A PARSER FOR PORTABLE NL INTERFACES USING GRAPH-UNIFICATION-BASED GRAMMARS

Kent Wittenburg

Microelectronics and Computer Technology Corporation

Abstract

This paper presents the reasoning behind the selection and design of a parser for the Lingo project on natural language interfaces at MCC. The major factors in the selection of the parsing algorithm were the choices of having a syntactically based grammar, using a graph-unification-based representation language, using Combinatory Categorial Grammars, and adopting a one-to-many mapping from syntactic bracketings to semantic representations in certain cases. The algorithm chosen is a variant of chart parsing that uses a best-first control structure managed on an agenda. It offers flexibility for these natural language processing applications by allowing for best-first tuning of parsing for particular grammars in particular domains while at the same time allowing exhaustive enumeration of the search space during grammar development. Efficiency advantages of this choice for graph-unification-based representation languages are outlined, as well as a number of other advantages that acrue to this approach by virtue of its use of an agenda as a control structure. We also mention two useful refinements to the basic best-first chart parsing algorithm that have been implemented in the Lingo project.

1. Introduction

In designing a portable natural language (NL) interface, one of the first crucial decisions is whether to require of the grammar used in the system that it be syntactically or semantically based. Existing NL interface systems can be classified along these lines: there are those that use a syntactically based, general grammar of English, e.g., TEAM (Martin, Appelt, and Pereira 1983), versus those that use a semantically based grammar particular to the domain, e.g., Plume (Hayes, Andersen, amd Safier 1985). The semantically based grammars offer customization of the entire system for the domain; robustness of parsing along domain sensitive lines is an advantage usually cited. However, they generally suffer from patchiness of syntactic coverage and the grammar must be rewritten from scratch, in general, for each new domain. The syntactically based grammars, on the other hand, offer the advantage of being able to avoid rehacking the grammar for each new domain and achieving a greater level of generality and sophistication in the syntactic variations of the input. Robustness of parsing is a component in the syntactically-based systems which, along with semantics generally, must be attended to separately; though robustness is obviously not precluded by this initial design choice, it does not come for free since it is not entwined with the grammar itself.

One of the first decisions in the MCC Lingo project on natural language interfaces was to go with a general, syntactically-based grammar. It was felt that, although semantically-based grammars may offer advantages in the short run for relatively unsophisticated systems in highly constrained domains, syntactically based grammars offer a modularity in design that will achieve greater payoffs in the long run. Not only does the modularity enhance transportability to new applications, but also, it makes possible the greater sophistication required for interfaces to knowledge-based applications of the future.

Given this first design choice, the next step was to choose a formalism for grammar representation and an approach to the grammar itself. The representation language chosen was a graph-unification-based formalism, in particular, one based on Karttunen (1984) and Shieber (1984). Graph-unification-based representation languages have had a tremendous impact in the field of computational linguistics, and in fact have been incorporated in one form or another into a number of influential linguistic theories (see Shieber 1986 for discussion). Among the many advantages of a graph-unification-based formalism are (i) it is easy to use, requiring no special training for grammar writers with linguistics backgrounds; (ii) it is a language separable from any particular machine-dependent implementation and thus amenable to optimizations at many levels; (iii) it avoids the typical explosion of ad hoc procedural operations in the grammar, being a purely declarative language; (iv) it is very flexible, accommodating a variety of grammatical theories; and (v) it is order free, which among other things means that the same grammar rules can be used to generate as well as to analyze.

Our choice for an approach to the grammar was Combinatory Categorial Grammar (Ades and Steedman 1982, Steedman 1985). Though untested in natural language applications to date, we felt this approach to grammar was particularly promising in the following respects: (i) it handles English extraction phenomena (wh-questions, relative clauses, etc.) efficiently and elegantly without resorting to empty rewrite rules or complicated feature passing schemes; (ii) it accounts for more of English coordination than alternative approaches without special rules or ad hoc operations; (iii) it offers a method of accounting for free word order and partially free word order with a simple and easily extendable formalism; (iv) it suggests natural techniques for dealing with lexical ambiguity and heuristic rule preferencing; (v) it is particularly suitable for left-to-right, incremental processing designs; and (vi) it is able to accomplish all this with a very small rule base, relative to the alternatives.

