "Dave created a file. He printed it."

[1] All existing theories are partial in the added sense of being fallible. That is, even when restricted to a narrow subdomain of the facts of anaphoric behavior, no account fully explains coreference possibilities that arise in context.

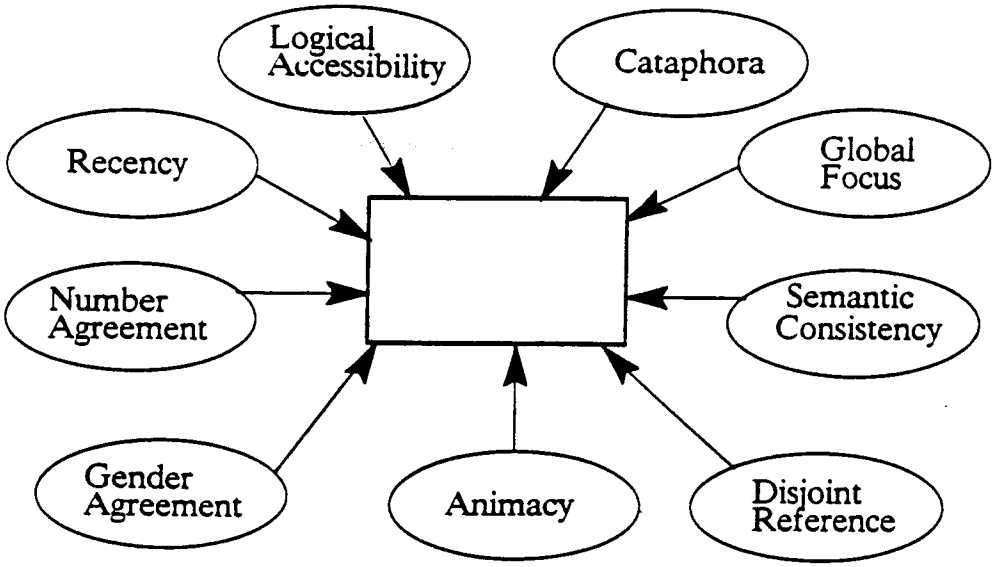

Figure 2: The Architecture of Anaphora Resolution

THE STRUCTURE OF A CONSTRAINT SOURCE

Each constraint source in this system is composed of a set of four functions (although any of these functions may be a no-op in any particular CS). These component functions are called at different times during the process of anaphor resolution, but they form a logical unit since they share knowledge structures that correspond to the particular partial theory that is implemented by the CS to which they belong. The four functions are the following:

- Modeller - This function maintains the CS's local model of the ongoing discourse. Not all CS's need such a model. Simple syntactic ones like number agreement typically do not build a local model since all the information they need is available in the core discourse structure and in the anaphoric referent being resolved. But CS's that describe more global phenomena, such as rhetorical structure, may need access to information that is neither local to a particular referent nor contained in the core discourse structure. If a local model is built, it is built to rest on top of the shared core discourse structure. The content of the local model is accessible to the other functions of the same CS but not to anyone else.
- Constraint poster - This function posts constraints that describe interactions among anaphora within a sentence. These constraints are then treated in exactly the same way as are semantic assertions about the referents. The algorithm for resolving anaphora (which will be described below) applies to a single anaphoric referent at a time. But, in some sentences, there are interactions among referents that make it impossible to treat them completely separately. By posting interaction constraints before processing individual anaphoric referents, we maintain the ability to treat the referents separately but also to guarantee that the complete interpretation for the sentence in which they occur will be consistent. As an example of this phenomenon, consider the sentence, "He saw him." Disjoint reference could post a constraint that *he* and *him* cannot co-refer before any attempt is made to find the antecedent for either of them. Most simple constraint sources do nothing when their constraint posters are called.
- Proposer - This function takes as input an anaphoric referent and returns as its output a list of candidate antecedents, each with an associated score (which will be described later). For example, recency proposes all referents in the two most recent previous sentences, as well as any in the current sentence that occur to the left of the referent that is being resolved. Some CS's, such as number agreement, never propose candidates.
- Evaluator - This function takes as input an anaphor and a candidate antecedent for that anaphor. The evaluator returns a score indicating strength of support for that candidate as antecedent to the anaphor. The returned score is based solely on the information available to the CS that is doing the evaluation. It is left to the handler, which invokes the various CS's, to combine scores from different evaluators and to resolve conflicts. Although every CS must be able to respond whenever its evaluator is called, there is a score that will be interpreted to mean, "I have no opinion." (See below.)

