THE COMPUTATIONAL DIFFICULTY OF ID/LP PARSING

G. Edward Barton, Jr.
M.I.T. Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139

ABSTRACT

Modern linguistic theory attributes surface complexity to interacting subsystems of constraints. For instance, the ID/LP grammar formalism separates constraints on immediate dominance from those on linear order. Shieber's (1983) ID/LP parsing algorithm shows how to use ID and LP constraints directly in language processing, without expanding them into an intermediate "object grammar." However, Shieber's purported $O(|G|^2 \cdot n^3)$ runtime bound underestimates the difficulty of ID/LP parsing. ID/LP parsing is actually NP-complete, and the worst-case runtime of Shieber's algorithm is actually exponential in grammar size. The growth of parser data structures causes the difficulty. Some computational and linguistic implications follow; in particular, it is important to note that despite its potential for combinatorial explosion, Shieber's algorithm remains better than the alternative of parsing an expanded object grammar.

INTRODUCTION

Recent linguistic theories derive surface complexity from modular subsystems of constraints; Chomsky (1981:5) proposes separate theories of bounding, government, θ-marking, and so forth, while Gazdar and Pullum's GPSG formalism (Shieber, 1983:2ff) uses immediate-dominance (ID) rules, linear-precedence (LP) constraints, and metarules. When modular constraints are involved, rule systems that multiply out their surface effects are large and clumsy (see Barton, 1984a). The expanded context-free "object grammar" that multiplies out the constraints in a typical GPSG system would contain trillions of rules (Shieber, 1983:1).

Shieber (1983) thus leads in a welcome direction by showing how ID/LP grammars can be parsed "directly," without the combinatorially explosive step of multiplying out the effects of the ID and LP constraints. Shieber's algorithm applies ID and LP constraints one step at a time, as needed. However, some doubts about computational complexity remain. Shieber (1983:15) argues that his algorithm is identical to Earley's in time complexity, but this result seems almost too much to hope for. An ID/LP grammar G can be much smaller than an equivalent context-free grammar G'; for example, if G_1 contains only the rule $S \rightarrow_{\text{ID}} abcde$, the corresponding G'_1 contains $5! = 120$ rules. If Shieber's algorithm has the same time complexity as Earley's, this brevity of expression comes free (up to a constant). Shieber says little to allay possible doubts:

> We will not present a rigorous demonstration of time complexity, but it should be clear from the close relation between the presented algorithm and Earley's that the complexity is that of Earley's algorithm. In the worst case, where the LP rules always specify a unique ordering for the right-hand size of every ID rule, the presented algorithm reduces to Earley's algorithm. Since, given the grammar, checking the LP rules takes constant time, the time complexity of the presented algorithm is identical to Earley's .... That is, it is $O(|G|^2 n^3)$, where $|G|$ is the size of the grammar (number of ID rules) and n is the length of the input. (:14f)

Among other questions, it is unclear why a situation of maximal constraint should represent the worst case. *Minimal* constraint may mean that there are more possibilities to consider.

Shieber's algorithm *does* have a time advantage over the use of Earley's algorithm on the expanded CFG, but it blows up in the worst case; the claim of $O(|G|^2 \cdot n^3)$ time complexity is mistaken. A reduction of the vertex-cover problem shows that ID/LP parsing is actually NP-complete; hence this blowup arises from the inherent difficulty of ID/LP parsing rather than a defect in Shieber's algorithm (unless $\mathcal{P} = \mathcal{NP}$). The following sections explain and discuss this result. LP constraints are neglected because it is the ID rules that make parsing difficult. Attention focuses on *unordered context-free grammars* (UCFGs; essentially, ID/LP grammars *sans* LP). A UCFG rule is like a standard CFG rule except that when used in a derivation, it may have the symbols of its expansion written in any order.

