Contents

Chapter 1

Introduction

This thesis presents a parser that operates rapidly over large quantities of unrestricted text. The parser is currently used on the Lancaster/Oslo-Bergen corpus, a million word database of unedited English text. Rather than attempt to provide full parses of all the sentences in the corpus, the parser instead breaks the text into the largest chunks it can reliably parse. This not only increases the speed and accuracy of the parser, it also allows for selective parsing and reduced output ambiguity. The parser

- is the fastest general purpose parser known, with .04 seconds per word parsing time;
- outputs only one parse, in sharp contrast to the hundreds of parses many current parsers produce;
- can be applied to arbitrary text, giving researchers access to far greater linguistic data;
- parses substructures of complex phrases that might otherwise be difficult to analyze;
- can be configured to parse only limited substructures, for data collection purposes;
- can modify its output to conform to any number of representations;
- essentially solves the problem of accurate part-of-speech disambiguation.

1.1 Some Failings of Standard Parsing Systems

Although a great number of new parsing algorithms are proposed each year, and various new linguistic theories [1] have expressly concerned themselves with questions of efficiency and practicality, there are still a number of basic parsing problems which have eluded researchers. Many of them are rooted in the difficulties of non-deterministic parsing, or parsing in which more than one parse is explored and, often, output.

The lure of non-determinism in parsing is that if one can examine all possible legal parses for a given sentence under some grammar, then there is no possibility of failing to produce the ideal one (defined, presumably, to be the one which most seems to agree with a human's natural interpretation of the sentence), assuming the grammar is broad enough to encompass the correct parse. The two primary drawbacks are that

- producing all the valid parses for a sentence under a broad grammar can be a computationally expensive task, especially as the grammar grows to include all of the sentences found in completely unrestricted text, and
- large grammars often admit a great number of legal parses for a given sentence, and any parser that returns all of these parses may be passing on an almost insurmountable disambiguation task.

The first drawback is slowly becoming a moot point in practical natural language processing as computers become ever more powerful, though few general text parsers are currently even close to being real-time. The second drawback is a far more fundamental and pressing problem, and one that unfortunately, too few have chosen to address[2]. Many supposedly efficient parsers return not one, but many hundreds of parse trees for a given input sentence (for a particular example, see [Harrison 1988], described in more detail in section 4.4.1 and Appendix E). All of these parse trees may be admissible under the grammar utilized, but only a very small number (usually one) are plausible interpretations of the sentence. The task of choosing the correct parse tree has been skirted by the parser, generally under the assumption that it is an impossible task without semantic information unavailable to current implementations. However, given that such information is unavailable, one must question the usefulness of any parser which returns large numbers of undifferentiated results.

[1] Generalized Phrase Structure Grammar, in particular. See [Gazdar *et al* 1985] for an exposition.

[2] [Marcus 1980] being a pointed exception.

```
The assembly language provides a means for writing a program without having
to be concerned with actual memory addresses .

> (IP (NP (DET "The") (N "assembly") (N "language"))
      (I-BAR (I-0)
             (VP (V "provides") (NP (DET "a") (N "means"))
                 (PP (P "for")
                     (NP (I-BAR (I-0) (VP (V "writing")
                                          (NP (DET "a") (N "program"))))))))
> (P "without")
> (I-BAR (I-0) (VP (HAVE "having")))
> (I-BAR (I-0 (TO "to") (IS "be")) (VP (V "concerned")))
> (PP (P "with") (NP (AP (JJ "actual")) (N "memory") (N "addresses")))
> (*PER ".")
```

Figure 1-1: Harrison's parser produced 1560 distinct parse trees for the above sentence in 4.41 seconds, which is considered quite efficient. Our parser took only .18 seconds to produce this single analysis.

1.2 The Solution Presented

The issues of efficiency and determinism discussed in the preceding section are addressed and, to a large extent, solved in this thesis. For example, a sentence taken from the relatively efficient parser described in Harrison's thesis (a representative contemporary parser) is parsed in Figure 1-1.

In the system presented, before parsing begins a lexical disambiguator is utilized to determine the part of speech of each input word. Previously this has been suggested and experimented with, but has rarely been practical because any error introduced by a disambiguator can cause a parser to fail completely. Chapter 3 describes a statistical part of speech disambiguator which returns multiple parts of speech per word on occasion, greatly increasing disambiguation accuracy.

The parser described in Chapter 4 then uses a combination of top-down and bottom-up methods to rapidly parse a sentence into the largest structures it can. Its ability to build the constituent phrases of a large structure which might otherwise be left unanalyzed is a necessity, given the complexity of the free text found in the LOB and other text databases. An innovative method of introducing determinism, called "lowering", increases the efficiency and accuracy of parses. The parser outputs a single set of uniquely parsed phrases for each sentence, that usually match human interpretations extremely closely despite the lack of semantic constraints.

Other advantages of the parser include its ability to selectively parse certain linguistic structures, which allows efficient gathering of certain data from text, and the ability to modify the output representation of the parser without fundamentally changing the grammar. It is the first parser of its kind sufficiently fast that million word texts such as the LOB parser have become available for practical use; the entire LOB can be parsed in about 11 hours on a Symbolics Lisp Machine.

1.3 Thesis Organization

The rest of the thesis is divided into four chapters and six appendices. Chapter 2 briefly introduces the LOB corpus. Chapter 3 covers statistical disambiguation, and Chapter 4 details the parser. Finally, Chapter 5 presents conclusions and directions for further research. Appendices A through F contain a description of the LOB part of speech tagging system; a parser grammar used in tests; LISP code generated automatically by the parser from the grammar; various graphs of parser performance; sample parses from the LOB; and sample disambiguations from the LOB.

Chapter 2

The LOB Corpus

2.1 Introduction

The Lancaster/Oslo-Bergen Corpus is an on-line collection of more than 1,000,000 words of English text taken from a variety of sources, including British newspaper articles, novels, speeches, and interviews. The text is broken up into sentences that are often 50 or more words long. Each word in the LOB has been manually tagged with a part of speech taken from a 136 category system. Appendix A describes the tagging system in more detail. Figure 2-1 contains a typical LOB sentence

2.2 Gathering Statistics from the LOB

The LOB corpus has been used in a standard way to build several statistical tables of part of speech usage. Foremost is a dictionary keying every word found in the corpus to the number of times it is used as a certain part of speech. Sample entries are given in Figure 2-2.

The dictionary allows one to compute the probability that a word takes on a given part

```
Sir/NPT Roy/NP is/BEZ violently/RB opposed/VBN to/IN Africans/NNPS
getting/VBG an/AT elected/JJ majority/NN in/IN Northern/NP Rhodesia/NP
,/, but/CC the/ATI colonial/JJ Secretary/NPT ,/, Mr/NPT Iain/NP
Macleod/NP ,/, is/BEZ insisting/VBG on/IN a/AT policy/NN of/IN change/NN
./.
```

Figure 2-1: A tagged LOB sentence, as encoded in the database.

Word	Dictionary Entry
a	(AT 21744) (AP 431) (ZZ 216) (RB 159) (JJ 3) (IN 1)
book	(NN 300) (VB 4)
in	(IN 20335) (RP 473) (RB 212) (TO 105) (CS 47) (&FW 3) (NN 2) (NC 1)
read	(VB 104) (VBN 61) (VBD 53) (NN 1) (NP 1)
that	(CS 7513) (DT 2521) (WPR 1351) (QL 15)
the	(ATI 68233) (RB 23) (NC 2)

Figure 2-2: Some entries in the Dictionary

of speech. Given the information in Figure 2-2, the probability that the word "read" is a past-tense verb (VBD) is 53/220, or .241. In addition to the construction of the dictionary, the number of times each part of speech occurred in the corpus was recorded, and a digram array was built, listing the number of times any given part of speech was followed by any other. Since the category NN appears 148,129 times in the entire LOB corpus and the sequence ATI NN appears 35,139 times, the probability of an ATI preceding a NN is .237.

2.3 LOB Shortcomings

The LOB corpus contains approximately 50,000 sentences and 40,000 different words, many of which are proper nouns. Given that an average dictionary might have entries for 80,000 words, it is not surprising that many seemingly common words are either not found in the LOB, or only occur once or twice. Thus, some dictionary entries are either incorrect or statistically misleading. The limited size of the corpus also prevents some potentially useful data from being gathered. For instance, some statistical disambiguation schemes (see [Church 1988] for example) require knowing the number of trigram (three category sequences) occurrences. Unfortunately, with a 136 category system and only one million words of tagged text, the statistical accuracy of LOB trigrams would be minimal because most entries would never occur. Indeed, even in the LOB digram table fewer than 3200 of the 18,500 digrams occur more than 10 times.

Another, more significant problem with the LOB and most other tagged text databases is inconsistency in the part of speech tags. Many seemingly unambiguous words are at rare intervals assigned very unusual tags (see "a" and "the" in Figure 2-2). This data noise can have a significant affect on statistical disambiguators that learn from the tags. It also makes the checking of disambiguations against the tags difficult. The problem occurs most commonly in verb argument attachment situations. For instance, in the sequence "walked out", "out" is sometimes tagged as a preposition indicating that it is being interpreted as a verb argument, while in other cases it is tagged as a particle (`RP`), indicating that it is being interpreted as part of the verb. When running a parser directly on the tags, this kind of tagging difference is significant.[1] See Appendix A for other LOB deficiencies.

[1]Indeed, the greater consistency provided by statistical disambiguators causes our parser to often perform better when not using the LOB pre-tags and performing disambiguation.

Chapter 3

Statistical Disambiguation

3.1 Introduction

Most common English words can take on more than one part of speech (in the LOB, the average rate is 2.3 categories per word), presenting parsers with an added source of ambiguity. Furthermore, since most parsers expend resources in proportion to the total number of parses they produce, and since this number can be an exponential function of the number of parts of speech per word, lexical ambiguity can be extremely expensive. Most researchers have made the assumption that lexical disambiguation is fundamentally easier than parsing, and hence that it is advantageous to disambiguate separately with a preprocessor to the parser.

In many situations extragrammatical clues such as voice tone and semantic context are available, and may help to resolve lexical ambiguity, but in the isolated context of parsing independent sentences from a corpus, disambiguation of words must be based on knowledge about individual words and their ability to appear in a given syntactic or lexical context.

One common style of lexical disambiguators, used here, rely on statistical cooccurrence information such as that described in Chapter 2. Statistical disambiguation is surprisingly successful, but like all methods it fails to perfectly disambiguate all words in all sentences. Thus, any parser that relies on a standard statistical disambiguator to preprocess sentences and remove lexical disambiguity will fail regularly (Figure 3-2 gives some indication how often). A statistical disambiguator is presented which acknowledges imperfection by returning multiple parts of speech per word in certain cases, and is compared to other statistical methods applied to the LOB corpus. It performs substantially more accurately, with only one error made every 1340 words with an average of 1.27 parts of speech returned per word.

3.2 Specific Schemes

Specific statistical disambiguators are described in both [DeRose 1988] and [Church 1988]. They can be thought of as algorithms that maximize a function over the possible selections of categories. For instance, for each word A^z in a sentence, the DeRose algorithm takes a set of categories $\{a_1^z, a_2^z, \ldots\}$ as input. It outputs a particular category $a_{i_z}^z$ such that the product of the probability that A^z is the category $a_{i_z}^z$ and the probability that the category $a_{i_z}^z$ occurs before the category $a_{i_{z+1}}^{z+1}$ is maximized. Applying his algorithm to the Brown Corpus[1], DeRose reports the accuracy rate of 96% of words correctly disambiguated. [Leech *et al* 1983] quotes an accuracy of between 96% and 97% in applying an algorithm similar to DeRose's to the LOB.

A number of assumptions about language are embodied in DeRose's and other schemes. The first is that for disambiguation purposes, language can be viewed as a stochastic process, *i.e.*, one with a limited memory. Thus, the probability of a category occurring in a given sentential position is dependent only on the immediate surroundings and not on the sentence as a whole. It is this in particular that makes statistical disambiguation a fundamentally easier problem than parsing. Secondly, it is assumed that word cooccurrence restrictions are the result only of their parts of speech and not other lexical properties. So, the likelihood that "turn" in "turn off" is a verb is a function of verbs' statistical propensity to be followed by prepositions, not any semantics specific to "turn". Thirdly, it is largely assumed that cooccurrence information averaged from a large corpus is usefully applicable to small sections of text. Given the reasonable success of current statistical disambiguation schemes, these assumptions seem to be born out.

Four statistical schemes are here applied to the LOB corpus. Each is described below. From here on, accuracy figures will be presented in terms of how often words are incorrectly disambiguated. Thus, 96% correctness is written as an accuracy of 25 words per error, and a 20% increase in accuracy refers to 20% more words disambiguated for every error made.

[1] The Brown Corpus is also a 1,000,000 word text database, with a catgory system quite similar to that used in the LOB.

3.2.1 Method A

Method A is also described in the DeRose paper. It optimizes the product of the probabilities of each category occurring before the next, or

$$\max \prod_{z=1}^{n-1} \mathrm{P}\left(a^z_{i_z} \text{is-followed-by } a^{z+1}_{i_{z+1}}\right)$$

3.2.2 Method B

Method B is the other half of the DeRose scheme, optimizing the product of the probabilities of each category occurring for its word. Method B simply selects each word's most probable category, regardless of context.

$$\max \prod_{z=1}^{n} \mathrm{P}\left(A^z \text{is-category } a^z_{i_z}\right)$$

3.2.3 Method C

The DeRose scheme, or

$$\max \prod_{z=1}^{n} \mathrm{P}\left(A^z \text{is-category } a^z_{i_z}\right) \prod_{z=1}^{n-1} \mathrm{P}\left(a^z_{i_z} \text{is-followed-by } a^{z+1}_{i_{z+1}}\right)$$

3.2.4 Method D

No statistical disambiguator can perform perfectly if it only returns one part of speech per word, because there are words and sequences of words that can be truly ambiguous in certain contexts. Method D addresses this problem by on occasion returning more than one category per word.

The algorithm presented in [DeRose 1988] moves from left to right assigning to each category a^z_j an optimal path of categories leading from the start of the sentence to a^z_j, and a corresponding probability. It then extends each path with the categories of the word A^{z+1} and computes new probabilities for the new paths. Call the greatest new probability P. Method D assigns to the word A^z those categories $\{a^z_i\}$ that occur in those new paths that have a probability within a factor F of P. Hence, Method D utilizes a one word lookahead to disambiguate words to a set of parts of speech that seem statistically probable at the time of disambiguation, where "probable" is interpreted relative to the most likely part of speech.