THE ANAPHORA RESOLUTION PROCEDURE

In the current implementation of Lucy, anaphora resolution occurs once semantic inter-

pretation is complete.[2] The procedure resolve-anaphors does the following for each sentence that is to be understood:

1. Update the core discourse structure with the results of syntactic and semantic processing of the current sentence.
2. For each CS in the system, invoke the modeller and the constraint poster.
3. For each anaphor in the sentence do:
    a. Invoke the anaphora resolution handler, which will in turn invoke the proposers and evaluators for all CS's. The output of the handler is a list of ordered pairings of possible antecedents with their overall scores.
    b. Invoke select-best-antecedent, which will decide whether there is enough evidence to make a coextension assertion and, if so, will add the assertion to the core discourse structure and to both anaphor and antecedent referents.

The anaphora resolution handler takes two inputs: an ordered list of available CS's and the anaphoric referent that needs to be resolved. The handler exploits a local data structure called CANDIDATES, which is reinitialized for each anaphoric referent that is processed and which contains a list of all the antecedents that have been proposed for that referent. This makes it possible to avoid considering any given candidate more than once, no matter how often it is proposed. The handler proceeds as follows:

1. For each CS(i) do:
    a. Invoke CS(i)'s proposer, which will return a list (possibly empty) of candidate antecedents, each with an associated initial score.
    b. For each candidate C that is not already on CANDIDATES do:
        i. Add C to CANDIDATES.
        ii. While running score of C is above threshold, for each CS(j) where $j \neq i$ do:
            1. Pass C to the evaluator to get a score.
            2. Update running score for C.

Although these algorithms do not care, in some sense, what list of constraint sources they are given, their success depends on having a set of constraint sources that cover the range of phenomena that occur in the discourse that must be processed. Also, the efficiency with which they reach a conclusion depends on the order in which the CS's are invoked. In Lucy, the recency constraint source is invoked first. The correct antecedent is almost always among the candidates recency proposes since it proposes liberally. To aid efficiency, the recency CS is followed immediately by the simple syntactic filtering CS's (number, gender and animacy agreement) which, with little effort and high certainty, are able to eliminate most candidates from the list that must be evaluated by more complex CS's.

The select-best-antecedent procedure applies to the final list of candidate antecedents. It must decide whether there is sufficient information on which to base a coreference assertion. If there is exactly one candidate with the highest rating and if the difference between that rating and the second highest rating exceeds the threshold δ (a system parameter currently set at 0.5), then a coreference assertion can be made.[3] If one candidate is not clearly best, however, two actions are possible. One is to do nothing (i.e., post no coreference constraint) and thus essentially to produce a partial interpretation of the input sentence. This may be acceptable, depending on the use to which the sentence's interpretation is to be put. For example, it may be possible to wait until subsequent sentences are processed to see if they yield the disambiguating information. If waiting is unacceptable, however, the other available action is to query the user who input the sentence. This option is available since Lucy is being designed to serve as an interactive English front end system; if this same approach were to be used during text comprehension, some alternative, such as choose a candidate and be prepared to back up if necessary, would be required instead.

THE CANDIDATE SCORING PROCEDURE

As we just saw, the final selection of an antecedent from among the set of candidate referents depends on the combined score that is attached to each candidate as a result of the examination of the candidate by the entire set of constraint sources. Thus the design of the scoring procedure has an important effect on the outcome of the resolution process. Our first implementation of this procedure exploited a single score, a number in the range -5 to +5. Each CS gave each candidate a score and the handler averaged the individual scores to form a composite score. The major drawback of this simple scheme is that there is no way for a CS to say, "I have no opinion," since any score it gives necessarily changes the composite score. There is also no way to say, "I have an opinion and here it is, (and I'm very)/(but I'm not at all) confident of it." As a result, the system was highly unstable. It could be tuned to perform fairly well,

[2] In future releases, a more flexible control structure will be exploited.

[3] In the current implementation, the value of δ does not matter very much, since the available CS's provide only very weak preferences or absolute filtering. Future CS's will exploit more domain knowledge to provide more accurate preferences.

but whenever a new CS was added or if a CS was changed even slightly, the whole system had to be retuned.

To remedy these problems, we now use a scoring procedure in which each CS provides both a score and a confidence measure. The score is a number in the range -5 to +5, the confidence a number in the range 0 to 1. The function that combines a set of n (score, confidence) pairs is

$$running\ score = \frac{\sum_{i=1}^{n} score(i) \times confidence(i)}{\sum_{i=1}^{n} confidence(i)}$$

This function computes an average that is weighted not by the number of distinct scores but by the total confidence expressed for the scores. Any CS that wishes to assert no opinion can now do so by giving a confidence of 0 to its opinion, which will then have no effect on a candidate's running score.