Given the three design decisions just mentioned, we now come to the subject of this paper, namely, the selection and design of a parser for NL interface applications using the grammars and the representation language we have just mentioned.

2. Charts

First let us ask what we should expect from the parser independently of the choices related to the grammar. The following desiderata should speak for themselves:

- A formally sound basis for the algorithm in order to ensure termination, completeness, and tractability.
- Modes for grammar development that maximize debugging facilities and inspection of parsing steps.
- The ability to tune particular grammars in particular domains such that prototypes for efficient applied systems can be developed.
- The potential to integrate semantics and contextual factors into preferencing factors for the purposes of tuning.
- A design that in principle allows for incorporating credit assignment schemes to automate adaptation to individual performance situations.
- A design that does not preclude future adaptation to parallel processing schemes.

For maximum flexibility and formal soundness, the most obvious place to begin in constructing a parser is with some variant of chart parsing (Kay 1980). The many advantages of charts have been extolled by Kay and others and will not be repeated here. One of the most persuasive pieces of evidence in favor of charts is their widespread adoption. Charts, or something very similar, have figured prominently in important theoretical work on parsing from the computer science perspective, e.g., Earley (1970), in natural language research, e.g., Kaplan (1973), Bear and Karttunen (1979), Thompson (1981), Ford, Bresnan, and Kaplan (1982), Martin, Church, and Patil (1981), Shieber (1985), and in applied systems, e.g., Slocum (1984).

However, despite the popularity of chart parsing in the literature, there has been relatively little attention in the more theoretical quarters to the role of agendas in chart parsing. For whatever reason, there seems to be an association of chart parsing in most published work with exhaustive breadth-first control designs, despite the fact that Kay (1980) and also Kaplan (1973) have written of the possibility of using an agenda to control the enumeration order in maximally flexible ways. Research into agenda driven chart parsers, on the other hand, seems to have been mainly driven by psycholinguistic concerns (e.g., Ford, Bresnan, and Kaplan 1982), not specifically by the need to develop efficient applied systems.

However, for the needs of NL interface development, an agenda is in fact the key ingredient. The advantages of having an agenda as a control mechanism for a parser go well beyond matters of increased efficiency. Retaining maximum flexibility for later adjustments to a parser without having to scrap or radically alter existing code or existing grammars is a major advantage. We will see an example shortly of the relative ease of adding meta-level operations to the agenda structure. Other possible uses of an agenda are (i) as a means of integrating semantics and contextual factors into the scoring; (ii) as a framework for credit assignment schemes to automate adaptation to individual performance situations; and (iii) as the control mechanism for parallel processing of parsing steps, a natural use of agendas pointed out by Kay (1980), among others. And, to complete our list of criteria, the presence of an inspectable control structure makes it possible to develop sophisticated grammar development tools so that the state of a parse can be examined at any point.

Another point worth making about chart parsing with agendas is its relation to questions of whether the control proceeds left-right, right-left, or some version of middle-out in processing natural language input. Enumeration in this respect is completely controlled by the heuristic ordering on the agenda, and any of these designs can be achieved as long as appropriate heuristics and agenda operations can be designed. In fact, an agenda can be designed to allow any mix of these orders; attention can be directed to any arbitrary area of the chart at any time, a property which produces the effect of being able to suspend processing on less promising paths with the possibility of resuming such processing later if necessary. Thus it is hard to imagine a more general algorithm with respect to control decisions.

2.1. Exhaustive vs. Partial Enumeration

While many chart-parsing algorithms use control schemes that exhaustively enumerate the search space, there are a number of reasons why exhaustive enumeration is unsuitable for NL interface applications given the choices mentioned. The most obvious reason is that syntactically based grammars of extensive coverage admit staggering amounts of syntactic ambiguity for certain kinds of constructions. For example, working with a corpus collected through a Wizard-of-Oz experiment, Martin, Church, and Patil (1981) found 958 parses for the sentence *In as much as allocating costs is a tough job I would like to have the total costs related to each product.* It seems safe to say that no matter what optimizations are added to the parser, and Martin, Church, and Patil added many, exhaustive enumeration in the face of such data will probably not lead to satisfactory performance for a natural language interface application.