SHIEBER'S ALGORITHM

Shieber generalizes Earley's algorithm by generalizing the dotted-rule representation that Earley uses to track progress through rule expansions. A UCFG rule differs from a CFG rule only in that its right-hand side is unordered; hence successive accumulation of set elements replaces linear advancement through a sequence. Obvious interpretations follow for the operations that the Earley parser performs on dotted rules: $X \rightarrow \{\}.\{A, B, C\}$ is a typical *initial* state for a dotted UCFG rule; $X \rightarrow \{A, B, C\}.\{\}$ is a typical *completed* state; $Z \rightarrow \{W\}.\{a, X, Y\}$ *predicts* terminal a and nonterminals X, Y; and $X \rightarrow \{A\}.\{B, C, C\}$ should be *advanced* to $X \rightarrow \{A, C\}.\{B, C\}$ after the predicted C is located.[1] Except for these changes, Shieber's algorithm is identical to Earley's.

As Shieber hoped, direct parsing is better than using Earley's algorithm on an expanded grammar. If Shieber's parser is used to parse *abcde* according to G_1, the state sets of the parser remain small. The first state set contains only $[S \rightarrow \{\}.\{a, b, c, d, e\}, 0]$, the second state set contains only $[S \rightarrow \{a\}.\{b, c, d, e\}, 0]$, and so forth. The state sets grow much larger if the Earley parser is used to parse the string according to G'_1 with its 120 rules. After the first terminal a has been processed, the second state set of the Earley parser contains $4! = 24$ states spelling out all possible orders in which the remaining symbols $\{b, c, d, e\}$ may appear: $[S \rightarrow a.bcde, 0]$, $[S \rightarrow a.cedb, 0]$, and so on. Shieber's parser should be faster, since both parsers work by successively processing all of the states in the state sets. Similar examples show that the Shieber parser can have an arbitrarily large advantage over the use of the Earley parser on the object grammar.

Shieber's parser does not *always* enjoy such a large advantage; in fact it can blow up in the presence of ambiguity. Derive G_2 by modifying G_1 in two ways. First, introduce dummy categories A, B, C, D, E so that $A \rightarrow a$ and so forth, with $S \rightarrow ABCDE$. Second, let x be ambiguously in any of the categories A, B, C, D, E so that the rule for A becomes $A \rightarrow a \mid x$ and so on. What happens when the string *xxxxa* is parsed according to G_2? After the first three occurrences of x, the state set of the parser will reflect the possibility that *any three* of the phrases A, B, C, D, E might have been seen and *any two* of them might remain to be parsed. There will be $\binom{5}{3} = 10$ states reflecting progress through the rule expanding S; $[S \rightarrow \{A, B, C\}.\{D, E\}, 0]$ will be in the state set, as will $[S \rightarrow \{A, C, E\}.\{B, D\}, 0]$, *etc.* There will also be 15 states reflecting the completion and prediction of phrases. In cases like this, Shieber's algorithm enumerates all of the combinations of k elements taken i at a time, where k is the rule length and i is the number of elements already processed. Thus it can be combinatorially explosive. Note, however, that Shieber's algorithm is still better than parsing the object grammar. With the Earley parser, the state set would reflect the same possibilities, but encoded in a less concise representation. In place of the state involving $S \rightarrow \{A, B, C\}.\{D, E\}$, for instance, there would be $3! \cdot 2! = 12$ states involving $S \rightarrow ABC.DE$, $S \rightarrow BCA.ED$, and so forth.[2] Instead of a total of 25 states, the Earley state set would contain $135 = 12 \cdot 10 + 15$ states.

With G_2, the parser could not be sure of the *categorial identities* of the phrases parsed, but at least it was certain of the *number* and *extent* of the phrases. The situation gets worse if there is uncertainty in those areas as well. Derive G_3 by replacing every x in G_2 with the empty string ϵ so that an A, for instance, can be either a or nothing. Before any input has been read, state set S_0 in Shieber's parser must reflect the possibility that the correct parse may include *any of the* $2^5 = 32$ *possible subsets* of A, B, C, D, E as empty initial constituents. For example, S_0 must include $[S \rightarrow \{A, B, C, D, E\}.\{\}, 0]$ because the input might turn out to be the null string. Similarly, S_0 must include $[S \rightarrow \{A, C, E\}.\{B, D\}, 0]$ because the input might be *bd* or *db*. Counting all possible subsets in addition to other states having to do with predictions, completions, and the parser start symbol that some implementations introduce, there will be 44 states in S_0. (There are 338 states in the corresponding state when the object grammar G'_3 is used.)