Although, in principle, a constraint source may function both as a proposer and as an evaluator and it may assign any (score, confidence) value it likes to a candidate, it turns out that the CS's that have been implemented so far are much more limited in their behavior. Most CS's either propose or evaluate, but not both. And there are patterns of (score, confidence) values that appear to be particularly useful. Each CS that has been implemented so far falls into one of the following four classes:

- Finite set generators (such as disjoint reference when applied to a reflexive pronoun) are constraint sources that propose a fixed set of candidates. They assign all such candidates the same score and that score is a function of the number of competing candidates:

# of candidates to propose	(score, confidence)	contribution of this CS
1	(5, 1)	= 5
2	(4, 1)	= 4
3	(3, 1)	= 3

  These CS's never evaluate (i.e., when asked to do so, they return a confidence of 0.)
- Fading infinite set generators (such as recency) are constraint sources that could keep proposing, perhaps indefinitely, but with lower and lower scores. Recency, for example, uses the following scoring:

Sentence	(score, confidence)	contribution of this CS
n (current)	(1, 0.5)	= 2
n-1	(2, 0.5)	= 1
n-2	(0, 0.5)	= 0

  These CS's never evaluate.
- Filters (such as number and gender agreement) are constraint sources that never propose candidates. They serve only to filter out candidates that fail to meet specific requirements (usually syntactic). Filters use the following two assignments when they evaluate candidates:

	(score, confidence)	contribution of this CS
pass	(0[4], 0)	= 0
fail	(-5, 0.9)	= -5

  These scores have the following effects:
  - pass - Since the confidence level is 0, the score does not matter and no change will be made to the composite score as a result of the evaluator being called. Thus a candidate's score is insensitive to the number of filter CS's that it passes.
  - fail - The low score with high confidence forces the composite score to drop below the minimum threshold eliminating this candidate from future consideration.
- Preferences (such as semantic content consistency) are constraint sources that impose preferences, rather than absolute opinions, on a set of candidates rating each member relative to others in the set. These constraint sources may use the full range of (score, confidence) values.

Although the scoring scheme we have just described exploits more knowledge about a CS's opinion than did our first, simpler one, it is not perfect. It can suffer from the usual problems that arise when numbers are used to represent uncertainty. Future implementations of this system may move more in the direction of symbolic justifications (as used, for example, in (Cohen, 1985)) if they appear to be necessary.

CONSTRAINT SOURCE EXAMPLES

In this section, we describe the constraint sources that have been implemented in Lucy as well as some (preceded by an asterisk) that are envisioned but not yet implemented.

Recency, whose function is to propose referents that have occurred in the recently preceding discourse. Recency has no opinion to offer on anyone else's proposals.

Number Agreement, which knows that singular pronouns must refer to singular things and plural pronouns must refer to plural things. Number Agreement does not propose antecedents; instead it serves only as a filter on candidates that are proposed by other CS's.

Gender Agreement, which knows that any pronoun that is marked for gender can refer only to something of the same gender as itself. Gender serves only as a filter in the current implementation.

Animacy, which knows that neuter pronouns refer to inanimate things, while masculine and

[4]When the confidence rating is 0, the score is arbitrary given the equation for running score values.

feminine pronouns must refer to animate things, usually people. Animacy functions only as a filter.

Disjoint Reference, which knows about structure-based coreference restrictions that apply to reflexive and to nonreflexive pronouns (as described in theories such as (Reinhart, 1983)). Disjoint Reference proposes antecedents for reflexive pronouns (as, for example, in a sentence like, "John saw himself.") For nonreflexive pronouns, it serves as a filter, eliminating, for example, *John* as the antecedent of *him* in the sentence, "John saw him."

Semantic Type Consistency, which functions as a filter and constrains antecedents to only those referents that satisfy the type constraints imposed by the semantic interpretation of the rest of the sentence. For example, consider the discourse, "The system created an error log. It printed it." Assume that the semantic interpretation of *print* imposes the following type constraints on its arguments:

```
agent: human v computer
object: information-structure
```

Then this CS will reject *an error log* as the antecedent of the first occurrence of *it*, assuming that the type hierarchy does not include *log* as a subclass of either *human* or *computer*. Further, this CS will reject *the system* as the antecedent of the second occurrence of *it*, assuming that the type hierarchy does not include *system* as a subclass of *information-structure*.