Our choices relating to the representation language and the grammar approach add further weight to this argument against exhaustive enumeration. It is a commonly acknowledged fact that unification of graphs tends to be expensive computationally, mostly because unification is destructive of the graphs on which it is called (see, e.g., Karttunen 1984). The expense is particularly evident in chart parsing because, by definition, one needs to retain edge graphs in their original state before unification as well as create new edge graphs that reflect the result of unifying the originals. Optimizations of the unification algorithm itself have been suggested as a means for coping with this problem (Pereira 1985; Karttunen and Kay 1985). While such optimizations are obviously welcome, another place to cut costs is to minimize the calls to unification in the first place.* Avoiding complete enumeration is a first step in this line.

Certain properties of Combinatory Categorial Grammars also suggest that exhaustive enumeration in parsing is inappropriate. As is detailed in Wittenburg (1985), a consequence of the mechanism for handling extraction and certain kinds of coordination in these grammars is the potential for ambiguous interpretations that have identical content. Another way to say this is that there are multiple paths through the search space of rule firings which lead to the identical solution. The obvious search method to use in such a domain avoids enumeration of all paths, since there is no obvious reason to do so. This general plan

*It is relevant to report, based on unpublished work by David Wroblewski, that unification using structure sharing may not yield the expected dramatic improvements in parsing performance. In fact, unification with structure sharing has been (at least temporarily) shelved in the Lingo project.

has been embraced in the theoretical linguistics literature dealing with the formal devices utilized in Combinatory Categorial Grammar. It has been suggested that one should use heuristic strategies in processing with these grammars in order to control which grammar rules are most appropriate to fire in a given context; as long as one proceeds forward to a successful parse, one has no reason to fire all the rules one can.

There is one further motivation for avoiding exhaustive enumeration that should be mentioned. Let us assume that certain individual syntactic analyses admitted by the grammar are designed so as to be ambiguous with respect to semantic interpretation. This general idea has previously surfaced in work by Church (1980), Pereira (1983), and Marcus, Hindle, and Fleck (1983). In Wittenburg (1986a) various arguments are advanced for using a one-to-many mapping from syntactic bracketings to semantic interpretations in the case of extraposition from noun phrases. Rich et al. (1986) give an account of how such ideas can be extended to a variety of syntactic constructions and to the semantic representation itself. Given this picture, we again have a case of a search domain in which it is inappropriate to generate all paths. For if we are able to reach a semantic interpretation through more than one path, then we have no reason to continue generating all paths once we have reached the first one. The reason we can reach the same solution in these cases is different than the reason mentioned above. We are assuming that within the set of interpretations assigned to some syntactic analysis reached by one path, there may be one or more of those interpretations that are reachable by a different path. An example would be a treatment of prepositional phrase attachment in which, say, "high" attachment would yield the full set of attachments from the semantic point of view while "low" attachment would yield only one semantic interpretation that happens to be a member of the first set.

Despite all these arguments against using exhaustive enumeration in an application mode, there are arguments *for* using exhaustive enumeration in a grammar development mode. Assuming that there will be times when parsing fails in an application (whether "hard" or "soft"), circumstances will probably arise in which all the search space will have to be explored. It is very important to have anticipated such an event during grammar development so that undesirable rule interactions can be eliminated. Also it is important to take note of the performance characteristics of the parser under such circumstances. Thus a parser that can be toggled between exhaustive and minimal enumeration, as well as arbitrary points in between, is the best we can ask for. Agenda controlled chart parsing offers this sort of flexibility.

2.2. Enumeration Order

The Earley algorithm (Earley 1970), along with other breadth-first ordering schemes associated with chart parsing, is designed for exhaustive enumeration of the grammar with respect to some string during parsing. Although such a control mechanism by itself is ill-suited for anything but exhaustive enumeration, there have been some efforts at partitioning grammars in such a way as to get partial enumeration with basically breadth-first ordering (Slocum 1984). Given our additional motivation for keeping rule firings to a minimum, however, even restricted breadth-first search is less desirable than some of the alternatives. We should distinguish, then, what is about to be proposed from other suggestions in the literature for heuristically ordering a collection of parses achieved with some form of breadth-first enumeration (e.g., Robinson 1982, Heidorn 1982, Slocum 1984). What is of maximum benefit for our purposes is a design that could return just the best parse, first of all, but that could continue to enumerate other parses *if subsequent semantic processing deems it necessary.*

Among the alternatives to breadth-first order is a depth-first, backtracking design, but since a strength of charts in general is that they do not require backtracking, a more attractive choice is a so-called "best-first" control scheme. Best-first search is of course a well-known paradigm in the A.I. literature. A number of interesting observations can be made about chart parsing from the perspective of heuristic graph search.