How can Shieber's algorithm be exponential in grammar size despite its similarity to Earley's algorithm, which is polynomial in grammar size? The answer is that Shieber's algorithm involves a much larger bound on the number of states in a state set. Since the Earley parser successively processes all of the states in each state set (Earley, 1970:97), an explosion in the size of the state sets kills any small runtime bound.

Consider the Earley parser. Resulting from each rule $X \rightarrow A_1 \ldots A_k$ in a grammar G_a there are only $k - 1$ possible dotted rules. The number of possible dotted rules is thus bounded by the number of symbols that it takes to write G_a down, *i.e.* by $|G_a|$. Since an Earley state just pairs a dotted rule with an interword position ranging from 0 to the length n of the input string, there are only $O(|G_a| \cdot n)$ possible states; hence no state set may contain more than $O(|G_a| \cdot n)$ (distinct) states. By an argument due to Earley, this limit allows an $O(|G_a|^2 \cdot n^3)$ bound to be placed on Earley-parser runtime. In contrast, the state sets of Shieber's parser may grow much larger relative to grammar size. A rule $X \rightarrow A_1 \ldots A_k$ in a UCFG G_b yields not $k + 1$ ordinary dotted rules, but but 2^k possible dotted UCFG rules tracking accumulation of set elements. In the worst case the grammar contains only one rule and k is on the order of $|G_b|$; hence a bound on the number of possible dotted UCFG rules is not given by $O(|G_b|)$, but by $O(2^{|G_b|})$. (Recall the exponential blowup illustrated for grammar G_3.) The parser sometimes blows up because there are exponentially *more possible ways* to to progress through an unordered rule expansion than an through an ordered one. In ID/LP parsing, the easiest case occurs

[1] For more details see Barton (1984b) and Shieber (1983). Shieber's representation differs in some ways from the representation described here, which was developed independently by the author. The differences are generally inessential, but see note 2.

[2] In contrast to the representation illustrated here, Shieber's representation actually suffers to some extent from the same problem. Shieber (1983:10) uses an ordered sequence instead of a multiset before the dot; consequently, in place of the state involving $S \rightarrow \{A, B, C\}.\{D, E\}$, Shieber would have the $3! = 6$ states involving $S \rightarrow \alpha.\{D, E\}$, where α ranges over the six permutations of ABC.

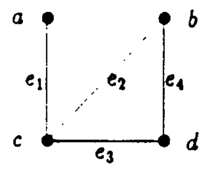

Figure 1: This graph illustrates a trivial instance of the vertex cover problem. The set $\{c, d\}$ is a vertex cover of size 2.

when the LP constraints force a unique ordering for every rule expansion. Given sufficiently strong constraints, Shieber's parser reduces to Earley's as Shieber thought, but strong constraint represents the *best case* computationally rather than the *worst case*.

NP-COMPLETENESS

The worst-case time complexity of Shieber's algorithm is exponential in grammar size rather than quadratic as Shieber (1983:15) believed. Did Shieber choose a poor algorithm, or is ID/LP parsing inherently difficult? In fact, the simpler problem of *recognizing* sentences according to a UCFG is NP-complete. Thus, unless $\mathcal{P} = \mathcal{NP}$, no ID/LP parsing algorithm can always run in time polynomial in the combined size of grammar and input. The proof is a reduction of the *vertex cover* problem (Garey and Johnson, 1979:46), which involves finding a small set of vertices in a graph such that every edge of the graph has an endpoint in the set. Figure 1 gives a trivial example.