Global Focus, which knows about objects that are globally salient throughout a discourse. In the current implementation, global Focus acts only when the anaphor being considered is *it*. In that case, it proposes as antecedents all referents that are in global focus. (Empirical evidence in support of this strategy is presented in (Guindon, 1986).) In the current implementation, the target system to which the English front end is attached is assumed always to be in global focus.

Cataphora, which knows about a class of syntactic constructions in which a pronoun can preceed the full lexical NP with which it corefers. This CS will propose *John* as a candidate antecedent for *he* in the sentence *When he is happy, John sings*. Cataphora acts as a generator and will never reject the proposal of another CS.

***Logical accessibility**, which knows about the constraints that are imposed on the accessibility of referents as a function of their embedding within logical structures such as quantifiers and negation (Kamp, 1981). Logical accessibilty functions only as a filter. It rules out, for example, *a donkey* as the antecedent for *it* in the sentence, "If a farmer doesn't own a donkey, he beats it," unless *a donkey* is interpreted as having wide scope over the sentence (i.e., "If there is a donkey such that the farmer doesn't own it then he beats it.")

***Semantic content consistency**, which exploits semantic knowledge about context dependent phenomena as opposed to simply applying static type constraints. The boundary between this CS and semantic type consistency is clearly fuzzy in general and depends in any particular case on the structure of the type hierarchy that is being used. The key difference between the CS's, though, is that accessing a type hierarchy is fast, whereas there are cases in which this CS will have to do arbitrary reasoning.

***Local Focus**, which tracks objects that are locally in focus in the discourse. This is the phenomenon that is addressed by theories such as focus spaces (Grosz, 1977) and centering (Grosz, 1986, Brennan, 1987).

***Rhetorical Structure**, which segments and organizes the discourse as a set of plans for fulfilling conversational goals. This is the phenomenon that is addressed by theories such as (Hobbs, 1985).

***Set generation**, which creates set-level referents that can serve as antecedents for plural pronouns. For example, this CS could propose *Mary and Sue* as the antedecent for *they* in the discourse, "Mary picked up Sue. They went to the movies."

***Generic They**, which knows about salient individuals and groups, and proposes them as antecedents for occurrences of the pronoun *they* in sentences such as, "Why don't they ever fix the roads?"

This list is intended to provide an example of the range of phenomena that can be combined using the architecture we have described. It is not intended to be a definitive list of constraint sources. In fact, the architecture allows for more than one implementation (i.e., CS) of a given theory or more than one theory (and associated implementations) of a given phenomenon. This redundancy can be useful, for example, as way of comparing the effectiveness of competing constraint sources within a complete anaphora resolution system.

DEBUGGING WITHIN ARCHITECTURE

Now that the above architecture has been implemented, further development of the system consists primarily of additions to the set of constraint sources and adjustments to score and confidence assignments. During a test run the developer needs to know which referents are being recognized as anaphors, which CS's get consulted and in what order and, most importantly, what effect each CS has on the overall rating received by each proposed antecedent. Our tracing tools will display this information for each anaphor processed by the handler in the following form. First the name of each proposer as it is called and the list of candidates and ratings it returns are displayed in the tracing window. Then for each of these candidates in turn, the name of every evaluator appears as it is

SAMPLE DISCOURSE:

Sentence 1: **Jon created a file for himself.**

U-1 x1 e1 x2 x3

Sentence 2: **He sent it to Carl and Dave.**

U-2 x4 e5 x6 x7 x8

x9

TRACE OUTPUT:

Invoking anaphor handler with anaphor: **#<X-6>**

Invoking proposer for RECENCY

Possible antecedents proposed:
(X-3 :1) (X-2 :1) (E-1 :1) (X-1 :1) (U-1 :1)
(E-5 :2) (X-4 :2) (X-9 :2) (U-2 :2)

Composite ratings for candidate X-3
After polling evaluator of TYPE rating is 1.0
After polling evaluator of IDENTITY rating is 1.0
After polling evaluator of GLOBAL-FOCUS rating is 1.0
After polling evaluator of ANIMACY rating is -2.857143

Composite ratings for candidate X-2
After polling evaluator of TYPE rating is 1.0
After polling evaluator of IDENTITY rating is 1.0
After polling evaluator of GLOBAL-FOCUS rating is 1.0
After polling evaluator of ANIMACY rating is 1.0
After polling evaluator of DISJOINT-REFERENCE rating is 1.0
After polling evaluator of GENDER rating is 1.0
After polling evaluator of NUMBER rating is 1.0
After polling evaluator of CATAPHORA rating is 1.0

Composite ratings for candidate E-1
After polling evaluator of TYPE rating is -2.857143

.
.
.