Naturally, Method D will return several categories for some words, and only one for others, depending on the particular sentence and the factor F. If $F = 1$, Method D will return only one category per word, but they are not necessarily the same categories as DeRose would return. A more obvious variation of DeRose, in which alternate categories are substituted into the DeRose disambiguation and accepted if they do not reduce the overall disambiguation probability significantly, would approach DeRose as F went to 1, but turns out not to perform as well as Method D. To be more precise, for a given average number of parts of speech returned V, the "substitution" method is about 10% less accurate when $1 \leq V < 1.1$ and is almost 50% less accurate for $1.1 \leq V < 1.2$.

3.3 Disambiguator Results

Each method was applied to the same 64,000 words of the LOB corpus. The results were compared to the LOB part of speech pre-tags, and are listed in Figure 3-1.[2] If a word was pre-tagged as being a proper noun, the proper noun category was included in the dictionary, but no special information such as capitalization was used to distinguish that category from others during disambiguation. For that reason, when judging accuracy, we provide two metrics: one simply comparing disambiguator output with the pre-tags, and another that gives the disambiguator the benefit of the doubt on proper nouns, under the assumption that an "oracle" pre-processor could distinguish proper nouns from contextual or capitalization information. Since Method D can return several categories for each word, we provide the average number of categories per word returned, and we also note the setting of the parameter F, which determines how many categories, on average, are returned. Individual examples of sentences disambiguated with Method D, $F = .01$ can be found in Appendix F.

The numbers in Figure 3-1 show that simple statistical schemes can accurately disambiguate parts of speech in normal text, confirming DeRose and others. The reason that statistical schemes achieve such accuracy should be clear from the results of Method B. By

[2]In all figures quoted, punctuation marks have been counted as words. Since punctuation marks account for 10.5% of the text analyzed, not including them would increase the average number of parts of speech listed per word to 2.32, and accuracy measurements would be reduced by 10.5%. Average numbers of categories returned by Method D would increase by 1-3%. Punctuation marks are treated as parts of speech by the statistical disambiguators.

Method:	A	B	C	D(1)	D(.3)	D(.1)	D(.03)	D(.01)	D(.003)
Accuracy:	7.9	17	23	25	41	70	126	265	1340
with oracle:	8.8	18	30	31	54	105	230	575	1840
No. of Categories:	1	1	1	1	1.04	1.09	1.14	1.20	1.27

Figure 3-1: Accuracy of various disambiguation strategies, in number of words per error. On average, the dictionary had 2.18 parts of speech listed for each of the 64,000 words.

simply using the part of speech a given word takes on most often, a disambiguator can be right almost 95% of the time. Clearly, if language is this regular then statistical methods are a good start.[3] The extraordinary accuracy one can achieve by accepting an additional category every several words indicates that disambiguators can predict when their answers are unreliable. Figure 3-2 graphs the statistical accuracy necessary to, with a given probability, correctly disambiguate every word of a sentence.

Readers may worry about correlation resulting from using the same corpus to both learn from and disambiguate. We have run tests by first learning from half of the LOB (600,000 words) and then disambiguating 80,000 words of random text from the other half. The accuracy figures varied by less than 5% from the ones we present, which, given the size of the LOB, is to be expected. We have also applied each disambiguation method to several smaller (13,000 word) sets of sentences that were selected at complete random from throughout the LOB. Accuracy varied both up and down from the figures we present, by up to 20% in terms of words per error, but relative accuracy between methods remained constant.

3.4 Multivalued Disambiguation and Parsing

Method D achieves its extraordinary accuracy at the expense of not performing a full disambiguation. For some applications, this is self-defeating. For instance, a parser that can only accept one part of speech per word would find Method D useless as a preprocessor. On the other hand, some disambiguation is performed, and this may be sufficient to greatly increase both the efficiency and the accuracy of a parser. In a non-deterministic parser, any reduction in the number of categories taken as input substantially decreases the time and

[3] One might wonder whether this is true for languages other than English; this is obviously difficult to test without tagged corpuses being available. Most fixed word-order languages should give similar results. Free word-order languages are likely to prove more difficult to analyze statistically, but most of these languages provide significantly more morphological clues to disambiguate with.

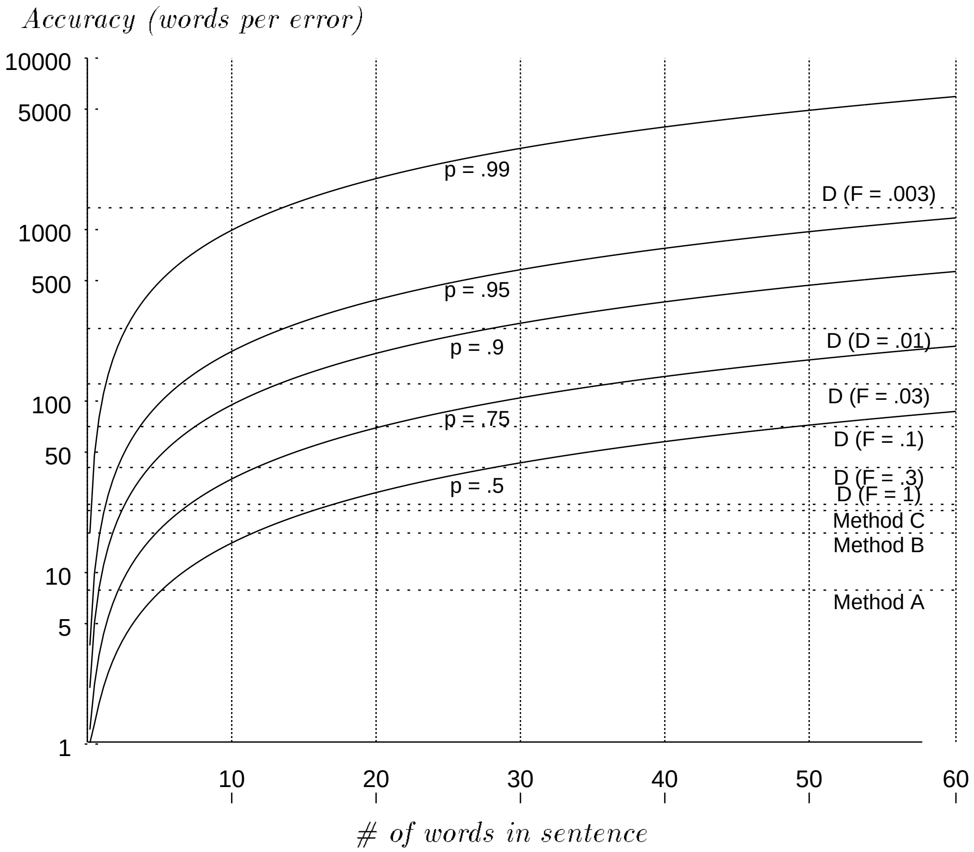


Figure 3-2: A graph of the accuracy necessary to, with probability p, correctly disambiguate an entire sentence of a given length, for various p. Also graphed are the accuracies of various disambiguation strategies.

space the parser uses, and more importantly, can reduce the number of parses output. Given the propensity of large grammars to produce huge numbers of parses for small sentences, this is important[4]. And in a deterministic or semi-deterministic parser, the elimination of input categories can be a great step in increasing overall accuracy, since it reduces the number of irreversible decisions the parser must make.

Disambiguators that do not achieve a sufficient level of accuracy, however, can be more harmful than helpful. As can be seen from Figure 3-2, on a common 20 word sentence DeRose's scheme (Method C) will return a completely correct disambiguation less than half the time. So, any parser that attempts to evaluate the sentence based on the assumption that the entire sentence is grammatical (almost all modern parsers) may often fail. Thus,

[4] See [Harrison 1988] for example. This area is discussed further in Chapter 4, section 4.4.1

even though DeRose's scheme performs a more comprehensive disambiguation than Method D might, it will quite likely be less useful.

3.5 Some Implementation Details

Given that the described disambiguators are being applied to large sections of unrestricted text, some efficiency and robustness issues arise. Importantly, many words are not in the statistical dictionary and many digram entries list as 0 occurrences. Naturally, when computing probability products, any multiplication by 0 leads to disaster because it reduces global probabilities to the result of a single local action. Consequently, any digram entry listed as having never occurred in the LOB is treated as having occurred once. Furthermore, any capitalized word not appearing in the dictionary is assumed to be a proper noun, a reasonable assumption given the large size of the dictionary. Other unknown words can be morphologically analyzed in an attempt to guess their possible parts of speech.

It may at first glance appear that the four statistical disambiguation schemes presented are exponential time in sentence length, because there are an exponential number of possible category selections. However, the limited leftward and rightward dependencies of the algorithm allow for linear time dynamic programming methods. Figure 3-3 presents an algorithm that implements Method D in linear time.

3.6 Digrams vs. Trigrams

The fact the Method D with $F = 1$ (with $F = 1$ Method D returns only one category per word) performs as well or even better on the LOB than DeRose's algorithm indicates that, with exceptions, disambiguation has very limited rightward dependence: Method D essentially employs a one category lookahead, whereas DeRose's algorithm looks to the end of the sentence. This suggests that Church's strategy of using trigrams instead of digrams may be wasteful. Church manages to achieve results similar or slightly better than DeRose's by defining the probability that a category A appears in a sequence ABC to be the number of times the sequence ABC appears divided by the number of times the sequence BC appears. In a 100 category system, this scheme requires an enormous table of data, which must be culled from tagged text. For example, the sequence `c1 noun c2` occurs in ordinary text with `c2` taking on just about any part of speech, and the category `c1` having very little

To disambiguate a sentence with words $A^1, \ldots, A^n$ and categories $a_1^1, \ldots, a_{m_1}^1; \ldots; a_1^n, \ldots, a_{m_n}^n$, where $N(A, a)$ is the number of times the word A occurred in the LOB with part of speech a, and $N(a_1, a_2)$ is the number of times the category a_1 was followed by the category a_2, and F is a sensitivity parameter:

- **Let** *Paths* **= nil.**
- **For** $i = 1$ **to** m_1
  - **Add** $(\{a_i^1\}; N(A^1, a_i^1))$ **to** *Paths*.
- **For** $j = 2$ **to** n
  - **Let** *Old-Paths* = *Paths*. **Set** *Paths* **= nil.**
  - **Let** *All-Paths* **= nil.**
  - **Let** $P = 0$.
  - **For** $i = 1$ **to** m_j
    - **Let** *New-Paths* $_i$ **= nil.**
    - **Let** *Best-New-Path* $_i$ **= nil. Let** $Q_i = 0$.
    - **For each** $(\{b, c, \ldots\}; p)$ **in** *Old-Paths*
      - **Let** $q = p \cdot N(A^j, a_i^j) \cdot N(b, a_i^j)$.
      - **Let** *New-Path* $= (\{a_i^j, b, c, \ldots\}, q)$.
      - **Add** *New-Path* **to** *New-Paths* $_i$.
      - **Add** *New-Path* **to** *All-Paths*.
      - **If** $q > Q_i$
        - **Set** $Q_i = q$.
        - **Set** *Best-New-Path* $_i$ = *New-Path*.
      - **If** $q > P$
        - **Set** $P = q$.
    - **Add** *Best-Path* $_i$ **to** *Paths*.
  - **Let** *Output-Cats* **= nil.**
  - **For each** $(\{a, b, \ldots\}; p)$ **in** *All-Paths*
    - **If** $p > FP$ **and** b **is not in** *Output-Cats*
      - **Add** b **to** *Output-Cats*.
  - **Return** *Output-Cats* **as disambiguation of** A^j.
- **Let** *Output-Cats* **= nil.**
- **For each** $(\{a, b, \ldots\}; p)$ **in** *All-Paths*
  - **If** $p \geq FP$ **and** a **is not in** *Output-Cats*
    - **Add** a **to** *Output-Cats*.
- **Return** *Output-Cats* **as disambiguation of** A^n.

Figure 3-3: A Pseudo-Code Algorithm for Method D

statistical correlation to `c2`. Therefore in disambiguating `c1`, the only significant impact of using trigrams is to greatly reduce the number of data points available for comparison (and therefore the statistical significance of any decision). Thus, to get equal statistical accuracy a trigram-based disambiguator needs far more learning text. If the rightward dependence of disambiguation is small, as the data suggests, then the extra effort needed for trigrams may be for naught. Based on these results, it is more efficient to use digrams in general and only mark special cases for trigrams, which would reduce space and learning requirements substantially.

This discussion brings up a basic question about the way words and category sequences are weighted during disambiguation. Currently, all of the algorithms presented weigh words with category distribution `((NN 2) (VB 4))` the same as words with distribution `((NN 20) (VB 40))`, despite the greater statistical significance of the second distribution. Given the ease with which the algorithms could be modified to weigh data by its statistical reliablity, would it not be a better idea to do so? Although tests of this idea have not yet been performed, they are a high priority.

3.7 The Performance of Method D

Appendix F presents the disambiguation of 20 sentences, selected completely randomly from throughout the LOB corpus. All sentences are disambiguate with Method D, $F = 1$, and no use of capitalization information. With these parameters, the data of Figure 3-1 suggests that there should be an error approximately every 265 words. Since 265 words are disambiguated, about one error is expected. Two errors are found. On average, 1.19 categories are returned for each word, closely matching Figure 3-1.

The two errors occur in sentences 7 and 12. In sentence 7, the word "a" in ".. just a little louder than was expected" is disambiguated properly to `AT` (determiner), but the pretag provided is `RB` (adverb). Obviously the disambiguator functioned properly but the LOB was mismarked. In sentence 12, the word "who" in ".. at Mrs. Batley who was now .." was disambiguated to both `CS` (complementizer) and `WP` (wh-pronoun) instead of the tagged `WPR` (relative wh-pronoun). The distinction is a fine one, and indeed the current grammar maps both `WP` and `WPR` to the same part of speech (see Appendix A). An error perhaps, but an extremely minor one.

In sentences 1,3,5,7,16,17 and 20 there are a total of ten cases where the disambiguator returns the correct category in a list of several categories but not as the first (most likely) one. These are cases where if F was reduced in an attempt to reduce the average number of categories returned, it is possible that more errors would occur.

Chapter 4

The Parser

4.1 Introduction

The LOB corpus contains unedited English, some of which is quite complex and some of which is ungrammatical. No known parser could produce full parses of all the material, and even one powerful enough to do so would undoubtedly take an impractical length of time. To facilitate the analysis of the LOB, a new parser has been designed and implemented, that is capable of rapidly parsing simple constructs and of "failing gracefully" in more complicated situations. By trading completeness for accuracy, and by utilizing the previously described multi-valued statistical disambiguator, the parser can perform rapidly and correctly enough to usefully parse the entire LOB in a day (11 hours to parse, the remainder to load in sentences, store output, etc.). Figure 4-1 presents a sample parse from the LOB.