To make the parser decide whether the graph in Figure 1 has a vertex cover of size 2, take the vertex names a, b, c, and d as the alphabet. Take H_1 through H_4 as special symbols, one per edge; also take U and D as dummy symbols. Next, encode the edges of the graph: for instance, edge e_1 runs from a to c, so include the rules $H_1 \rightarrow a$ and $H_1 \rightarrow c$. Rules for the dummy symbols are also needed. Dummy symbol D will be used to soak up excess input symbols, so $D \rightarrow a$ through $D \rightarrow d$ should be rules. Dummy symbol U will also soak up excess input symbols, but U will be allowed to match only when there are four occurrences in a row of the same symbol (one occurrence for each edge). Take $U \rightarrow aaaa$, $U \rightarrow bbbb$, and $U \rightarrow cccc$, and $U \rightarrow dddd$ as the rules expanding U.

Now, what does it take for the graph to have a vertex cover of size $k = 2$? One way to get a vertex cover is to go through the list of edges and underline one endpoint of each edge. If the vertex cover is to be of size 2, the underlining must be done in such a way that only two distinct vertices are ever touched in the process. Alternatively, since there are 4 vertices in all, the vertex cover will be of size 2 if there are $4 - 2 = 2$ vertices left *untouched* in the underlining. This method of finding a vertex cover can be translated

$$START \rightarrow H_1H_2H_3H_4UUDDDD$$

$$\begin{array}{ll} H_1 \rightarrow a \mid c & U \rightarrow aaaa \mid bbbb \mid cccc \mid dddd \\ H_2 \rightarrow b \mid c & D \rightarrow a \mid b \mid c \mid d \\ H_3 \rightarrow c \mid d & \\ H_4 \rightarrow b \mid d & \end{array}$$

Figure 2: For $k = 2$, the construction described in the text transforms the vertex-cover problem of Figure 1 into this UCFG. A parse exists for the string $aaaabbbbccccdddd$ iff the graph in the previous figure has a vertex cover of size ≤ 2.

into an initial rule for the UCFG, as follows:

$$START \rightarrow H_1H_2H_3H_4UUDDDD$$

Each H-symbol will match one of the endpoints of the corresponding edge, each U-symbol will correspond to a vertex that was left untouched by the H-matching, and the D-symbols are just for bookkeeping. (Note that this is the only rule in the construction that makes essential use of the unordered nature of rule right-hand sides.) Figure 2 shows the complete grammar that encodes the vertex-cover problem of Figure 1.

To make all of this work properly, take

$$\sigma = aaaabbbbccccdddd$$

as the input string to be parsed. (For every vertex name x, include in σ a contiguous run of occurrences of x, one for each edge in the graph.) The grammar encodes the underlining procedure by requiring each H-symbol to match one of its endpoints in σ. Since the expansion of the $START$ rule is unordered, an H-symbol can match anywhere in σ, hence can match any vertex name (subject to interference from previously matched rules). Furthermore, since there is one occurrence of each vertex name for every edge, it's impossible to run out of vertex-name occurrences. The grammar will allow either endpoint of an edge to be "underlined" — that is, included in the vertex cover — so the parser must figure out which vertex cover to select. However, the grammar also requires two occurrences of U to match. U can only match four contiguous identical input symbols that have not been matched in any other way; thus if the parser chooses too large a vertex cover, the U-symbols will not match and the parse will fail. The proper number of D-symbols equals the length of the input string, minus the number of edges in the graph (to account for the H_i-matches), minus k times the number of edges (to account for the U-matches): in this case, $16 - 4 - (2 \cdot 4) = 4$, as illustrated in the $START$ rule.