- The search can be represented as a standard **or** graph (as opposed to an **and/or** graph).
- The system is commutative in the sense of Nilsson (1980); thus the control scheme can be *irrevocable*, i.e., it need not involve backtracking.
- The chart itself as database has, under certain conditions, the implicit effect of merging nodes in the search graph appropriately; thus, there may be no need to check whether newly added nodes have already been generated in the search graph.
- The relative lengths of alternate paths through the search graph to the solution are of little importance; thus *admissability* of the search algorithm is not a strong factor.
- On the other hand, the *optimality* of the algorithm, i.e., a measure of the total number of nodes in the complete search graph that are expanded, will have a strong bearing on efficiency.
- The set of edges appearing on a chart can be looked at as the *closed* nodes in a best-first search algorithm; the other data structure we need is one to keep track of *open* nodes, which can be viewed as an agenda of possible actions to take.**

These characteristics of chart parsing, then, allow a simplification of the general graph-searching procedure presented in Nilsson (1980). We now turn to an overview of the algorithm itself.

3. The Best-First Algorithm

My purpose here is not to present the best-first chart parsing algorithm in depth. Readers may consult the original sources or Wittenburg (1986b) for detail in this respect. What I wish to do is highlight certain features of the algorithm that help achieve the goals outlined above.

The parser for this project has been dubbed Astro, which is a reflection of the importance of the A* search algorithm (Hart, Nilsson, and Raphael 1968) in its design. The algorithm we use is less general than previously published chart parsing algorithms in two respects. We assume a grammar whose rules have only binary or unary daughters. Such a simplification is a consequence of the

** Intuitively, closed nodes in a heuristic search algorithm are options that have been exercised; open nodes are options which have been generated during the search but which have not yet been exercised. The observation that chart edges can be viewed as closed nodes must be understood in light of the fact that in the algorithm discussed here, an edge is placed on the chart concurrent with the expansion of all successors of that edge on the agenda.

grammars being used in the MCC Lingo project, and not an inherent restriction due to chart parsing. Most published chart parsing algorithms generalize to grammars that have rules with right hand sides of arbitrary length by using the dotted rule technique introduced by Earley (1970). However, there is no motivation for complicating the algorithm in this way when using Combinatory Categorial Grammars; the effect of dotted rules is in fact already achieved in the categories of such grammars. The only binary rules that are needed are a small, fixed number of very general combination rules, operating over these complex categories. The remainder of the grammar consists of unary rules, which have the effect of altering the complex categories in some way. The second point about this chart parsing algorithm with respect to previously published ones is that it does not permit formal operations such as transformations or register setting as was done by Kaplan (1973), nor is it designed for backtracking. In our experience, these simplifications allow more flexibility in some of the agenda maintence aspects of the system.

3.1. The Main Loop

The following procedure suffices as a basis for the main loop; it follows the basic organization of Nilsson (1980:64).

1. Initialize *agenda*.
2. Initialize *chart*.
3. LOOP: if *agenda* is empty, exit with failure.
4. Select the best action from the *agenda*, remove it from the *agenda*, and apply the action. Set *best-edge* to the new edge added to the *chart*, if any; else, NIL.
5. If *best-edge* satisfies the terminating conditions, exit successfully.
6. If *best-edge* is non-nil, generate the set M of its successors. Install the members of M on the *agenda*, following a heuristic ordering scheme.
7. Go LOOP.