The parser uses three methods to build phrases. Context-free-grammar-like rules are used to recognize lengthy, less structured constructions such as noun phrases, names, dates, and verb systems. Neighboring phrases can connect to build the higher level binary-branching structure found in English, and single phrases can be projected into new ones. The ability of neighboring phrase pairs to initiate the CFG-like rules permits context-sensitive parsing. And, to increase the efficiency of the parser, an innovative system of deterministically discarding certain phrases is used, called "lowering".

```
MR MICHAEL FOOT HAS PUT DOWN A RESOLUTION ON THE SUBJECT AND HE IS TO BE BACKED
BY MR WILL GRIFFITHS , MP FOR MANCHESTER EXCHANGE .

==> (IP (NP (PROP (N MR) (NAME MICHAEL) (NAME FOOT)))
        (I-BAR (I (HAVE HAS) (RP DOWN))
                  (VP (V PUT) (NP (DET A) (N RESOLUTION)))))
                              (PP (P ON) (NP (DET THE) (N SUBJECT)))
==> (CC AND)
==> (IP (NP HE)
        (I-BAR (I)
           (VP (IS IS)
               (I-BAR (I (PP (P BY) (NP (PROP (N MR) (NAME WILL) (NAME GRIFFITHS))))
                      (TO TO) (IS BE)) (VP (V BACKED))))))
==> (*CMA ",")
==> (NP (N MP))
==> (PP (P FOR) (NP (PROP (NAME MANCHESTER) (NAME EXCHANGE))))
==> (*PER ".")
```

Figure 4-1: The parse of a sentence taken verbatim from the LOB corpus, printed without features. Notice that the grammar does not attach PP adjuncts.

4.2 Some Parser Details

Each word in an input sentence is tagged as starting and ending at a specific numerical location. In the sentence "I saw Mary" the parser would insert the locations 0-3, `0 I 1 SAW 2 MARY 3`. A phrase consists of a category, starting and ending locations, and a collection of feature and tree information. A verb phrase extending from 1 to 3 would print as `[VP 1 3]`. Rules consist of a state name and a location. If a verb phrase recognition rule was firing in location 1, it would get printed as `(VP0 at 1)` where `VP0` is the name of the rule state. Phrases and rules that have yet to be processed are placed on a queue. At parse initialization, phrases are created from each word and its category(ies), and placed on the queue along with an end-of-sentence marker. The parse proceeds by popping the top rule or phrase off the queue and performing actions on it. Figures 4-2 and 4-3 contain a detailed specification of the parser algorithm, along with parts of a grammar. It should be comprehensible after the following overview and parse example.

When a phrase is popped off the queue, rules are checked to see if they fire on it, a table is examined to see if the phrase automatically projects to another phrase or creates a rule, and neighboring phrases are examined in case they can pair with the popped phrase to either connect into a new phrase or create a rule. Thus the grammar consists of three tables, the "rule-action-table" that specifies what action a rule in a certain state should take if it encounters a phrase with a given category and features; a "single-phrase-action-table" that specifies whether a phrase with a given category and features should project or start a rule; and a "paired-phrase-action-table" that specifies possible actions to take if two certain phrases abut each other. For example, if the sentence "I walked home that day in April" is being parsed and the noun phrase "that day" is popped, several things might happen. If, as in the grammar in Appendix B, the single-phrase action table contains a specification for temporal noun phrases to start a verb phrase recognition rule (because noun phrases like "that day" can be used as adverbs) then a rule would be created to the left of "that". And, if "in April" has already been parsed into a prepositional phrase, a grammar might in the paired-phrase action table specify that "that day" and "in April" be combined into a single noun phrase. Lastly, a verb phrase recognition rule that has already fired on "walked" and "home" might fire on "that day" to complete a verb phrase.

For a rule to fire on a phrase, the rule must be at the starting position of the phrase. Possible actions that can be taken by the rule are: accepting the phrase (shift the dot in the rule); closing, or creating a phrase from all phrases accepted so far; or both, creating a phrase and continuing the rule to recognize a larger phrase should it exist. These actions are essentially identical to those most context-free parsers use. One important diffence is that when an enqueued phrase is accepted, it is "lowered" to the bottom of the queue, and when a rule closes to create a phrase, all other phrases it may have already created are lowered also. As will be seen this introduces an efficient determinism into the parser.

As phrases are created, calls can be made to transducer functions that generate more principled interpretations of the phrases, with appropriate features and tree relations. The representations they build are only for output, and do not affect the parse. An exception is made to allow the functions to project and modify features, which eases handling of subcategorization (verb argument structure). The transducer functions can be used to generate a constant output syntax as the internal grammar varies, and *vice versa*.

New phrases and rules are placed on the queue only after all actions resulting from a given pop of the queue have been taken. The ordering of their placement has a dramatic effect on how the parse proceeds. By varying the queuing placement and the definition of when a parse is finished, the efficiency and accuracy of the parser can be radically altered, forming an interesting test bed for experimentation. The parser orders new rules and phrases by placing rules first, and then pushes all of them onto the stack. This means that

To parse a sentences S of length n:

- Perform multivalued disambiguation of S.
- Create empty queue Q. Place End-of-Sentence marker on Q.
- Create new phrases from disambiguator output categories, and place them on Q.
- Until Q is empty, or top(Q) = End-of-Sentence marker,
  - Let I = pop(Q). Let *new-items* = nil
  - If I is phrase `[`*cat i j*`]`
    - Let *rules* = all rules at location i.
    - Let *lefts* = all phrases ending at location i.
    - Let *rights* = all-phrases starting at location j.
    - Perform `rule-actions(`*rules*`,`$\{I\}$`)`
    - Perform `paired-phrase-actions(`*lefts*`,`$\{I\}$`)`
    - Perform `paired-phrase-actions(`$\{I\}$`,`*rights*`)`
    - Perform `single-phrase-actions(`I`)`.
  - If I is rule `(`*state* `at` i`)`
    - Let *phrases* = all phrases starting at location i.
    - Perform `rule-actions(`$\{I\}$`,`*phrases*`)`.
  - Place each item in *new-items* on Q, rules first.
- Let $i = 0$. Until i = n,
  - Output longest phrase `[`*cat i j*`]`. Let $i = j$.

To perform `rule-actions(`*rules*`,`*phrases*`)`**:**

- For all rules R = `(`*state* `at` i`)` in *rules*,
  - And all phrases P = `[`*cat+features i j*`]` in *phrases*,
    - If there is an action A in the `rule-action-table` with key `(`*state, cat+features*`)`,
      - If A = `(accept` *new-state*`)`,
        - Create new rule `(`*new-state* `at` j`)`.
      - If A = `(close` *new-cat [tr-function]*`)`,
        - Let *daughters* = the set of all phrases that have been accepted in the rule chain which led to R, including the phrase P.
        - Let l = the leftmost starting location of any phrase in *daughters*. Create new phrase `[`*new-cat l j*`]` with daughters *daughters*.
        - For all phrases p in *daughters*, perform `lower(`p`)`.
        - For all phrases p created by the rule chain that led to R, perform `lower(`p`)`.
        - If a transducer-function *tr-function* is specified, call *tr-function* on `[`*new-cat l j*`]` and *daughters*.

Figure 4-2: Part 1 of a pseudo-code representation of the parser algorithm, omitting implementation and feature-handling details.

new rules will always have precedence over newly created phrases, and hence will fire in a successive "rule chain". For example, if a noun phrase recognition rule fires on "the" in "the car", any new rule generated to the left of "car" will be allowed to fire on "car" before

To perform `paired-phrase-actions(`*lefts*`,`*rights*`)`**:**

- For all phrases Pl = `[`*left-cat+features l i*`]` in *lefts*,
  - And all phrases Pr = `[`*right-cat+features i r*`]` in *rights*,
    - If there is an action A in the `paired-phrase-action- table` with key (*left-cat+features, right-cat+features*),
      - If A = `(connect` *new-cat [tr-function]*`)`,
        - Create new phrase `[`*new-cat l r*`]` with daughters Pl and Pr.
        - If a transducer-function *tr-function* is specified, call *tr-function* on `[`*new-cat l j*`]` and *daughters*.
      - If A = `(project` *new-cat*`)`,
        - Create new phrase `[`*new-cat i r*`]` with daughter Pr.
      - If A = `(start-new-rule` *state*`)`,
        - Create new rule `(`*state* `at` *i*`)`.
      - Perform `lower(`*Pl*`)` and `lower(`*Pr*`)`.

To perform `single-phrase-actions([`*cat+features i j*`])`**:**

- If there is an action A in the `single-phrase-action-table` with key *cat+features*,
  - If A = `(project` *new-cat*`)`,
    - Create new phrase `[`*new-cat i j*`]`.
  - If A = `(start-rule` *new-state*`)`,
    - Create new rule `(`*state* `at` *i*`)`.

To perform `lower(`*I*`)`**:**

- If I is in Q, remove it from Q and reinsert it at end of Q.
- If I is a lexical level phrase `[`*cat i i+1*`]` created from the disambiguator output categories,
  - For all other lexical level phrases p starting at i, perform `lower(`*p*`)`.

When creating a new rule R**:**

- Add R to list of *new-items*.

When creating a new phrase P = `[`*cat+features i j*`]` **with daughters** D**:**

- Add P to list of *new-items*.

Figure 4-3: Part 2 of a pseudo-code representation of the parser algorithm, omitting implementation details.

"car" is popped from the stack. Thus "car" can be accepted and therefore lowered before it is ever popped. If all items were eventually popped off the stack, the ordering of rules and phrases would be irrelevant. However, since the parse is stopped at the end-of-sentence marker, all phrases that have been "lowered" past the marker are never examined!

The part of speech disambiguator can pass in several categories for any one word, which are ordered on the stack by likelihood, most probable first. When any lexical phrase is lowered to the back of the queue (presumably because it was accepted by some rule) all other lexical phrases associated with the same word are also lowered. We have found that this both speeds up parsing and increases accuracy. That this speeds up parsing should be obvious. That it increases accuracy is much less so. Remember that disambiguation Method D is substantially more accurate than other algorithms only because it can return more than one category per word. One might guess that if the parser were to lower all extra categories on the queue, that nothing would have been gained. But the top-down nature of the parser is sufficient in most cases to "pick out" the correct category from the several available (see [Milne 1988] for a detailed example of this in connection with Marcus's [Marcus 1980] deterministic parser). Once a single category has been chosen, the removal of others from consideration frees the parser from using these other categories to produce incorrect phrases that might be returned instead of those containing the correct category.

4.3 A Parse in Detail

Figure 4-4 shows a parse of the sentence "The pastry chef placed the pie in the oven." In the figure, items to the left of the vertical line are the phrases and rules popped off the stack. To the right of each item is a list of all new items created as a result of it being popped. At the start of the parse, phrases were created from each word and their corresponding categories, which were correctly (and uniquely) determined by the disambiguator.

The first item is popped off the queue, this being the `[DET 0 1]` phrase corresponding to the word "the". The single-phrase action table indicates that a `DET0` rule should be started at location 0 and immediately fires on "the", which is accepted and the rule `(DET1 at 1)` is accordingly created and placed on the queue. This rule is then popped off the queue, and accepts the `[N 1 2]` corresponding to "pastry", also closing and creating the phrase `[NP 0 2]`. When this phrase is created, all queued phrases that contributed to it are lowered in priority, *i.e.*, "pastry". The rule `(DET2 at 2)` is created to recognize a possibly longer NP, and is popped off the queue in line 4. Here much the same thing happens as in line 3, except that the `[NP 0 2]` previously created is lowered as the phrase `[NP 0 3]` is created. In line 5, the rule chain keeps firing, but there are no phrases starting at location 3 that can be used by the rule state `DET2`.

The next item on the queue is the newly created `[NP 0 3]`, but it neither fires a rule

```
0 The 1 pastry 2 chef 3 placed 4 the 5 pie 6 in 7 the 8 oven 9 . 10

1.  Phrase  [DET 0 1]    | (DET0 at 0)
2.  Rule    (DET0 at 0)  | (DET1 at 1)
3.  Rule    (DET1 at 1)  | [NP 0 2]
                           (DET1 at 2)
                           Lowering: [N 1 2]
4.  Rule    (DET2 at 2)  | [NP 0 3]
                           (DET2 at 3)
                           Lowering: [NP 0 2]
                           Lowering: [N 2 3]
5.  Rule    (DET2 at 3)  |
6.  Phrase  [NP 0 3]     |
7.  Phrase  [V 3 4]      | (VP3 at 3)
8.  Rule    (VP3 at 3)   | [VP 3 4]
                           (VP4 at 4)
9.  Rule    (VP4 at 4)   |
10. Phrase  [VP 3 4]     |
11. Phrase  [DET 4 5]    | (DET0 at 4)
12. Phrase  (DET0 at 4)  | (DET1 at 5)
13. Rule    (DET1 at 5)  | [NP 4 6]
                           (DET2 at 6)
                           Lowering: [N 5 6]
14. Rule    (DET2 at 6)  |
15. Phrase  [NP 4 6]     | [VP 3 6]
16. Phrase  [VP 3 6]     | [S 0 6]
17. Phrase  [S 0 6]      |
18. Phrase  [P 6 7]      |
19. Phrase  [DET 7 8]    | (DET0 at 7)
20. Rule    (DET0 at 7)  | (DET1 at 8)
21. Rule    (DET1 at 8)  | [NP 7 9]
                           (DET2 at 9)
                           Lowering: [N 8 9]
22. Rule    (DET2 at 9)  |
23. Phrase  [NP 7 9]     | [PP 6 9]
24. Phrase  [PP 6 9]     |
25. Phrase  [*PER 9 10]  |

==> (IP (NP (DET "The") (N "pastry") (N "chef"))
        (I-BAR (I) (VP (V "placed") (NP (DET "the") (N "pie")))))
==> (PP (P "in") (NP (DET "the") (N "oven")))
==> (*PER ".")

Phrases left on Queue: [N 1 2] [N 2 3] [NP 0 2] [N 5 6] [N 8 9]
```

Figure 4-4: A detailed parse of the sentence “The pastry chef placed the pie in the oven”. Dictionary look-up and disambiguation were performed prior to the parse.

(which would have to be in location 0), finds any action in the single-phrase table, or pairs with any neighboring phrase to fire an action in the paired-phrase table, so no new phrases or rules are created. Hence, the verb “placed” is popped and the single-phrase table indicates that it should create a rule that then immediately accepts “placed”, creating a

VP and placing the rule `(VP4 at 4)` in location 4. The VP is popped off the stack, but not attached to `[NP 0 3]` to form a sentence, because the paired-phrase table specifies that for those two phrases to connect to become an `S`, the verb phrase must have the feature `(expect . nil)`, indicating that all of its argument positions have been filled. However when the VP was created, the VP transducer function call gave it the feature `(expect . NP)`, indicating that it is lacking an NP argument (a noun phrase representing the object to be "placed").