The result of this construction is that in order to decide whether σ is in the language generated by the UCFG, the

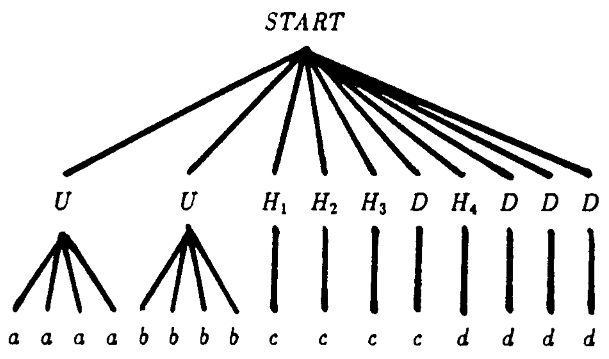

Figure 3: The grammar of Figure 2, which encodes the vertex-cover problem of Figure 1, generates the string $\sigma = aaaabbbbccccdddd$ according to this parse tree. The vertex cover $\{c, d\}$ can be read off from the parse tree as the set of elements dominated by H-symbols.

parser must search for a vertex cover of size 2 or less.[3] If a parse exists, an appropriate vertex cover can be read off from beneath the H-symbols in the parse tree; conversely, if an appropriate vertex cover exists, it shows how to construct a parse. Figure 3 shows the parse tree that encodes a solution to the vertex-cover problem of Figure 1. The construction thus reduces Vertex Cover to UCFG recognition, and since the construction can be carried out in polynomial time, it follows that UCFG recognition and the more general task of ID/LP parsing must be computationally difficult. For a more detailed treatment of the reduction, see Barton (1984b).

IMPLICATIONS

The reduction of Vertex Cover shows that the ID/LP parsing problem is NP-complete; unless $\mathcal{P} = \mathcal{NP}$, its time complexity is not bounded by any polynomial in the size of the grammar and input. Hence complexity analysis must be done carefully: despite similarity to Earley's algorithm, Shieber's algorithm does not have complexity $O(|G|^2 \cdot n^3)$, but can sometimes undergo exponential growth of its internal structures. Other computational and linguistic consequences also follow.

Although Shieber's parser sometimes blows up, it remains better than the alternative of parsing an expanded "object grammar." The NP-completeness result shows that the general case of ID/LP parsing is inherently difficult; hence it is not surprising that Shieber's ID/LP parser sometimes suffers from combinatorial explosion. It is more important to note that parsing with the expanded CFG blows up in *easy* cases. It should not be hard to parse the language that consists of all permutations of the string *abcde*, but in so doing, the Earley parser can use 24 states or more to encode what the Shieber parser encodes in only one (recall G_1). The significant fact is not that the Shieber parser can blow up; it is that the use of the object grammar blows up *unnecessarily*.

The construction that reduces the Vertex Cover problem to ID/LP Parsing involves a grammar and input string that *both* depend on the problem instance; hence it leaves it open that a clever programmer might concentrate most of the computational difficulty of ID/LP parsing into an offline grammar-precompilation stage independent of the input — under optimistic hopes, perhaps reducing the time required for parsing an input (after precompilation) to a polynomial function of grammar size and input length. Shieber's algorithm has no precompilation step,[4] so the present complexity results apply with full force; any possible precompilation phase remains hypothetical. Moreover, it is not clear that a clever precompilation step is even *possible*. For example, if n enters into the true complexity of ID/LP parsing as a *factor* multiplying an exponential, an input-independent precompilation phase cannot help enough to make the parsing phase always run in polynomial time. On a related note, suppose the precompilation step is conversion to CFG form and the runtime algorithm is the Earley parser. Although the precompilation step does a potentially exponential amount of work in producing G' from G, another exponential factor shows up at runtime because G' in the complexity bound $G'^2 n^3$ is exponentially larger than the original G.

The NP-completeness result would be strengthened if the reduction used the same grammar for all vertex-cover problems, for it would follow that precompilation could not bring runtime down to polynomial time. However, unless $\mathcal{P} = \mathcal{NP}$, there can be no such reduction. Since grammar size would not count as a parameter of a fixed-grammar ID/LP parsing problem, the use of the Earley parser on the object grammar would already constitute a polynomial-time algorithm for solving it. (See the next section for discussion.)