Invoking proposer for GENDER
Possible antecedents proposed: *none*

Invoking proposer for NUMBER
Possible antecedents proposed: *none*

Invoking proposer for CATAPHORA
Possible antecedents proposed: *none*

Final candidate ratings:
X-3 -2.857143
X-2 1.0
E-1 -2.857143
X-1 -2.857143
U-1 -2.857143
E-5 -2.5
X-4 -2.5
X-9 -2.5

ANNOTATIONS:

begin resolving anaphor "it"

list of candidates and their initial scores

evaluate first candidate, "himself"

animate candidate cannot corefer with "it"

next candidate to be evaluated is "file"

no CS evaluator wants to reject or support this candidate
"file" survives with orignal score of 1.0

next candidate is the event "create"
event referent filtered out due to type mismatch

.

(Recency's remaining candidates get filtered*)*

.

begin calling other proposers and evaluate any new referents they introduce as candidates

filtering CS's have nothing to propose

Cataphora and Disjoint Reference propose nothing new for this anaphor in this discourse

unordered list of all candidates ever proposed

all candidates except "file" have been rejected

minimum score threshold currently set at -2.50

difference between "file" x-2 and first runner-up (4.5) exceeds delta currently set at 0.5

Figure 3: Tracing the Anaphora Resolution Process

called followed by the effect of that evaluator's response on the running score for the candidate referent. At the end of processing for each anaphor the list of all candidates ever proposed and their composite ratings is displayed. Figure 3 shows an example of the use of the tracing tools.

CONCLUSION

In this paper, we have described an architecture for pronominal anaphora resolution that allows implementations of partial theories of anaphora to be combined into a complete system, and we have illustrated an implementation of such a system. This architecture makes no commitment on the question of what theories should be used or how conflicts among the theories should be resolved. In this respect, it differs from other proposals (such as (Hobbs, 1978)) in which a specific strategy for applying knowledge is encoded into the control structure of the system. As a result of its loose structure, this architecture supports the empirical investigation of the effectiveness of competing theories and their implementations within a complete anaphora resolution system.

One interesting comment that can be made about this architecture is its similarity to architectures that have been used to perform other parts of the natural language understanding task. For example, TEAM (Grosz, 1987) uses a similar architecture and a set of critics to perform quantifier scope assignment. The critics function in much the same way CS's do. And, like CS's, there are classes of critics. For example, some are pure filters. Others impose preferences on the set of candidate interpretations.

ACKNOWLEDGEMENTS

We would like to thank Nicholas Asher, Kent Wittenburg, Dave Wroblewski, Jim Barnett, Jon Schlossberg, and Carl Weir for many discussions about this architecture. We would also like to thank Carl Weir for his contribution to the implementation of this system.

REFERENCES

Brennan, S. E., M. W. Friedman, & C. J. Pollard. (1987). A Centering Approach to Pronouns. *Proceedings ACL*. .

Cohen, R. R. (1985). *Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach*. Boston: Pitman Advanced Publishing Program.

Erman, L. D., P. E. London, & S. F. Fickas. (1981). The Design and an Example Use of Hearsay III. *Proc. IJCAI 7*. .

Grosz, B. J. (1977). The Representation and Use of Focus in a System for Understanding Dialogs. *IJCAI 5*. .

Grosz, B. J., A. K. Joshi, & S. Weinstein. (1986). Towards a Computational Theory of Discourse Interpretation.

Grosz, B. J., D. E. Appelt, P. A. Martin, & F. C. N. Pereira. (May 1987). TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces. *Artificial Intelligence, 32*(2), 173-243.

Guindon, R., P. Sladky, H. Brunner & J. Conner. (1986). The Structure of User-Advisor Dialogues: Is there Method in their Madness? *Proceedings of the 24th Meeting of the Association for Computational Linguistics*. .

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

Hobbs, J. R. (1985). *On the Coherence and Structure of Discourse* (Tech. Rep.). CSLI-85-37,

Kamp, H. (1981). A Theory of Truth and Semantic Representation. In J. Froenendijk, T. Janssen, & M. Stokhof (Eds.), *Formal Methods in the Study of Language, Part 1*. Amsterdam, The Netherlands: Mathematisch Centrum.

Reinhart, T. (1983). *Anaphora and Semantic Interpretation*. Chicago, Ill.: University of Chicago Press.

Rich, E. A., J. Barnett, K. Wittenburg & D. Wroblewski. (1987). Ambiguity Procrastination. *Proceedings AAAI 87*. .

Wittenburg, K. (1986). A Parser for Portable NL Interfaces Using Graph-Unification-Based Grammars. *Proceedings AAAI 86*. .