In line 15, such an argument is popped from the stack and pairs with the VP as specified in the paired-phrase table, creating a new phrase, `[VP 3 6]`. This new VP then pairs with the subject, forming `[S 0 6]`. In line 18, the preposition "in" is popped, but it does not create any rules or phrases. Only when the NP "the oven" is popped does it pair to create `[PP 6 9]`. Although it should be attached as an argument to the verb, the subcategorization frames (contained in the `expect` feature of the VP) do not allow for a prepositional phrase argument. After the period is popped in line 25, the end-of-sentence marker is popped and the parse stops. At this time, 5 phrases have been lowered and remain on the queue. To choose which phrases to output, the parser picks the longest phrase starting at location 0, and then the longest phrase starting where the first ended, etc.

4.4 The Reasoning behind the Methodology

The parser has a number of salient features to it, including the combination of top-down and bottom-up methods, the use of transducer functions to create tree structure, and the system of lowering phrases off the queue. Each was necessary to achieve sufficient flexibility and efficiency to parse the LOB corpus. It is worth exploring the necessity of such methods in light of other common approaches to parsing larges texts.

4.4.1 The Standard Approach

The primary difficulty that arises in analyzing unrestricted text is coping with the large number of different types of constructs found. Almost all parsers in use today employ individual rules or procedures for every phrase type, and work on the assumption that all sentences must be parsed in their entirety, *i.e.*, the grammar in use must describe a language that encompasses every sentence in the text to be analyzed.

Unfortunately, while it is relatively easy to add constructs to a grammar until all or most sentences are accepted, the size of the resulting grammar prevents a detailed understanding of how the constructs may interact, and in many cases the grammar becomes substantially ambiguous, usually in unintended ways. Thus, the addition of a single rule designed to accept a rare phrase may cause other seemingly unrelated phrases to be parsed ambiguously, with the result that simple sentences may often have hundreds or even thousands of legal parses. Furthermore, since parsers (or at least systems that use parsers as pre-processors) usually expend sizable resources to analyze the huge numbers of different parses, considerable energy must be spent optimizing code and algorithms if the parser is to be at all useful. A good example of this is found in [Harrison 1988]. Harrison's thesis presents a GPSG[1] parser with a grammar sufficiently large that the parser can be applied to a fairly broad range of text. An enormous effort was expended to optimize the parser to the point where it could produce parses of sentences fairly rapidly (usually less than 5 seconds). Unfortunately, even on the simplest of sentences the parser often produces more than a thousand different parses. It was largely this problem which prompted Marcus to introduce his Determinism Hypothesis in [Marcus 1980]. As an example, the first sentence in Harrison's set of examples, "The assembly language provides a means for writing a program without having to be concerned with actual memory addresses", results in over 1500 parses in 4.4 seconds. For comparison with the parser presented here, some of the parsed sentences in Appendix E are from Harrison's thesis.

In fact, one must question the usefulness of a natural language parser that returns inordinate numbers of parses for a single sentence. While a case might be made that it is a parser's job to produce all syntactically legal parse trees for a given grammar, such a parser is merely passing much of its computational work on to its successors. Some other device is going to have to perform a substantial amount of energy disambiguating between sentences, and performing such disambiguation is a non-trivial task, especially since few text analyzers can provide significant semantic context.

[1]Generalized Phrase Structure Grammar, or GPSG, is a popular and widely utilized linguistic theory that describes languages in terms of annotated context-free grammars and principles that generate and constrain production of new context-free rules from pre-existing ones. See [Gazdar *et al* 1985] for a detailed exposition of GPSG.

4.4.2 The Alternative: Limited Parsing

The alternative to fully parsing all sentences in the corpus is to limit the scope of parsing to the simpler and more common sentences and phrases of the text, as we have done, on the assumption that simple phrases are prevalent, and that for many applications a parser which does not parse an entire sentences can still be useful (almost as useful as one that parses a sentence fully, and more useful than one that parses a sentence fully into many different trees).

Since much of the incentive for parsing large texts such as the LOB lies in their richness as a source of lexical and syntactic data, most of which can be accessed from the parses of smaller phrases, a parser that does not produce a sentence-encompassing analysis can still be useful. Furthermore, any reasonably rapid parser that is capable of "chunking" a sentence into a few large substructures is valuable as a preprocessor to a system that might perform a final merging of phrases. This view is supported by the fact that many of the syntactic structures that give standard parsers so much trouble, such as movement[2], are probably better handled by more unusual approaches, such as constraint satisfaction, which can reason about phrase structure independently of linear precedence information. However, any system efficient at dealing with movement may well be ill-adept at rapidly building the phrases standard parsers can analyze with ease. Thus a parser of the sort described here, which can perform a good portion of the work involved in parsing an entire sentence, is in many cases a better alternative than a parser that can parse all sentences completely, but only at the expense of generating numerous additional incorrect parses.

4.4.3 The Need for Bottom-Up Methods

Any parser that hopes to break a sentence up into several unconnected parts must be capable of analyzing phrases with no predetermined knowledge of their highest projection level. Traditional top-down approaches assume that all phrases will eventually combine into a final sentence-wide phrase, and thus work from the presumed top category downward. If no top category is known, a parser must work from lexical level categories projecting structure

[2]Movement is the name given to the general language mechanism that moves phrase from a posited "deep structure" to the position they are found in "surface structure", the form we see. For example, it is common in linguistics to assume that the phrase "who" in "Who did Mary believe?" moves to the front of the sentence from a position to the right of believe, and that "did" moves also. The deep structure more closely resembles "Mary did believe who?".

upward.

The parser described here achieves a bottom-up capacity through the single-phrase and paired-phrase action tables, which describe how lexical and phrasal categories either project to higher phrases, or create a rule that combines several phrases into one of higher level.

Readers may notice that the use of special mechanisms to project single phrases and to connect neighboring phrases is unnecessary, since rules could perform the same task. However, since projection and binary attachment are so common, the parser's efficiency is greatly improved by the additional methods.

4.4.4 The Need for Top-Down Methods

If the parser overutilized bottom-up methods, it might find itself in the position of continually projecting lexical-level phrases with no rational for combining them. Thus, rules that have been created are given precedence to accept in new phrases before those phrases project. In conjunction with the "lowering" system, which usually prevents phrases from projecting once they have been accepted by rules, and the method of selecting output phrases, which gives precedence to longer phrases over many small phrases, the rules provide the parser with a top-down capability that prevents many unnecessary small phrases from being built bottom-up. An example of where this might apply is in the analysis of "the car doors". Were "car" and "doors" not lowered after being accepted by the rule started by the determiner "the", three unnecessary noun phrases would be built, "car", "doors", and "car doors".

4.4.5 The Need for Lowering

In the discussion on the need for top-down methods a rational for lowering phrases in the stack is given. The mechanism of lowering phrases is also intended to reduce computational cost, by introducing determinism into the parser. The effectiveness of the method can be seen from Graphs 11 through 14 of Appendix D, which compare the parser's speed with and without lowering. The difference is tremendous, a factor of more than 15 in time. Indeed, some parses which take much less than a second with lowering can take 5 minutes without.

Although lowering can stifle some correct parses, it does not do so often. Potential difficulties could arise from the parser's property of always generating the longest possible phrase it can from a sequence of words, a direct consequence of lowering. A problem phrase, with structure analogous to "the car doors" example above, might be "the car faces" in "the car faces an uphill battle". There "faces", if disambiguated incorrectly into a noun as well as a verb interpretation, would be included into a phrase "the car faces" and the verb interpretation would be lowered along with the noun "faces", preventing the sentence from being properly parsed. Fortunately, the only situation in which the longest-phrase property fails regularly is in verb argument attachment; with a more restrictive subcategorization system, it would not be much of a problem at all.

One might wonder why lowering phrases does not cause more problems for the parser than it does. Essentially, lowering applies whenever a phrase is built, preventing daughter phrases from projecting or starting rules. Thus, problems could arise whenever a phrase is lowered that should be the leftmost daughter in another phrase. Since lowering does not occur unless the same phrase were a (non-leftmost) daughter of another phrase, the possibility of conflict arises only when a process building a phrase oversteps its right boundary. English, generally thought to be a right-branching language[3], would seem to be a prime candidate for problems. Apparently, in English at any rate, the probability of failure is fairly low, probably because many phrases are started by determiners and other parts of speech which do not usually occur anywhere but the leftmost position. Problems can crop up, however, such as in "the car faces" above where a poor disambiguation could cause problems, or in "the walrus hunting men were tired" where "walrus" would be lowered by a rule started by "the", though it should become part of the adjectival "walrus hunting". Verbs which take multiple arguments can also cause problems, as in "I made the car doors". "The car doors" will be built into one phrase, preventing the interpretation where doors are being built and given to the car.

4.4.6 The Need for Transducer Functions

The inclusion of an ability to use transducer functions to create tree structure has roots in our previous experiences with principle-based structures. Modern linguistic theories, such as Government-Binding Theory, have shown themselves to be valuable constraint systems when applied to sentence tree-structure, but do not necessarily provide efficient means of initially generating the structure. By using functions to map between surface structure

[3] In other words, the structure tends to look like [a [b [c [...]]]].

and more principled trees, we have eliminated much of the computational cost involved in principled representations.

The advantage of utilizing such principled representation lies in the knowledge that it may be necessary for some other entity to perform a more detailed analysis of the sentence after being provided with the phrases the parser has created. It may be far easier to perform an analysis of the phrase structure of the sentence if the phrases the parser produces have been mapped into some standard representation. Similarly, the parser can be used as a substitute for other parsers in existing systems simply by modifying the output format of the phrases to conform with existing protocols.

4.5 The Grammar

As has been discussed, the grammar the parser uses includes a set of rule states; a set of paired-phrase actions; a set of single-phrase actions; and perhaps some transducer functions. Appendix B contains an entire sample grammar, used in various tests. To help understand the grammar a description of its format is given here.

4.5.1 Features

Although they were mentioned briefly in the section discussing the parser algorithm, phrase features deserve more attention. Every phrase has a category (part of speech) and a possibly empty set of features. The features serve as a further specification of the category, in the form of an association list: a list of feature names paired with feature values. Thus, a noun phrase might have the feature list `((plural . +) (nominative . -))` specifying that it is a plural, but not does not have nominative case. Such a phrase would match a grammar specification for a plural noun phrase. Primitive feature specification and passing mechanisms are provided by the grammar, though full unification is avoided.

4.5.2 The Grammar's Interpretation

Most of the grammar's interpretation has been described in the overview of the parser's description. However, some details remain to be discussed. Figure 4-5 provides a full Backus-Naur Form specification of the grammar.

Only the first matching `rule-action-spec` or `single-phrase-action-spec` will have

```
grammar ::= single-phrase-action-table | rule-state-action-table |
            paired-phrase-action-table

single-phrase-action-table ::= single-phrase-action-spec*
single-phrase-action-spec ::= (key-phrase single-phrase-action*)
single-phrase-action ::= ('START-RULE' rule-state-name-spec) |
                         ('PROJECT' phrase [single-action-feature-passing-method])
single-action-feature-passing-method ::= 'NEW-ONLY'

rule-state-action-table ::= rule-state-spec*
rule-state-spec ::= (RULE-STATE-NAME rule-action-spec*)
rule-action-spec ::= (key-phrase rule-action*)
rule-action ::= ('ACCEPT' rule-state-name-spec [rule-feature-passing-method]) |
                ('CLOSE' phrase [rule-close-specs]) |
                ('INSERT' phrase rule-insert-specs) |
                ('PASS' rule-state-name-spec [rule-feature-passing-method])
rule-feature-passing-method ::= 'NEW-FIRST' | 'OLD-FIRST' | 'NEW-ONLY'
rule-close-specs ::= rule-feature-passing-method [TRANSDUCER-NAME]
rule-insert-specs ::= rule-feature-passing-method rule-insert-locn
rule-insert-locn ::= 'LEFT' | 'RIGHT'

paired-phrase-action-table ::= paired-phrase-action-spec*
paired-phrase-action-spec ::= ((key-phrase . key-phrase) paired-phrase-action*)
paired-phrase-action ::= ('START-RULE' rule-state-name-spec paired-rule-specs) |
                         ('CONNECT' phrase [paired-connect-specs]) |
                         ('PROJECT' phrase paired-project-specs) |
                         ('INSERT' phrase paired-insert-specs)
paired-rule-specs ::= paired-action-feature-passing-method [paired-rule-locn]
paired-rule-locn ::= 'LEFT' | 'MIDDLE' | 'RIGHT'
paired-connect-specs ::= paired-action-feature-passing-method [TRANSDUCER-NAME]
paired-project-specs ::= paired-action-feature-passing-method paired-phrase-spec
paired-phrase-specs ::= 'LEFT' | 'RIGHT'
paired-action-feature-passing-method ::= 'LEFT-FEATURES' | 'RIGHT-FEATURES' |
                                         'LEFT-RIGHT' | 'RIGHT-LEFT'
paired-insert-specs ::= paired-action-feature-passing-method paired-insert-locn
paired-insert-locn ::= 'LEFT' | 'MIDDLE-LEFT' | 'MIDDLE-RIGHT' | 'RIGHT'

key-phrase ::= phrase
phrase ::= CATEGORY | (category feature*)
feature ::= (FEATURE-NAME . feature-value)
feature-value ::= VALUE | VARIABLE
rule-state-name-spec ::= RULE-STATE-NAME | (RULE-STATE-NAME feature*)
```

Figure 4-5: A BNF Specification for the Grammar

its actions applied.

Two action types were not discussed earlier, `INSERT` and `PASS`. `INSERT` is a means of inserting an empty category into a parse. The category is only made available for attachment to a particular phrase- either the phrase being analyzed in the case of a rule action, or either the left or right phrase in a paired-phrase action. The details of the attachment are specified by either `LEFT` or `RIGHT` in the case of a rule action (which side of the current phrase to insert the empty category on); or LEFT, MIDDLE-LEFT, MIDDLE-RIGHT or RIGHT in a paired-phrase action (which side of which phrase to insert the empty category on).

PASS is identical to ACCEPT except that the phrase being analyzed is not included in the list of phrases accepted so far. It still creates a new rule state. START-RULE, the paired-phrase action for starting a rule, takes a specification that describes where to put the new rule state, to the left of the left phrase (LEFT), to the left of the right phrase (MIDDLE), or to the right of the right phrase (RIGHT). The START-RULE of the single-phrase action table always places the rule state to the left of the phrase being analyzed.

The CLOSE and CONNECT actions can both include specifications of a transducer function. This must be the name of a function that will create a new phrase from the category, features, and daughters passed to it.

The remaining action specifications describe how features are passed in rules and in phrase creation. When a phrase is specified for creation (through PROJECT, CLOSE, INSERT, or CONNECT), features can be specified along with the category. These features are given to the phrase being created. When a PROJECT single-phrase action is taken, the default is to also give the new phrase all the features of the phrase being analyzed. If NEW-ONLY is specified, this will not happen. Furthermore, when rules are created, they can be given features (see rule-state-name-spec in Figure 4-5). These features accumulate as a rule chain fires, and all of the features will be given to any phrase created by the rule chain (with conflicts broken in favor of recent rule states). When the CLOSE rule action is taken, if NEW-ONLY is specified, instead of giving the phrase the features built up as the rule chain fired, the features of the phrase currently being analyzed will be passed. NEW-FIRST specifies to give both, resolving conflicts in favor of the current phrase, and OLD-FIRST resolves in the opposite manner.