The Vertex Cover reduction also helps pin down the computational power of UCFGs. As G_1 and G'_1 illustrated, a UCFG (or an ID/LP grammar) is sometimes much smaller than an equivalent CFG. The NP-completeness result illuminates this property in three ways. First, the reduction shows that enough brevity is gained so that an instance of any problem in $\mathcal{NP}$ can be stated in a UCFG that is only polynomially larger than the original problem instance. In contrast, the current polynomial-time reduction could not be carried out with a CFG instead of a UCFG, since the necessity of spelling out all the orders in which symbols might appear could make the CFG exponentially larger than the instance. Second, the reduction shows that this brevity of expression is not free. CFG

[3]If the vertex cover is *smaller* than expected, the D-symbols will soak up the extra contiguous runs that could have been matched by more U-symbols.

[4]Shieber (1983:15 n. 6) mentions a possible precompilation step, but it is concerned with the LP relation rather than the ID rules.

recognition can be solved in cubic time or less, but unless $\mathcal{P} = \mathcal{NP}$, general UCFG recognition cannot be solved in polynomial time. Third, the reduction shows that only *one essential use* of the power to permute rule expansions is necessary to make the parsing problem NP-complete, though the rule in question may need to be arbitrarily long.

Finally, the ID/LP parsing problem illustrates how weakness of constraint can make a problem computationally difficult. One might perhaps think that *weak* constraints would make a problem easier since weak constraints sound easy to verify, but it often takes *strong* constraints to reduce the number of possibilities that an algorithm must consider. In the present case, the removal of constraints on constituent order causes the dependence of the runtime bound on grammar size to grow from $|G|^2$ to $2^{|G|}$.

The key factors that cause difficulty in ID/LP parsing are familiar to linguistic theory. GB-theory and GPSG both permit the existence of constituents that are empty on the surface, and thus in principle they both allow the kind of pathology illustrated by G_3, subject to amelioration by additional constraints. Similarly, every current theory acknowledges lexical ambiguity, a key ingredient of the vertex-cover reduction. Though the reduction illuminates the power of certain mechanisms and formal devices, the direct implications of the NP-completeness result for grammatical theory are few.

The reduction does expose the weakness of attempts to link context-free generative power directly to efficient parsability. Consider, for instance, Gazdar's (1981:155) claim that the use of a formalism with only context-free power can help explain the rapidity of human sentence processing:

> Suppose ... that the permitted class of generative grammars constituted a subset of those phrase structure grammars capable only of generating context-free languages. Such a move would have two important metatheoretical consequences, one having to do with learnability, the other with processability ... We would have the beginnings of an explanation for the obvious, but largely ignored, fact that humans process the utterances they hear very rapidly. Sentences of a context-free language are provably parsable in a time that is proportional to the cube of the length of the sentence or less.

As previously remarked, the use of Earley's algorithm on the expanded object grammar constitutes a parsing method for the fixed-grammar ID/LP parsing problem that is indeed no worse than cubic in sentence length. However, the most important aspect of this possibility is that it is devoid of practical significance. The object grammar could contain trillions of rules in practical cases (Shieber, 1983:4). If $|G'|^2 \cdot n^3$ complexity is too slow, then it remains too slow when $|G'|^2$ is regarded as a constant. Thus it is impossible to sustain this particular argument for the advantages of such formalisms as GPSG over other linguistic theories; instead, GPSG and other modern theories seem to be (very roughly) in the same boat with respect to complexity. In such a situation, the linguistic merits of various theories are more important than complexity results. (See Berwick (1982), Berwick and Weinberg (1984), and Ristad (1985) for further discussion.)