3.2. Generating Successors

The critical feature of this algorithm that achieves a major efficiency advantage for graph-unification-based formalisms can be found in the *generate successors* step, which appears in step 6 of the main loop, and also in chart initialization. We make a critical distinction between *checking* to see if a rule call is likely to succeed and actually *applying* a rule to a set of edges. The checking operation is a part of generating the successors of a new edge on the chart, leading to augmentations on the agenda only, while actually applying a rule call is done only when invoking the highest ranked action on the agenda. Given this distinction, we can make make use of an optimized *test* function for checking edge graphs for rule application, leaving actual unification with its destructive effects to applying a rule call. Since the latter operation is invoked much less often than the generate successors step, there are major savings in unification costs. For graph unification-based grammars, at least three options present themselves for the test function. First is Shieber's notion of restricted unification (Shieber 1985). Second, Karttunen (1986) has suggested a scheme where the destructive effects of unification can be undone. Last, a more "porous", but more efficient *check-graphs* function could be used that is nondestructive of its graph arguments. The last of these options is the one used in the Lingo project; it seems the best choice given our algorithm because there are no penalties, except a slight downgrade in performance, if rule calls generated in step 6 fail to apply successfully once they are chosen upon iteration in step 4.

3.3. The heuristic function

Critical to the success of this algorithm, as with all algorithms based on heuristic graph search, is the choice of a heuristic function to order the successors of a given node expansion. Developing the right set of heuristics is the product of intuition, trial, and error, and depends upon the particulars of the grammar and the domain. Here we mention some general factors that are a start for designing a heuristic function.

- An important factor in scoring any rule call is the span of the edge to be added to the chart if the rule fires successfully. This span can be computed from the spans of the daughter edges in the rule call. In general, longer spans should be favored.[***]
- The rules that are more likely to lead to success should be given higher intrinsic scores, and play a role in scoring a rule call.[****]
- The linguistic content in the edges involved in a rule call should also play a role. Differentiating among the scores of ambiguous lexical sense assignments is one method for designing heuristics that take advantage of the fact that some word senses will be statistically more likely than others in certain domains of discourse.

Naturally, such heuristics will tend to become much more refined and sophisticated through experience with particular grammars, lexicons, and corpora. Automated techniques for statistical information gathering and heuristic refinement are always a possibility.

4. Refinements

We make mention here of two useful refinements to the basic chart parsing algorithm. The first introduces a technique for making environments for unary rules more restrictive, thus avoiding ultimately useless unary rule applications.[*****] Also we review a technique to partition the rules of the grammar that is used to optimize the *generate successors* step. The partitioning of the grammar in this way also allows for further heuristic control of the parser's actions.

4.1. Avoiding useless unary rule proposals

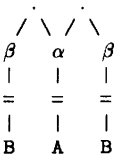
Many chart parsing algorithms developed for context-free grammars make use of relations in the grammar which can prune the total search space. Kay (1980) mentions collecting this sort of *reachability* information into tables which help control parsing actions. Calls to unary rule productions are an obvious candidate to attempt to prune from the search space for Combinatory Categorial Grammars. It is a general fact about unary rules that

[***] Note that this scheme by itself will tend to favor binary rule applications over unary rule applications.

[****] Among the rules may be some whose function is to recover from ungrammatical input. These, in general, will receive a lower priority.

[*****] This augmentation also helps to establish a heuristic scoring technique for agenda items involving unary rule calls. See Wittenburg (1986b) for details.

they tend to be overly promiscuous, i.e., typically, the right-hand-side of the rule is a poor measure by itself of the ultimate usefulness of a unary rule application. Even when a unary rule call can succeed in adding an additional edge to the chart, that edge often turns out to be unable to combine with edges on either side. Adding superfluous edges like this in chart parsing has the effect of making subsequent operations more expensive since this new edge must now be taken under consideration in all *generate successors* procedures involving any adjacent edges. What is called for then is the computation of some grammar relation that can be used to further restrict the conditions for applications of unary rules. A relation that has been found to be useful in this regard we call the *extended sister relation*. It is defined as follows: node A is an extended sister of node B if and only if there exists some node α that is a sister of node β where α is a non-branching exhaustive dominator of A and β is a non-branching exhaustive dominator of B. In the derivation tree below, extended sisters of A are shown as all B's that can stand in the relation shown.



The extended sister relation is used in the following way. We precompute a grammar table that holds the extended sisters for the left-hand-sides of each unary rule. An additional condition for unary-rule-call successors of a new edge is now that some adjacent edge must match at least one of the extended sisters of the left-hand-side of that unary rule. As pointed out in Wittenburg (1986b), a consequence of this move is that the generate successors step now has to consider as successors of a given edge not just those unary rule calls which apply to the edge directly, but also those unary rule calls which involve the edge as an extended sister.