In the context of the CONNECT paired-phrase action, normally no features beyond those immediately specified are given the new phrase. However, if LEFT-FEATURES or RIGHT-FEATURES is specified, the features of the left or right phrase will also be projected. And if LEFT-RIGHT or RIGHT-LEFT is given, features from both phrases will be projected with conflicts resolved in favor of the left or right phrase respectively.

Feature lists may include features of the form (FEATURE-NAME . variable) as well as the ordinary (FEATURE-NAME . VALUE). A variable is a symbol starting with a question mark, such as ?PL. In the context of a key-phrase (see Figure 4-5), a variable will be

```
(((DID (tense . +) (pl . ?A)) . (NP (pl . ?A)))
 (start-rule Q0 nil left))

(NAME1 ((N (proper . +) (pl . ?PL))
        (accept NAME1)
        (close (PROP (pl . ?PL)) new-first)))
```

Figure 4-6: Two feature passing examples, the first from a paired-phrase action table, and the second from a rule-state action table.

bound to the value `FEATURE-NAME` has in the relevant phrase being analyzed. Then, when features are being given to a rule state or new phrase, a variable feature will be given the value the variable was bound to, permitting the passing of arbitrary features. In addition, the variable `PHRASE` is bound to the current phrase in the context of rules and single-phrase actions, and `L-PHRASE` and `R-PHRASE` are bound to the respective phrases in paired-phrase actions. Actually, the current implementation allows arbitrary functions to be wrapped around variables. In the context of paired-phrases, if the same variable feature is given in both the right and the left key phrase, only if the features match will the actions be taken. In the first example in Figure 4-6, the action from the paired-phrase table specifies that if a phrase with category `DID` and feature-list `((tense . +))` is followed by a phrase with category `NP`, and both phrases have the same value for feature `pl`, then a rule with state `Q0` should be started to the left of the `DID`. In the second example, taken from a rule action table, if a rule with state `NAME1` occurs at the left end of a noun phrase with feature-list `((proper . +))`, then two actions should be taken. First, the noun phrase should be accepted and the rule state `NAME1` be placed at the right end of it, and secondly, a phrase with category `PROP` should be created, given the same `pl` value as the noun had, any features given over the course of the rule chain, and (with least priority) any features of the noun phrase.

4.5.3 Customizing a Grammar

There is a wealth of data to be gathered from free text, which this parser makes available at low cost. Depending on the particular application, various different linguistic or lexical structures will have particular relevence. The bottom up nature of the parser allows grammars to be written which selectively parse and analyze certain phrases but not others.

As an example, if a statistical distribution of various adjectival phrases is desired, it is possible to write a grammar which fires rules from adjectives, but leaves other structures unparsed. By utilizing a transducer function to record data rather than produce output phrases, such an analysis can be very efficiently implemented.

4.6 An Efficient Implementation

4.6.1 Introduction

The parser was designed to have a reasonably efficient implementation. No operation requires great searching or unification, and most lookup can be done through arrays. The grammar can be compiled into procedures, and if it is more efficient, projection and paired-phrase actions can be implemented as macros that produce rules.

Although the pseudo-algorithm given for the parser in Figures 4-2 and 4-3 is the parser's clearest interpretation, a modification can be made that results in extensionally equivalent. Because rules are given precedence on the stack, rules always fire in a successive chain. But then, it is not really necessary to put rules on the stack at all- they can just be fired immediately. Hence, by implementing rule states as functions that can call other rule state functions, some added efficiency can be gained.

Similarly, even non-rule actions (single-phrase, paired-phrase) can be translated into compiled functions, even though they are not called in succession.

The parser implemented utilizes these methods, and several others (indexing lookups into arrays by both sentence location and a category number, for instance) to parse sentences extremely rapidly.

4.6.2 Action Compilation

To illustrate how actions are compiled into LISP functions, Figure 4-7 contains one example taken from the grammar given in Appendix B. Appendix C contains several more examples.

4.6.3 Transducer Functions

As should be apparent from the sample grammar in Appendix B, the grammar provides sufficient feature passing methods that transducer functions serve primarily to adjust output form. The only other use in the example grammar is to provide a means of loading in verb

A rule-action-spec:

```
(CS2 ((VP (expect) (tense . -) (has-trace . ?a))
     (close (S (has-trace . ?a)) nil sentence))
```

The resulting compilation into a function follows. Notice the explicit handling of the variable ?A, and the building of a new phrase 'S (category number 139) via the transducer function `SENTENCE`.

```
(LAMBDA (PHRASE PREV-PHRASES PREV-FEATURES OLD-PHRASES PSTATE)
  (COND ((PHRASE-HAS-FEATURES PHRASE '((EXPECT) (TENSE . -)))
         (LET ((FEATURES (PHRASE-FEATURES PHRASE)))
           (LET ((?A (GET-FEATURE 'HAS-TRACE FEATURES)))
             (LOWER-PHRASES (CONS PHRASE PREV-PHRASES) PSTATE)
             (LOWER-PHRASES OLD-PHRASES PSTATE)
             ;; Call transducer-function SENTENCE to build new phrase.
             (PHRASEMAKER-SENTENCE PSTATE 139
              (IF ?A (CONS (CONS 'HAS-TRACE ?A) PREV-FEATURES) PREV-FEATURES)
              (CONS PHRASE PREV-PHRASES) 'S))))))
```

The following rule-storage-function inserts the compiled rule-action into a table at the correct location. The `:CAT-EXPECTED 138` indicates that the above function will only be applied to phrases with category 138, which represents VP. `INTRODUCE-RULE` enters the rule-action into an array. Notice how rule states are broken up into groups of actions indexed by particular categories.

```
(LAMBDA (N PREV-PHRASES PREV-FEATURES NEW-PHRASES PSTATE)
  ;; Make sure rule state does not already exist in location N.
  (COND ((NOT (MEMBER PREV-PHRASES
                (AREF (CDR (PSTATE-STATES PSTATE)) N (GET 'CS2 'STATE-NO)) :TEST #'EQUAL))
         ;; Notify parser of new rule state in location N.
         (PUSH PREV-PHRASES (AREF (CDR (PSTATE-STATES PSTATE)) N (GET 'CS2 'STATE-NO)))
         ;; Build new rule-action-spec.
         (LET ((NEW-RULE
                (MAKE-RULE :PREV-PHRASES PREV-PHRASES
                           :PREV-FEATURES PREV-FEATURES
                           :LOC N :CAT-EXPECTED 138 :FN #<DTP-COMPILED-FUNCTION VP-ACTION>
                           :NEW-PHRASES NEW-PHRASES :STATE-NAME 'CS2)))
           ;; Notify parser of new rule-action-spec.
           (INTRODUCE-RULE NEW-RULE PSTATE)))))
```

Figure 4-7: An example of action compilation. See Appendix C for more examples.

argument information.

The current implementation provides macros for creating transducer functions. The sample grammar illustrates the use of the `defphrasemaker` construct, which defines transducer functions. In the body of such a function, two special forms are available. One, `output-phrase`, takes in a category, features, and a list of daughters and returns a phrase to be used by the parser. Internally the category remains the one specified by the grammar, but the printed output form is modified. The `internal-phrase` special form efficiently creates a phrase through similar arguments, but the phrase cannot be processed by the parser. Its use is entirely for adjusting the output format.

4.6.4 Rapid Access

To keep lookup times to a minimum, the parser keeps most data in arrays indexed by both sentence location and category number or rule-state number. For instance, phrases are stored in arrays indexed by their start location and category number; their end location and category number; and their start and end location. Rule-action-spec functions (rule actions grouped by the category needed to fire them) are stored in an array indexed by their location and category-expected number. Rule states are stored in an array indexed by their location and their state number (this servers to facilitate checking whether a rule state already exists in a given location, in which case an identical one will not be created). Paired-phrase actions are stored in an array indexed by the category of the phrase needed to complete the pair. The large number of categories used by the sample grammar keeps the size of array entries small.

One unfortunate disadvantage of using arrays is the amount of time necessary to clear them after a parse. Each phrase and rule state is recorded during the parse, so that array entries not utilized are not cleared, but a sizable percentage of total parse time is still devoted to cleanup (almost 10%).

4.7 Parser Performance

In judging the parser performance, there are three distinct areas to consider. One is simply the traditional time and space considerations. How fast is the parser? A second is grammar coverage- how much work does the parser perform on the average sentence, breaking it up into how many pieces? The third, and hardest to measure, is parser accuracy. Of the parses produced, what percentage represent those consistent with a human's interpretation of the sentence? Each question is discussed in turn.

4.7.1 Statistical Performance Figures

The graphs of Appendix D represent the parser's performance on 1300 LOB sentences, randomly chosen from throughout the LOB. The average time spent by the parser on each word is .039 seconds, with a standard deviation of .07 seconds. The average length of a phrase produced is 3.51 words, with a standard deviation of 2.08 words.

The time/word average of .039 seconds is much faster than most general purpose parsers.

Many parsers in which far most effort has been spent optimizing algorithms and code take far more time. As an example, [Haas 1989] reports that after implementing numerous optimizations, rewriting his unification grammar parser in C, and running his parser on a Sun 4, it still takes an average of .25 seconds per word.

Graphs 1 and 2 show the distribution of parse times vs. the number of words in a sentence. A polynomial fit gives an exponent of .66, less than linear. An examination of the graphs gives a feel for why- the graph is almost random. Graphs 3 and 4 describe the size of phrases output by the parser, and the size's relation to parser time spend on each word. The clumping of graph 4 makes the .04 sec/word and 3.5 word/phrase figures visible. Graphs 5 and 6 compare the total number of phrases and rules created during the parse (not necessarily output) to time and sentence length. Here there is a clearer curve. The number of items generated is essentially linear with sentence length (an exponent of 1.038). So, the reason for the small relation between input length and parse time is that different phrases and rules absorb vastly different lengths of time, depending on the amount of feature passing that they do, the number of items they pair with, etcetera. Graphs 7,8,9 and 10 break Graphs 5 and 6 up into rules and phrases individually. Graphs 11 through 14 display the parser's performance without lowering. Obviously, this significantly degrades performance (more than a factor of 15).

Graphs 1 through 10 resulted from a single run on a Symbolics 3600 series Lisp Machine. Undoubtedly, by implementing the code in a language other than LISP, say C running on a Sun, the speeds could be improved dramatically. However, the current speeds are sufficient to parse the entire LOB in about 11 hours, which is plenty for most applications. It is unlikely that parsing would be the bottleneck in any application of the parser. Currently, fetching a sentence and performing dictionary lookup take more time than parsing. In fact, during long runs garbage collection of all the parses generated seems to become the dominant factor.

4.7.2 Parser Coverage

The sample parses in Appendix E show that the parser, even with a limited grammar, performs a substantial amount of useful parsing. As the above section mentioned, the average length of a phrase produced is 3.5 words, quite large when one considers that this the average is brought down substantially by single-word phrases such as conjunctions at regular intervals, and that the grammar currently only rarely attaches phrases across punctuation marks.

There are several glaring omissions in the grammar, such as dates, complicated titles, parentheticals, adjuncts, complicated adverbial and adjectival phrases, and quotes. As the grammar is expanded the average length of phrases should increase dramatically, but it is important not to introduce constructs that will accept too many illegal parses. A larger dictionary will help prevent this.

4.7.3 Parser Accuracy

Judging parser accuracy is a very difficult task, because there is no way to do it automatically. Sentences and their parses must be individually evaluated. More importantly, there is no standard metric for measuring a parser's accuracy. Indeed, without knowing how the output is going to be used, there can be none. For example, a post-processor that can utilize the phrases the parser produces that it does not output may find a slightly malformed parse correct, while another might not. And it is unclear whether a phrase is completely misparsed if its top-level is incorrect but constituent phrases are accurate.

Some Parser Errors

The best way to visualize the parser's accuracy is to examine the sample parses of Appendix E. Representative errors that occur during the parsing of Harrison's sentences are discussed here. All of these errors are essentially minor, and the parser manages to build up substanital sentence structure despite them. Most could be prevented with more detailed lexical knowledge or a larger grammar, and most would have occurred in other parsers also.

In sentence 1, the parser is unable to build any substantial structure out of the second half of the sentence because the grammar is not sophisticated enough to introduce an empty (non lexical) phrase as the subject of "having" and "concerned". However, the parse is accurate as far as it goes, and a slightly improved grammar would extend the parse considerably.

In sentence 7, "computer networks" is mistakenly made the object of "build", because the subcategorization scheme does not provide enough information to distinguish this case from the case of "build me a house". Without more detailed lexical information, it is hard to see how any parser could prevent such a mistake.

In sentence 8, the verb phrase "using packed address recognition" is interpreted more as an adjective than correctly as an adverb. Outside of not attaching the phrase, it is difficult to see how this error could be prevented by any parser.

In sentence 9, "estimating" apparently takes a prepositional phrase as an argument in the subcategorization system. However, "under heavy loads" would not have been attached to "performance" in any case in the grammar used. Again, a larger and more detailed grammar would likely improve the parse.

In sentence 12, "with", which should be interpreted as a particle in the phrase "deal with", is interpreted as a preposition, incorrectly causing a prepositional phrase to be built. More detailed lexical knowledge would have prevented this error.

In sentence 15, "of materiel" is incorrectly attached to "introduce", again reflecting a lack of lexical and subcategorization information that makes such errors hard to prevent.

In sentence 19, "of" is interpreted as an adverb in a situation where it is clearly a particle in the phrase "think of". Here the irregularity of the LOB tagging system comes up: "of" should not be defined as an adverb at all.

Chapter 5

Conclusions and Further Work

The parser clearly represents a significant advance in that it performs extremely rapidly; runs on essentially unrestricted text; and produces accurate, single-valued results. It is a strong support for the notion that nondeterministic parsers running large grammars designed to fully describe all potential inputs are the wrong approach to parsing large texts.

If text access can be made fast enough, there is no reason to suppose that the parser can not be reimplemented in a faster language or on a faster computer to reduce the time taken to parse a million words of text to an hour, or fifteen minutes. A more carefully planned grammar should be able to drastically increase the portion of text parsed into large phrases without greatly reducing accuracy. This combination of speed, coverage, and accuracy without ambiguity make this system a uniquely practical one for the analysis of unrestricted text. With luck, some of the researchers producing parsers with substantial output ambiguity will be swayed by its results.

5.1 Future Goals

The long term intentions for the parsing system are for it to become a general-purpose interface to text, providing researchers with a standard method of retrieving and parsing arbitrary documents into a linguistic form which suits their particular needs. The widespread availability of large on-line texts should provoke innovative new avenues for linguistic research.