The reduction does not rule out the use of formalisms that decouple ID and LP constraints; note that Shieber's direct parsing algorithm wins out over the use of the object grammar. However, if we assume that natural languages are efficiently parsable (EP), then computational difficulties in parsing a formalism *do* indicate that the formalism itself *fails to capture* whatever constraints are responsible for making natural languages EP. If the *linguistically relevant* ID/LP grammars are EP but the *general* ID/LP grammars are not, there must be additional factors that guarantee, say, a certain amount of constraint from the LP relation.[5] (Constraints beyond the bare ID/LP formalism are required on linguistic grounds as well.) The *subset principle* of language acquisition (*cf.* Berwick and Weinberg, 1984:233) would lead the language learner to initially hypothesize strong order constraints, to be weakened only in response to positive evidence.

However, there are other potential ways to guarantee that languages will be EP. It is possible that the principles of grammatical theory permit languages that are not EP in the worst case, just as grammatical theory allows sentences that are deeply center-embedded (Miller and Chomsky, 1963). Difficult languages or sentences still would not turn up in general use, precisely *because* they would be difficult to process.[6] The factors making languages EP would not be part of grammatical theory because they would represent extragrammatical factors, *i.e.* the resource limitations of the language-processing mechanisms. In the same way, the limitations of language-acquisition mechanisms might make hard-to-parse languages *inaccessible* to the language learner in spite of satisfying grammatical constraints. However, these "easy explanations" are not tenable without a detailed account of processing mechanisms; correct predictions are necessary about *which constructions* will be easy to parse.

ACKNOWLEDGEMENTS

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of

[5] In the GB-framework of Chomsky (1981), for instance, the syntactic expression of unordered θ-grids at the $\bar{X}$ level is constrained by the principles of Case theory. Endocentricity is another significant constraint. See also Berwick's (1982) discussion of constraints that could be placed on another grammatical formalism — lexical-functional grammar - to avoid a similar intractability result.

[6] It is often anecdotally remarked that languages that allow relatively free word order tend to make heavy use of inflections. A rich inflectional system can supply parsing constraints that make up for the lack of ordering constraints; thus the situation we do *not* find is the computationally difficult case of weak constraint.

Technology. Support for the Laboratory's artificial intelligence research has been provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505. During a portion of this research the author's graduate studies were supported by the Fannie and John Hertz Foundation. Useful guidance and commentary during this research were provided by Bob Berwick, Michael Sipser, and Joyce Friedman.

REFERENCES

Barton, E. (1984a). "Toward a Principle-Based Parser," A.I. Memo No. 788, M.I.T. Artificial Intelligence Laboratory, Cambridge, Mass.

Barton, E. (1984b). "On the Complexity of ID/LP Parsing," A.I. Memo No. 812, M.I.T. Artificial Intelligence Laboratory, Cambridge, Mass.

Berwick, R. (1982). "Computational Complexity and Lexical-Functional Grammar," *American Journal of Computational Linguistics* 8.3-4:97–109.

Berwick, R., and A. Weinberg (1984). *The Grammatical Basis of Linguistic Performance*. Cambridge, Mass.: M.I.T. Press.

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

Earley, J. (1970). "An Efficient Context-Free Parsing Algorithm," *Comm. ACM* 13.2:94–102.

Garey, M., and D. Johnson (1979). *Computers and Intractability*. San Francisco: W. H. Freeman and Co.

Gazdar, Gerald (1981). "Unbounded Dependencies and Coordinate Structure," *Linguistic Inquiry* 12.2:155–184.

Miller, G., and N. Chomsky (1963). "Finitary Models of Language Users," in R. D. Luce, R. R. Bush, and E. Galanter, eds., *Handbook of Mathematical Psychology*, vol. II, 419–492. New York: John Wiley and Sons, Inc.

Ristad, E. (1985). "GPSG-Recognition is NP-Hard," A.I. Memo No. 837, M.I.T. Artificial Intelligence Laboratory, Cambridge, Mass., forthcoming.

Shieber, S. (1983). "Direct Parsing of ID/LP Grammars." Technical Report 291R, SRI International, Menlo Park, California. Also appears in *Linguistics and Philosophy* 7:2.