4.2. Meta Agenda Items

An additional augmentation that we have implemented involves adding a new type of agenda item. In the basic best-first algorithm, agenda items are made up of rule calls over a set of edges. These agenda items are generated by checking these edges against all the rules in in the grammar, weeding out all rules which fail the test. It is also possible to define a meta agenda item that generates these basic agenda items, i.e., that itself generates these rule calls. This augmentation is a way of breaking apart the iteration of checking the edges with respect to the whole grammar into n steps corresponding to a partitioning of the set of grammar rules into n cells.

Adding this sort of agenda item has the effect that in the *generate successors* step, we produce agenda items of this new type at the first pass. These new agenda items will have the form $[i, P, E']$, where i is a heuristic score assignment, P is a partition cell of the grammar rules, and E' is a triple that represents the new edge and its adjacent edges at the time the new edge was added. Exercising an agenda item of this form involves checking the edges in E' with respect to the grammar rules of partition P only. Any rules which then pass these further tests will result in rule calls added to the agenda of a type that we assumed previously.

The advantages of incorporating this new type of agenda item are that it is possible to heuristically order the iteration over the grammar by assigning different weights to partitions as a whole. Thus we can postpone the checking of subsets of rules that are less likely or perhaps whose checking is more expensive than other subsets. Also, it is possible to define a single check for an entire grammar partition that in one fell swoop can eliminate the further consideration of any of those rules for the edge sets in question.

5. Evaluation and further study

Since the makeup of the heuristic function itself plays such a critical role in a parser based on heuristic graph search, it is difficult to evaluate the efficiency of the design in the general case when compared to other non-heuristically based designs. For chart parsing systems that use different representation languages and different grammars than the ones used in the Lingo project, some of the arguments advanced here in favor of a best-first enumeration order would lose force. However, given an NL interface application, a graph-unification-based representation language, Combinatory Categorial Grammars, and a many-to-one mapping from syntactic bracketings to semantic representations, it is hard to imagine that any radically different alternative could beat the parsing program we have outlined here on the grounds of efficiency, clarity, and flexibility. Experience in the Lingo project indicates an overwhelming performance improvement with a best-first parsing design when compared to a non-selective, bottom-up, breadth-first design that figured in our original bootstrapping operation.

Current research involves evaluation and refinement of heuristic scoring methods as well as a consideration of further design changes involving the agenda. We are using a set of tools for the tuning of grammars for particular corpora which take advantage of the inspectable control structure of the chart. Longer-term research includes the possibility of designing credit assignment schemes to automatically adapt to specific performance situations. We also count the possibility of incorporating some form of buffering within the agenda structure so that attention can be focused incrementally on local areas of the chart. Initial experience indicates that such a factor would improve the reliability of heuristics, which tends to be best when considered in a local fashion rather than globally. Also, left-to-right buffering could lead to a cascading design feeding into semantic and pragmatic components, an important step towards any future approximations of real-time parsing.

Acknowledgements

The work reported on here involved many others at MCC besides the author. I gratefully acknowledge the contribution of other members of the Lingo team to this work. In particular, Elaine Rich made extensive contributions to the research itself and also offered valuable comments on earlier drafts of this paper. David Wroblewski has also made significant contributions to this work, including the coding of structure-sharing algorithms for graph unification and the design and coding of the window system interface to the parser and grammar development environment. I also wish to thank Lauri Karttunen for his long-standing support and advice on parsing matters.

References

1. Ades, A. and M. Steedman. 1982. On the Order of Words. Linguistics and Philosophy 4: 517-558.

2. Bear, J., and L. Karttunen. 1979. PSG: A Simple Phrase Structure Parser. Texas Linguistic Forum 15: 1-46.

3. Church, K. 1980. On Memory Limitations in Natural Language Processing. MIT Doctoral Dissertation. [Available through the Indiana University Linguistics Club.]

4. Earley, J. 1970. An Efficient Context-Free Parsing Algorithm. Communications of the ACM 13:94-102.

5. Ford, M., J. Bresnan, and R. Kaplan. 1982. A Competence-Based Theory of Syntactic Closure. In J. Bresnan (ed.), The Mental Representation of Grammatical Relations, pp. 727-796. Cambridge, Mass.: MIT.