A number of previous sections have alluded to various areas for future system enhancement, and for more detailed research in general. Current plans include

- introducing a weighting criterion into the disambiguator so that words and part of speech sequences with greater statistical significance affect the disambiguation more than statistically unreliable data;
- switching from the current verb argument system, taken from the Oxford Advanced Learner's Dictionary of Contemporary English [OALDCE 1973] to the more detailed and constrained system in Longman's Dictionary of Contemporary English [LDOCE 1987], which should increase parse accuracy some;
- expanding the grammar to include some more unusual structures such as dates, names of locations, and parentheticals, and also some more sentence level structures;
- reading in a large dictionary such as [LDOCE 1987] so as to expand the lexicon from the LOB-derived dictionary;
- implementing a morphological analyzer to guess the categories of unknown words;
- speeding up the parser support system so that the parser can perform to its full potential;
- applying the parser to several million words of Wall Street Journal text.

In terms of fundamental research, the most interesting next step is to build a post-processor to the parser that can deterministically build up structures the current parser suffers with. Needless to say, this will require greater lexical and semantic data to be available. The parser's current architecture enables considerable interaction and would permit a post-processor to "request" further processing of certain sentence areas, for instance, or the introduction of new grammar rules at specific points in the parse.

Bibliography

[Church 1988] Church, K. W. (1988) A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. *Proceedings of the Second Conference on Applied Natural Language Processing*, 136-143.

[DeRose 1988] DeRose, S. J. (1988) Grammatical Category Disambiguation by Statistical Optimization. *Computational Linguistics* 14:1 31-39

[Garside *et al* 1987] Garside, R., Leech, G. and Sampson, G. (1987) *The Computational Analysis of English: A Corpus-Based Approach.* Longman.

[Gazdar *et al* 1985] Gazdar, G., Klein, E., Pullam, G. K. and Sag, I. A. (1985) *Generalized Phrase Structure Grammar.* Harvard University Press.

[Haas 1989] Haas, A. (1989) Practical Parsing of Generalized Phrase Structure Grammars. *Compuational Linguistics* 15:4 219-232

[Harrison 1988] Harrison, P. (1988) "A New Algorithm for Parsing Generalized Phrase Structure Grammar," Ph.D. Thesis, Applied Mathematics, Univ. of Washington.

[OALDCE 1973] Oxford Advanced Learner's Dictionary of Contemporary English (1973), Oxford University Press.

[Leech *et al* 1983] Leech, G., Garside R. and Atwell, E. S. (1983) The automatic grammatical tagging of the LOB Corpus. *ICAME News* 7, 13-33.

[LDOCE 1987] Longman Dictionary of Contemporary English (1978), second edition (1987), Longman Group Limited.

[Marcus 1980] Marcus, M. (1980) *A Theory of Syntactic Recognition for Natural Language.* MIT Press.

[Milne 1988] Milne, R. (1988) Lexical Ambiguity Resolution in a Deterministic Parser, in *Lexical Ambiguity Resolution*, ed. by S. Small *et al.* Morgan Kaufmann.

Appendix A: The LOB Category System

6.1 The Categories

The list of categories and their interpretations presented here is taken from [Garside *et al* 1987], and has been slightly changed.

Category	Interpretation
*	opening quote
**	closing quote
*-	-
*COL	:
*COM	,
*EXC	!
*LBR	[
*LPR	(
*PER	.
*QUE	?
*RBR	]
*RPR	)
*SEM	;
ABL	pre-quantifier (quite, rather)
ABN	pre-quantifier (half, all)
ABX	pre-quantifier and double conjunction (both)
AP	post-determiner (many, much, next, former, other)
APS	plural post-determiner (others)
APS$	possessive plural post-determiner (others')
AP$	possessive post-determiner (other's)
AT	singular article (a, an, every)
ATI	neutral article (the, no)
BE	be
BED	were
BEDZ	was
BEG	being
BEM	am, 'm

Category	Interpretation
BEN	been
BER	are 're
CC	conjunction (and, but, or, yet)
CD	general cardinal
CDS	plural cardinal (threes, 3s)
CD1	1, one
CD1S	ones
CD1$	one's
CD$	possessive cardinal (two's complement)
CD-CD	hyphenated number
CS	subordinating conjunction (if, although, that, because, as)
DO	do
DOD	did
DOZ	does
DT	singular determiner (this, that, another)
DT$	possessive singular determiner (another's)
DTI	neutral determiner capable of pronominal function (any, some)
DTS	plural determiner (these, those)
DTX	determiner and double conjunction (either, neither)
EX	existential (there)
&FO	formula
&FW	foriegn word
HV	have
HVD	had 'd
HVG	having
HVN	had
HVZ	has 's
IN	preposition
JJ	general adjective
JJB	attributive adjective (utter, late, 15-nation in "the 15-nation council") or sematically superlative adjective (main, chief, top)
JJR	comparative adjective (older, better)
JJT	morphologically superlative adjective (smartest, brightest)
JNP	proper adjective (Welsh, Keynsian)
MD	modal auxiliary
NC	cited word (too many ifs and buts)
NN	singular common noun
NNP	singular common noun with wic (Londoner)
NNPS	plural common noun with wic (Englishmen)
NNPS$	possessive plural common noun with wic (Associate's)
NNP$	possessive singular common noun with wic (Institution's)
NNS	plural common noun
NNS$	possessive plural common noun (girls')
NNU	abbreviated unit of measurement (in, kg)
NNUS	plural unit of measurement (ins, feet)
NNUS$	possessive plural abbrev unit of measurement (ccs')
NNU$	possessive abbreviated neutral unit of measurement (cwt's)
NN$	possessive singular common noun (girl's)
NP	singular proper noun (Naismith, October)
NPL	singular locative noun with wic (Street, Island)
NPLS	plural locative noun with wic (Streets)
NPLS$	possessive plural locative noun with wic (Streets')
NPL$	possessive singular locative noun with wic (Island's)
NPS	plural proper noun (Americas)
NPS$	possessive plural proper noun (Americas')

Category	Interpretation
NPT	singular titular noun (Bishop, Mrs)
NPTS	plural titular noun (Presidents)
NPTS$	possessive titular noun (Presidents')
NPT$	possessive singular titular noun with wic (Mayor's)
NP$	possessive singular proper noun (Juliet's)
NR	adverbial noun (home, west, Sunday)
NRS	plural adverbial noun (Sundays)
NRS$	possessive plural adverbial noun (Sundays')
NR$	possessive adverbial noun (tomorrow's)
OD	ordinal numeral (second, 2nd)
OD$	possessive ordinal (sixth's)
PN	nominal pronoun (anybody, everyone, none, nothing)
PN$	possessive nominal pronoun (everyone's)
PPL	singular reflexive personal pronoun (myself, herself)
PPLS	plural reflexive personal pronoun (ourselves, themselves)
PP1A	I
PP1AS	we
PP1O	me
PP1OS	us
PP2	you
PP3	it
PP3A	he, she
PP3AS	they
PP3O	him, her
PP3OS	them
PP$	my, your, our
PP$$	mine, yours, ours
QL	degree adverb (very, so, too)
QLP	post-adjectival/adverbial degree adverb (enough, indeed)
RB	adverb (quickly, well, else, along)
RBR	comparative adverb (better, longer, more)
RBT	superlative adverb (best, longest)
RB$	adverb with possessive suffix (else's)
RI	adverb, homograph of preposition (before, along, by)
RN	nominal adverb (here, there, now, then)
RP	prepositional adverb which is also particle (across, off, up)
TO	infinitive to
UH	interjection (hello, no)
VB	base verb form (eat, request)
VBD	verb, preterite (ate, requested)
VBG	present participle (eating, requesting)
VBN	past participle (eaten, requested)
VBZ	3rd singular form of verb (eats, requests)
WDT	interrogative wh-determiner (what, which, whatsoever)
WDTR	relative wh-determiner (which)
WP	wh-pronoun neutral for case (who, whoever)
WPA	nominative wh-pronoun (whosoever)
WPO	interrogative objective wh-pronoun (whom, whomever)
WPOR	relative objective wh-pronoun (whom)
WPR	relative wh-pronoun neutral for case (that, who)
WP$	interrogative whose
WP$R	relative whose
WRB	wh-adverb (when, how in "how green", however in "however well")
XNOT	not
ZZ	letter of the alphabet

6.2 Deficiencies in the Category System

Several deficiencies make the LOB category system less useful than could be desired. Most can be described as conflation- too many different uses bound up into one category symbol.

First, while some differentiation is made between noun types (whether they are plural or possessive, for example) and verb types (the tense) not enough usage information is provided. For instance, no indication is provided as to whether a verb is transitive or intransitive, what it's particular subcategorization is in the context, or whether it is being used as an adjective. Similarly, proper nouns which are first or last names are not distinguished from location names or organizations.

Thus, during disambiguation many parts of speech which should not appear in a certain context are accepted, because other categories that they are conflated with do appear in the context.

Another common deficiency is irregularity in tagging. Several common words, such as "a" and "the" are at rare intervals assigned unusual categories ("the" is an adverb, etc.) which, despite their count, can throw off disambiguation and parsing significantly.

6.3 Mapping Between the LOB and the Parser

Because the LOB category system is not ideal for use in parsing, a table, given here, maps between LOB categories and parser categories. All LOB categories not found in the table map to themselves.

LOB Categegory	Parser Category and Features
abn "all"	det (pro . +)
abn	pro)
at	det (def . -)
ati	det (def . +)
be	is (int-roles . ((np) (i-bar) (ap) (pp)))
bed	is (tense . +) (pl . +) (int-roles . ((np) (i-bar) (ap) (pp)))
bedz	is (tense . +) (pl . -) (int-roles . ((np) (i-bar) (ap) (pp)))
beg	is (ing . +) (int-roles . ((np) (i-bar) (ap) (pp)))
bem	is (tense . +) (pl . -) (pers . 1) (int-roles . ((np) (i-bar) (ap) (pp)))
ben	is (int-roles . ((np) (i-bar) (ap) (pp)))
ber	is (tense . +) (pl . +) (int-roles . ((np) (i-bar) (ap) (pp)))
bez	is (tense . +) (pl . -) (int-roles . ((np) (i-bar) (ap) (pp)))
cds	cd (pl . +)
cd1	cd
cd1s	cd (pl . +)
cs "if"	comp (wh . -) (adverbial . +)
cs "that"	comp (wh . -) (that . +)
cs "because"	comp (adv-comp . +)
cs "unless"	comp (adv-comp . +)
cs "which"	comp (wh . +)
cs "who"	comp (wh . +)
cs	comp (wh . -)
do	did (int-roles . (((np (action . +)))))
dod	did (tense . +) (int-roles . (((np (action . +)))))
doz	did (tense . +) (pl . -) (int-roles . (((np (action . +)))))
dt "another"	det (pro . +)
dt "this"	det (def . +) (pro . +)
dt "that"	det (def . +) (pro . +)
dt	det (def . +)
dti	det (def . +)
dts "these"	det (def . +) (pl . +) (pro . +)
dts "those"	det (def . +) (pl . +) (pro . +)
dts	det (def . +) (pl . +)
ex	pro
hvg	have (ing . +) (int-roles . ((np) (i-bar)))
hvd	have (tense . +) (int-roles . ((np) (i-bar)))
hvn	have (int-roles . ((np) (i-bar)))
hvz	have (tense . +) (int-roles . ((np) (i-bar)))
hv	have (tense . +) (int-roles . ((np) (i-bar)))
in "of"	p (imm-connect . +) (takes-vp . +)
in "than"	p (than . +)
in "per"	p (imm-connect . +)
in "by"	p (passive-pp . +) (takes-vp . +)
in "between"	p (between . +)
in "for"	p (takes-s . +)
jjb	jj
jjr	jj
jjt	jj
jnp	jj
md "to"	md
md	md (tense . +)
nn	n (pl . -)
nn$	n (possessive . +)

LOB Categegory	Parser Category and Features
nnps	n (pl . +) (proper . +)
nnps$	n (pl . +) (proper . +) (possessive . +)
nns	n (pl . +)
nns$	n (possessive . +) (pl . +)
nnus	nnu (pl . +)
np	name (pl . -)
nps	name (pl . +)
np$	name (possessive . +)
npl	n (proper . +)
npt	n (title . +) (pl . -)
npts	n (title . +) (pl . +)
nr	n (temporal . +)
od	jj
pn	n (somebody . +)
pp3a "he"	pro (pers . 3) (nom . +) (gen . masc) (pl . -)
pp3a "she"	pro (pers . 3) (nom . +) (gen . fem) (pl . -)
pp3o "him"	pro (pers . 3) (acc . +) (gen . masc) (pl . -)
pp3o "her"	pro (pers . 3) (acc . +) (gen . fem) (pl . -)
pp3os "them"	pro (pers . 3) (acc . +) (pl . +)
pp3as "they"	pro (pers . 3) (nom . +) (pl . +)
pp1os "us"	pro (pers . 1) (acc . +) (pl . +)
pp1as "we"	pro (pers . 1) (nom . +) (pl . +)
pp1a "i"	pro (pers . 1) (nom . +) (pl . -)
pp1o "me"	pro (pers . 1) (acc . +) (pl . -)
pp2 "you"	pro (pers . 2)
pp3 "it"	pro (pl . -)
ppls	pro
pp$	det (def . +)
rb	adv
rbr	adv (er . +)
rn	adv (temporal . +)
rp	rp
to	to (to . +)
vbd	v (tense . +)
vbg	v (ing . +)
vbn	v (passive . +)
vb	v (ambig-tense . +)
vbz	v (tense . +)
wdt	pro (wh . +) (action . +)
wdtr	pro (wh . +)
wp	pro (wh . +)
wpr	pro (wh . +)
wrb "when"	adv (wh . +) (comp . +) (temporal . +)
wrb "why"	adv (comp . +)
wrb "how"	adv (comp . +)
wrb "where"	adv (comp . +) (locative . +)
wrb	adv (wh . +)

Appendix B: A Parser Grammar

The grammar used by the parser for all tests in this thesis is presented here. It is not particularly comprehensive, and does not represent a grammar that is tailored to any specific application.

7.1 The Single-Phrase Action Table

```
((DET (PRO . +)) (start-rule DET0)
                 (project (NP (big . -)) new-only) )
(DET (start-rule DET0) )
(NUM (start-rule DET1) )
(JJ (start-rule JJ0) )
(AP (start-rule DET1) )
(IS (start-rule (VP1 (tense . -)))
    (start-rule ISQ1) )
(MD (start-rule (VP0 (tense . -))))
((NP (temporal . +)) (start-rule (VP0 (tense . -))))
(ADV (start-rule (VP1 (tense . -)))
     (start-rule JJ0) )
(TO (start-rule (VP0 (tense . -))))
(HAVE (start-rule (VP1 (tense . -) )))
(DID (start-rule (VP0 (tense . -))))
(V (start-rule (VP3  (tense . -))))
(NNU (start-rule NUM0) )
(CD (start-rule NUM0) )
((N (title . +)) (start-rule NAME0)
                 (start-rule DET1) )
(N (start-rule DET1) )
(PROP (start-rule (DET1 (prop . +))))
(NAME (start-rule NAME1) )
(QL (start-rule JJ0) )
(QLP (start-rule JJ0) )
(PRO (project (NP (big . -)) new-only))
```

7.2 The Rule-State Action Table

```
(BTWN-NUM0 (P (accept BTWN-NUM1)))
(BTWN-NUM1 (NUM (accept BTWN-NUM2)))
(BTWN-NUM2 (CC (accept BTWN-NUM3)))
(BTWN-NUM3 (NUM (close NUM)))
(CMA0 ((NP (trace)) (accept CMA1 new-only)))
(CMA1 (*CMA (accept CMA2))
      (*- (accept CMA4)))
(CMA2 ((NP (trace) (det . +)) (accept (CMA3 (cma . +))))
      (PROP (accept (CMA3 (cma . +)))))
(CMA3 (*CMA (close NP))
      (*PER (close NP)))
(CMA4 ((NP (trace) (det . +)) (accept (CMA5 (cma . +))))
      (PROP (accept (CMA3 (cma . +)))))
(CMA5 (*- (close NP)))
(CP0 ((NP (has-trace)) (pass (CP2 (bound-to . phrase)))))
(CP2 (COMP (pass CP2))
     ((NP (wh) (trace)) (pass CP3) ))
(CP3 (VP (insert (NP (trace . +) (has-trace . +)) nil right) ))
(CS1 ((NP (nom) (has-trace)) (accept CS2)))
(CS2 ((VP (expect) (tense . -) (has-trace . ?a))
      (close (S (has-trace . ?a)) nil sentence)))
(DET0 ((DET (def . ?A)) (accept (DET1 (def . ?A) (det . +) (big . +)))))
(DET1 ((V (ING . +)) (accept DET1))
      ((V (passive . +)) (accept DET1))
      (AP (accept DET1))
      (NUM (accept DET1) (close NP old-first))
      ((PROP (possessive . +)) (close NP old-first))
      (PROP (accept (DET2 (prop))) (close NP old-first))
      ((N (somebody . +)) (accept DET3 old-first) (close (NP (big . +)) old-first))
      ((N (possessive . +) (somebody)) (close (NP (possessive . +)) old-first))
      ((N (pl . +)) (close (NP (det . +) (big . +)) old-first))
      ((N (temporal . +)) (close (NP (big . +)) old-first))
      (N (close (NP (big . +)) old-first) (accept DET2)))
(DET2 ((N (possessive . +)) (close (NP (big . +)) old-first))
      ((N (pl . +) (temporal)) (close (NP (det . +) (big . +)) old-first))
      ((N (temporal)) (accept DET2) (close (NP (big . +)) old-first)))
(DET3 (JJ (close NP))
      (AP (close NP)))
(ISQ1 (IS (accept ISQ2)))
(ISQ2 ((NP (trace)) (accept ISQ3)))
(ISQ3 (NP (close (IP (question . +))))
      (AP (close (IP (question . +))))
      (PP (close (IP (question . +))))
      (IP (close (IP (question . +)))))
(JJ0 (QL (accept JJ0))
     (ADV (accept JJ0))
     (QLP (accept JJ0) (close AP))
     (JJ (accept JJ0) (close AP)))
(AP-AND0 (AP (accept AP-AND0))
         (*CMA (accept AP-AND0))
         (CC (accept AP-AND1)))
(AP-AND1 (AP (close AP) (accept AP-AND0))
         (CC (accept AP-AND1)))
(NAME0 ((N (TITLE . +)) (accept NAME1) ))
(NAME1 ((NAME (possessive . +) (pl . ?A)) (close (PROP (pl . ?A)) new-first))
```

```
      ((N (proper . +) (pl . ?A)) (accept NAME1)
       (close (PROP (pl . ?A)) new-first))
      ((NAME (pl . ?A)) (accept NAME1) (close (PROP (pl . ?A)) new-first)))
(NUM0 ((NNU (pl . ?A) (possessive . ?b))
       (accept NUM0) (close (NUM (pl . ?A) (possessive . ?B))))
      ((CD1 (pl . ?A) (possessive . ?b))
       (accept NUM0) (close (NUM (pl . ?A) (possessive . ?B))))
      ((CD (pl . ?A) (possessive . ?b))
       (accept NUM0) (close (NUM (pl . ?A) (possessive . ?B)))))
(VP0 ((MD (tense . ?t)) (accept (VP1 (tense . ?t))))
     ((NP (TEMPORAL . +)) (accept VP1))
     (TO (accept (VP1 (to . +))))
     ((DID (tense . ?t))
      (accept (VP1 (tense . ?t)))
      (close (VP (verb-phrase . phrase)) new-first vp))
     (ADV (accept VP0)))
(VP1 ((IS (tense . ?t)) (close (VP (verb-phrase . phrase)) new-first vp)
      (accept (VP2 (is . +) (tense . ?t))))
     ((HAVE (tense . ?t)) (accept (VP2 (tense . ?t)))
      (close (VP (verb-phrase . phrase)) new-first vp))
     (XNOT (accept VP1))
     (ADV (accept VP1))
     (DID (close (VP (verb-phrase . phrase)) new-first vp))
     (V (accept (VP4 (verb-phrase . phrase)) new-first)
        (close (VP (verb-phrase . phrase)) new-first vp)))
(VP2 ((IS (tense . ?t)) (close (VP (verb-phrase . phrase)) new-first vp)
      (accept (VP3 (is . +) (tense . ?t))))
     (XNOT (accept VP2))
     (ADV (accept VP2))
     (DID (close (VP (verb-phrase . phrase)) new-first vp))
     (V (accept (VP4 (verb-phrase . phrase)) new-first)
        (close (VP (verb-phrase . phrase)) new-first vp)))
(VP3 (V (accept (VP4 (verb-phrase . phrase)) new-first)
        (close (VP (verb-phrase . phrase)) new-first vp))
     (HAVE (close (VP (verb-phrase . phrase)) new-first vp))
     (DID (close (VP (verb-phrase . phrase)) new-first vp))
     (XNOT (accept VP3))
     (ADV (accept VP3) ))
(VP4 (RP (close VP old-first vp)))
(Q0 ((DID (tense . +)) (accept Q1))
    ((MD (tense . +)) (accept Q1))
    ((HAVE (tense . +)) (accept Q1))
    ((IS (tense . +)) (accept Q1)))
(Q1 (NP (accept Q2) ))
(Q2 ((VP (tense . -) (expect)) (close CP nil question)))
(NP-AND0 ((NP (cma)) (accept NP-AND0))
         (*CMA (accept NP-AND0))
         (CC (accept NP-AND1)))
(NP-AND1 ((NP (cma)) (close NP) (accept NP-AND0))
         (CC (accept NP-AND1)))
```

7.3 The Paired-Phrase Action Table

```
((COMP . S) (connect CP left-right))
(((ADV (comp . +)) . S) (connect NP left-right))
```

```
((COMP . VP) (insert (NP (trace . +) (has-trace . +)) nil middle-right))
(((P (between . +)) . NUM) (start-rule BTWN-NUM0 nil left))
(((DID (tense . +) (pl . ?a)) . (NP (pl . ?a))) (start-rule Q0 nil left))
(((MD (tense . +) (pl . ?a)) . (NP (pl . ?a))) (start-rule Q0 nil left))
(((HAVE (tense . +) (pl . ?A)) . (NP (pl . ?a))) (start-rule Q0 nil left))
(((HAVEP (expect . +)) . NP) (connect HAVEP))
((AP . CC) (start-rule AP-AND0 nil left))
((AP . *CMA) (start-rule AP-AND0 nil left))
((NP . CC) (start-rule NP-AND0 nil left))
((NP . *CMA) (start-rule NP-AND0 nil left))
((NP . (NP (locative . +))) (connect NP left))
(((NP (possessive . +)) . NP) (connect (NP (det . +) (big . +)) right-features))
(((NP (temporal) (possessive)) . V) (start-rule (VP3  (tense . -)) nil middle))
((NP . (PP (imm-connect . +))) (connect (NP (big . +)) left-features))
(((NP (temporal) (acc) (wh . ?a) (has-trace . ?b)) .
  (VP (expect) (tense . +) (has-trace . ?c)))
 (connect (S (wh . ?a) (has-trace . ?b) (has-trace . ?c)) nil sentence))
(((NP (temporal) (acc) (wh . ?a) (has-trace . ?b)) .
  (VP (to) (expect) (ambig-tense . +) (tense . -) (has-trace . ?c)))
 (connect (S (wh . ?a) (has-trace . ?b) (has-trace . ?c)) nil sentence))
(((NP (has-trace)) .
  (VP (has-trace) (passive . +) (is . +) (expect . passive-arg)))
 (insert (NP (trace . +) (has-trace . +) (bound-to . l-phrase)) nil right))
(((NP (temporal) (det . +)) . (VP (expect) (passive . +) (tense . -)))
 (connect NP left-features))
(((NP (temporal) (wh) (det . +)) . (VP  (expect) (ing . +) (tense . -)))
 (connect NP left-features))
(((NP (wh . +)) . (VP (expect . NP) (to . +) (has-trace)))
 (insert (NP (trace . +) (has-trace . +) (bound-to . l-phrase)) nil right))
(((NP (temporal) (has-trace) (prop) (det . +)) .
  (CP (adv-comp) (has-trace . +)))
 (connect NP left-features))
(((NP (temporal) (has-trace) (possessive) (det. +)) . (S (has-trace . +)))
 (project CP right-features right))
(((NP (temporal) (wh . +)) . (VP (expect) (has-trace . +) (tense . -)))
 (connect S left-features))
(((NP (temporal) (det . +)) . (COMP (adv-comp)))
 (start-rule CP0 nil left))
(((NP (temporal) (has-trace) (det . +) (possessive)) . (NP (temporal) (possessive)))
 (start-rule CP0 nil left))
((NP . *CMA) (start-rule CMA0 nil left))
((NP . *-) (start-rule CMA0 nil left))
(((ADV (er . +)) . (PP (than . +))) (connect ADV))
(((P (takes-vp . +)) . VP) (connect (PP (assigns-case . acc)) left-right))
(((P (takes-s . +)) . S) (connect (PP (assigns-case . acc)) left-right))
((P . NP) (connect (PP (assigns-case . acc)) left-right))
(((VP (expect . (CP (adverbial . +)))) . (CP (has-trace . ?a) (adverbial . +)))
 (connect (VP (has-trace . ?a) (arg-type . cp)) left-features vp-take-arg))
(((VP (expect . AP)) . AP)
 (connect (VP (arg-type . ap)) left-features vp-take-arg))
(((VP (expect . (CP (that . +)))) . (CP (has-trace . ?a))) ;; (that . +)))
 (connect (VP (has-trace . ?a) (arg-type . cp)) left-features vp-take-arg))
(((VP (expect . CP)) . (CP (has-trace . +)))
 (connect (VP (has-trace . +) (arg-type . cp)) left-features vp-take-arg))
(((VP (expect . CP)) . S) (project CP right-features right))
(((VP (expect . NP)) . (NP (has-trace . ?a) (temporal) (wh) (nom)))
 (connect (VP (has-trace . ?a) (arg-type . np)) left-features vp-take-arg))
(((VP (expect . passive-arg)) . (NP (trace . +) (temporal) (wh) (nom)))
 (connect (VP (arg-type . passive-arg)) left-features vp-take-arg))
```

```
(((VP (expect . (NP (action . +)))) .
  (NP (action . +) (has-trace . ?a) (temporal) (nom)))
 (connect (VP (has-trace . ?a) (arg-type . np)) left-features vp-take-arg))
(((VP (expect . PP)) . (PP (has-trace . ?a)))
 (connect (VP (has-trace . ?a) (arg-type . pp)) left-features vp-take-arg))
(((VP (expect . (VP (tense . -) (ing . +)))) .
  (VP (expect) (tense . -) (has-trace . ?a) (ing . +)))
 (connect (VP (has-trace . ?a) (arg-type . vp)) left-features vp-take-arg))
(((VP (expect . (VP (tense . -) (to . +)))) .
  (VP (expect) (has-trace . ?a) (tense . -) (to . +)))
 (connect (VP (has-trace . ?a) (arg-type . vp)) left-features vp-take-arg))
(((VP (expect . (VP (case-project . +)))) . (S (has-trace . ?a)))
 (connect (VP (has-trace . ?a) (arg-type . vp)) left-features vp-take-arg))
(((VP (expect . (VP (case-project . +)))) . (NP (nom) (has-trace)))
 (start-rule CS1 nil middle))
(((VP (case-project . +) (expect . CP)) . (NP (trace))) (start-rule CS1 nil middle))
(((VP (case-project . +) (expect . CP)) . COMP) (start-rule CS1 nil right))
(((VP (expect . VP)) . (VP (has-trace . ?a) (expect)))
 (connect (VP (has-trace . ?a) (arg-type . vp)) left-features vp-take-arg))
((VP . (NP (temporal . +))) (connect VP left-features vp-take-modifier))
((VP . XNOT) (connect VP left-features vp-take-modifier))
((VP . ADV) (connect VP left-features vp-take-modifier))
((VP . RP) (connect VP left-features vp-take-modifier))
((VP . (CP (adv-comp . +))) (connect VP left-features vp-take-modifier))
(((VP (passive . +) (expect) (tense . +)) . (PP (passive-pp . +)))
 (connect VP left-features vp-take-modifier))
((QL . ADV) (connect ADV))
((QL . AP) (connect AP))
```