6. Hart, P., N. Nilsson, and B. Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on SSC 4:100-107.

7. Hayes, P., P. Andersen, and S. Safier. 1985. Semantic Caseframe Parsing and Syntactic Generality. In Proceedings of the 23rd Meeting of the Association for Computational Linguistics, pp. 153-160.

8. Heidorn, G. 1982. Experience with an Easily Computed Metric for Ranking Alternative Parses. In Proceedings of the 20th Meeting of the Association for Computational Linguistics, pp. 82-84.

9. Kaplan, R. 1973. A General Syntactic Processor. In R. Rustin (ed.), Natural Language Processing, pp. 193-241. New York: Algorithmics.

10. Karttunen, L. 1984. Features and Values. Proceedings of Coling, pp. 28-33. Association for Computational Linguistics.

11. Karttunen, L. 1986. HUG: A development environment for unification-based grammars. Unpublished ms., SRI International and CSLI, Stanford University.

12. Karttunen, L., and M. Kay. 1985. Structure Sharing with Binary Trees. In Proceedings of the 23rd Meeting of the Association for Computational Linguistics, pp. 133-136.

13. Kay, M. 1980. Algorithm Schemata and Data Structures in Syntactic Processing. Xerox Palo Alto Research Center, tech report no. CSL-80-12.

14. Marcus, M., D. Hindle, and M. Fleck. 1983. D-Theory: Talking about Talking about Trees. In Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics, pp. 129-136.

15. Martin, P., D. Appelt, and F. Pereira. 1983. Transportability and Generality in a Natural-Language Interface System. In Proceedings of IJCAI, pp.574-581.

16. Martin, W., K. Church, and R. Patil. 1981. Preliminary Analysis of a Breadth-First Parsing Algorithm: Theoretical and Experimental Results. MIT tech report no. MIT/LCS/TR-261.

17. Nilsson, N. 1980. Principles of Artificial Intelligence. Palo Alto, Ca.: Tioga.

18. Pereira, F. 1983. Logic for Natural Language Analysis. Technical note 275, A.I. Center, SRI International.

19. Pereira, F. 1985. A Structure-Sharing Representation for Unification-Based Grammar Formalisms. In Proceedings of the 23rd Meeting of the Association for Computational Linguistics, pp. 137-144.

20. Rich, E., J. Barnett, K. Wittenburg, and G. Whittemore. 1986. Ambiguity and Procrastination in NL Interfaces. Technical report no. HI-073-86, Microelectronics and Computer Technology Corporation.

21. Robinson, J. 1982. DIAGRAM: A Grammar for Dialogues. Communications of the ACM 25:27-37.

22. Shieber. S. 1984. The Design of a Computer Language for Linguistic Information. Proceedings of Coling84, pp. 362-366. Association for Computational Linguistics.

23. Shieber, S. 1985. Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms. In Proceedings of the 23rd Meeting of the Association for Computational Linguistics, pp. 145-152.

24. Shieber, S. 1986. An Introduction to Unification-Based Approaches to Grammar. Chicago: University of Chicago Press, forthcoming.

25. Slocum, J. 1984. METAL: The LRC Machine Translation System. Paper presented at the ISSCO Tutorial on Machine Translation, April 2-6, 1984, Lugano, Switzerland. [Working paper no. LRC-84-2, Linguistics Research Center, University of Texas at Austin.]

26. Steedman, M. 1985. Dependency and Coordination in the Grammar of Dutch and English. Language 61:523-568.

27. Thompson, H. 1981. Chart Parsing and Rule Schemata in PSG. In Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics, pp. 167-172.

28. Wittenburg, K. 1985. Some Properties of Combinatory Categorial Grammars of Relevance to Parsing. Paper presented at the Annual Meeting of the Linguistics Society of America, December 27-30, Seattle. [MCC tech report no. HI-012-86.]

29. Wittenburg, K. 1986a. Extraposition from NP as Anaphora. To appear in Syntax and Semantics, Volume 20: Discontinuous Constituencies. New York: Academic. [MCC tech report no. HI-118-85.]

30. Wittenburg, K. 1986b. Parsing as Heuristic Graph Search. Technical report no. HI-075-86, Microelectronics and Computer Technology Corporation.