7.4 The Transducer Functions

```
(defphrasemaker sentence (cat feats ds)
  (output-phrase 'ip feats ds))

(defphrasemaker question (cat feats ds)
  ;;
  ;; Build structure for questions of the form "Did John see Mary",
  ;; "Is John seeing Mary"
  ;;
  (output-phrase 'cp
                 (cons '(ques . +) feats)
                 (list (internal-phrase 'c-bar nil (list (first ds)))
                       (internal-phrase 'ip nil (cdr ds)))))

(defphrasemaker vp (cat feats ds)
  ;;
  ;; Builds (I-BAR I-0 (VP V)) structure, and reads in
  ;; subcategorization information.
  ;;
  (let ((v (get-feature 'verb-phrase feats)))
    (let ((subcats (or (get-feature 'int-roles (phrase-features v))
                       ;; Lookup subcategorization from OALDCE
                       (lvs (get-feature 'word (phrase-features v)))
                       '((NP) nil))))
      (let ((int-roles
```

```
              (if (and (eq '+ (get-feature 'passive feats))
                       (eq '+ (get-feature 'is feats)))
                  (mapcar #'(lambda (roles) (cons 'passive-arg (cdr roles)))
                  (remove-if-not #'(lambda (x) (eq 'NP (car x))) subcats))
                                 subcats)))
         (output-phrase
            'i-bar
            (cons `(int-roles . ,int-roles)
                  (cons `(expect . ,(cons 'or (mapcar #'car int-roles)))
                        feats))
            (list (internal-phrase 'i-0 nil (remove v ds))
                  (internal-phrase 'vp nil (list v))))))))

(defphrasemaker vp-take-arg (cat feats ds)
  ;;
  ;; Attaches argument to verb phrase in VP, not IP node.  Also removes one
  ;; argument from subcategorization list.
  ;;
  (let ((arg-type (get-feature 'arg-type feats))
        (old (phrase-daughters (first ds))))
    (let ((ok-subcats (remove-if-not #'(lambda (x) (eq arg-type (base-category x)))
                                     (get-feature 'int-roles feats) :key #'car)))
      (output-phrase 'i-bar
                     (cons `(int-roles . ,(mapcar #'cdr ok-subcats))
                           (cons `(expect . ,(cons 'or (mapcar #'cadr ok-subcats)))
                                 feats))
                     (list (first old)
                           (internal-phrase
                              'vp
                              nil
                              (append (internal-phrase-daughters (second old))
                                      (list (second ds)))))))))

(defphrasemaker vp-take-modifier (cat feats ds)
  ;;
  ;; Attaches modifier arguments to verb phrase in VP, not IP node.
  ;;
  (let ((old (phrase-daughters (first ds))))
    (output-phrase 'i-bar feats
                   (list (internal-phrase
                           'i-0
                           nil
                           (append (internal-phrase-daughters (first old))
                                   (list (second ds))))
                         (second old)))))
```

Appendix C: Compilation Examples

8.1 Examples From the Rule-Action Table

```
(DET1 ((V (ING . +)) (accept DET1))
      ((V (passive . +)) (accept DET1))
      (AP (accept DET1))
      (NUM (accept DET1) (close NP old-first))
      ((PROP (possessive . +)) (close NP old-first))
      (PROP (accept (DET2 (prop))) (close NP old-first))
      ((N (somebody . +)) (accept DET3 old-first) (close (NP (big . +)) old-first))
      ((N (possessive . +) (somebody)) (close (NP (possessive . +)) old-first))
      ((N (pl . +)) (close (NP (det . +) (big . +)) old-first))
      ((N (temporal . +)) (close (NP (big . +)) old-first))
      (N (close (NP (big . +)) old-first) (accept DET2))))
```

The `(V (ING . +))` and `(V (passive . +))` rule-actions:

```
(LAMBDA (PHRASE PREV-PHRASES PREV-FEATURES OLD-PHRASES PSTATE)
  (COND ((PHRASE-HAS-FEATURES PHRASE '((ING . +)))
         (FUNCALL (GET 'DET1 'RULE-STORAGE-FUNCTION) (PHRASE-END PHRASE)
                  (CONS PHRASE PREV-PHRASES) PREV-FEATURES OLD-PHRASES PSTATE))
        ((PHRASE-HAS-FEATURES PHRASE '((PASSIVE . +)))
         (FUNCALL (GET 'DET1 'RULE-STORAGE-FUNCTION) (PHRASE-END PHRASE)
                  (CONS PHRASE PREV-PHRASES) PREV-FEATURES OLD-PHRASES PSTATE))))
```

The `AP` rule action:

```
(LAMBDA (PHRASE PREV-PHRASES PREV-FEATURES OLD-PHRASES PSTATE)
  (FUNCALL (GET 'DET1 'RULE-STORAGE-FUNCTION) (PHRASE-END PHRASE)
           (CONS PHRASE PREV-PHRASES) PREV-FEATURES OLD-PHRASES PSTATE))
```

The `NUM` rule actions:

```
(LAMBDA (PHRASE PREV-PHRASES PREV-FEATURES OLD-PHRASES PSTATE)
  (LET ((NEW-PHRASES OLD-PHRASES))
    (LOWER-PHRASES (CONS PHRASE PREV-PHRASES) PSTATE)
    (LOWER-PHRASES OLD-PHRASES PSTATE)
    (PUSH (MAKE-PHRASE-FROM-RULE PSTATE 10 (APPEND PREV-FEATURES (PHRASE-FEATURES PHRASE))
                                  (CONS PHRASE PREV-PHRASES) 'NP) NEW-PHRASES)
    (FUNCALL (GET 'DET1 'RULE-STORAGE-FUNCTION) (PHRASE-END PHRASE)
             (CONS PHRASE PREV-PHRASES) PREV-FEATURES NEW-PHRASES PSTATE)))
```

The `(PROP (possessive . +))` and `PROP` rule-actions:

```
(LAMBDA (PHRASE PREV-PHRASES PREV-FEATURES OLD-PHRASES PSTATE)
  (COND ((PHRASE-HAS-FEATURES PHRASE '((POSSESSIVE . +)))
         (LOWER-PHRASES (CONS PHRASE PREV-PHRASES) PSTATE)
         (LOWER-PHRASES OLD-PHRASES PSTATE)
         (MAKE-PHRASE-FROM-RULE PSTATE 10 (APPEND PREV-FEATURES (PHRASE-FEATURES PHRASE))
                                (CONS PHRASE PREV-PHRASES) 'NP))
        (T (LET ((NEW-PHRASES OLD-PHRASES))
             (LOWER-PHRASES (CONS PHRASE PREV-PHRASES) PSTATE)
             (LOWER-PHRASES OLD-PHRASES PSTATE)
             (PUSH (MAKE-PHRASE-FROM-RULE PSTATE 10 (APPEND PREV-FEATURES
                                                            (PHRASE-FEATURES PHRASE))
                                          (CONS PHRASE PREV-PHRASES) 'NP)
                   NEW-PHRASES)
             (FUNCALL (GET 'DET2 'RULE-STORAGE-FUNCTION) (PHRASE-END PHRASE)
                      (CONS PHRASE PREV-PHRASES) (CONS (CONS 'PROP 'NIL) PREV-FEATURES)
                      NEW-PHRASES PSTATE)))))
```

The various `N` rule-actions:

```
(LAMBDA (PHRASE PREV-PHRASES PREV-FEATURES OLD-PHRASES PSTATE)
  (COND ((PHRASE-HAS-FEATURES PHRASE '((SOMEBODY . +)))
         (LET ((NEW-PHRASES OLD-PHRASES))
           (LOWER-PHRASES (CONS PHRASE PREV-PHRASES) PSTATE)
           (LOWER-PHRASES OLD-PHRASES PSTATE)
           (PUSH (MAKE-PHRASE-FROM-RULE PSTATE 10
                                        (CONS (CONS 'BIG '+)
                                              (APPEND PREV-FEATURES (PHRASE-FEATURES PHRASE)))
                                        (CONS PHRASE PREV-PHRASES) 'NP)
                 NEW-PHRASES)
           (FUNCALL (GET 'DET3 'RULE-STORAGE-FUNCTION) (PHRASE-END PHRASE)
                    (CONS PHRASE PREV-PHRASES) (APPEND PREV-FEATURES (PHRASE-FEATURES PHRASE))
                    NEW-PHRASES PSTATE)))
        ((PHRASE-HAS-FEATURES PHRASE '((POSSESSIVE . +) (SOMEBODY)))
         (LOWER-PHRASES (CONS PHRASE PREV-PHRASES) PSTATE)
         (LOWER-PHRASES OLD-PHRASES PSTATE)
         (MAKE-PHRASE-FROM-RULE PSTATE 10 (CONS (CONS 'POSSESSIVE '+)
                                                (APPEND PREV-FEATURES (PHRASE-FEATURES PHRASE)))
                                (CONS PHRASE PREV-PHRASES) 'NP))
        ((PHRASE-HAS-FEATURES PHRASE '((PL . +)))
         (LOWER-PHRASES (CONS PHRASE PREV-PHRASES) PSTATE)
         (LOWER-PHRASES OLD-PHRASES PSTATE)
         (MAKE-PHRASE-FROM-RULE PSTATE 10
                                (CONS (CONS 'BIG '+) (CONS (CONS 'DET '+)
```

```
                              (APPEND PREV-FEATURES (PHRASE-FEATURES PHRASE))))
                          (CONS PHRASE PREV-PHRASES) 'NP))
      ((PHRASE-HAS-FEATURES PHRASE '((TEMPORAL . +)))
       (LOWER-PHRASES (CONS PHRASE PREV-PHRASES) PSTATE)
       (LOWER-PHRASES OLD-PHRASES PSTATE)
       (MAKE-PHRASE-FROM-RULE PSTATE 10
                          (CONS (CONS 'BIG '+)
                                (APPEND PREV-FEATURES (PHRASE-FEATURES PHRASE)))
                          (CONS PHRASE PREV-PHRASES) 'NP))
      (T (LET ((NEW-PHRASES OLD-PHRASES))
           (LOWER-PHRASES (CONS PHRASE PREV-PHRASES) PSTATE)
           (LOWER-PHRASES OLD-PHRASES PSTATE)
           (PUSH (MAKE-PHRASE-FROM-RULE PSTATE 10
                                    (CONS (CONS 'BIG '+)
                                          (APPEND PREV-FEATURES (PHRASE-FEATURES PHRASE)))
                                    (CONS PHRASE PREV-PHRASES) 'NP)
                 NEW-PHRASES)
           (FUNCALL (GET 'DET2 'RULE-STORAGE-FUNCTION) (PHRASE-END PHRASE)
                    (CONS PHRASE PREV-PHRASES) PREV-FEATURES NEW-PHRASES PSTATE)))))
```

And the rule-storage-function that places all of the above rules in a table.

```
(LAMBDA (N PREV-PHRASES PREV-FEATURES NEW-PHRASES PSTATE)
  (COND ((NOT (MEMBER PREV-PHRASES
                 (AREF (CDR (PSTATE-STATES PSTATE)) N (GET 'DET1 'STATE-NO)) :TEST #'EQUAL))
         (PUSH PREV-PHRASES (AREF (CDR (PSTATE-STATES PSTATE)) N (GET 'DET1 'STATE-NO)))
         (LET ((NEW-RULE
                (MAKE-RULE :PREV-PHRASES PREV-PHRASES :PREV-FEATURES PREV-FEATURES
                           :LOC N :CAT-EXPECTED 141 :FN #<DTP-COMPILED-FUNCTION V-ACTION>
                           :NEW-PHRASES NEW-PHRASES :STATE-NAME 'DET1)))
           (INTRODUCE-RULE NEW-RULE PSTATE))
         (LET ((NEW-RULE
                (MAKE-RULE :PREV-PHRASES PREV-PHRASES :PREV-FEATURES PREV-FEATURES
                           :LOC N :CAT-EXPECTED 15 :FN #<DTP-COMPILED-FUNCTION AP-ACTION>
                           :NEW-PHRASES NEW-PHRASES :STATE-NAME 'DET1)))
           (INTRODUCE-RULE NEW-RULE PSTATE))
         (LET ((NEW-RULE
                (MAKE-RULE :PREV-PHRASES PREV-PHRASES :PREV-FEATURES PREV-FEATURES
                           :LOC N :CAT-EXPECTED 135 :FN #<DTP-COMPILED-FUNCTION NUM-ACTION>
                           :NEW-PHRASES NEW-PHRASES :STATE-NAME 'DET1)))
           (INTRODUCE-RULE NEW-RULE PSTATE))
         (LET ((NEW-RULE
                (MAKE-RULE :PREV-PHRASES PREV-PHRASES :PREV-FEATURES PREV-FEATURES
                           :LOC N :CAT-EXPECTED 136 :FN #<DTP-COMPILED-FUNCTION PROP-ACTION>
                           :NEW-PHRASES NEW-PHRASES :STATE-NAME 'DET1)))
           (INTRODUCE-RULE NEW-RULE PSTATE))
         (LET ((NEW-RULE
                (MAKE-RULE :PREV-PHRASES PREV-PHRASES :PREV-FEATURES PREV-FEATURES
                           :LOC N :CAT-EXPECTED 142 :FN #<DTP-COMPILED-FUNCTION N-ACTION>
                           :NEW-PHRASES NEW-PHRASES :STATE-NAME 'DET1)))
           (INTRODUCE-RULE NEW-RULE PSTATE)))))
```

8.2 Examples From the Single-Phrase Action Table

```
((DET (PRO . +))
 (start-rule DET0)
 (project (NP (big . -)) new-only))
(DET (start-rule DET0))
(NUM (start-rule DET1))
```

The `DET` single-phrase action:

```
(LAMBDA (PHRASE N PSTATE)
  (COND ((PHRASE-HAS-FEATURES PHRASE '((PRO . +)))
         (FUNCALL (GET 'DET0 'RULE-STORAGE-FUNCTION) N NIL NIL NIL PSTATE)
         (LOWER-PHRASE PHRASE PSTATE)
         (MAKE-PHRASE-FROM-RULE PSTATE 10 (CONS (CONS 'BIG '-) (PHRASE-FEATURES PHRASE))
                                (LIST PHRASE) 'NP))
        (T (FUNCALL (GET 'DET0 'RULE-STORAGE-FUNCTION) N NIL NIL NIL PSTATE))))
```

The `NUM` single-phrase action:

```
(LAMBDA (PHRASE N PSTATE) (FUNCALL (GET 'DET1 'RULE-STORAGE-FUNCTION) N NIL NIL NIL PSTATE))
```

8.3 Examples From the Paired-Phrase Action Table

```
(((IS (tense . +) (pl . ?a)) . (NP (pl . ?a)) ) (start-rule Q0 nil left))
(((NP (wh . +)) . (VP (expect . NP) (to . +) (has-trace)))
 (insert (NP (trace . +) (has-trace . +) (bound-to . l-phrase)) nil right))
(((VP (expect . (CP (adverbial . +)))) . (CP (has-trace . ?a) (adverbial . +)))
 (connect (VP (has-trace . ?a) (arg-type . cp)) left-features vp-take-arg))
(((P (takes-vp . +)) . VP) (connect (PP (assigns-case . acc)) left-right))
```

The `IS NP` paired-phrase action:

```
(LAMBDA (L-PHRASE R-PHRASE PSTATE)
  (COND ((PHRASE-HAS-FEATURES L-PHRASE '((TENSE . +)))
         (LET ((L-FEATS (PHRASE-FEATURES L-PHRASE))
               (R-FEATS (PHRASE-FEATURES R-PHRASE)))
           (LET ((?A (GET-FEATURE 'PL L-FEATS)))
             (COND ((AGREEMENT-EQUAL ?A (GET-FEATURE 'PL R-FEATS))
                    (FUNCALL (GET 'Q0 'RULE-STORAGE-FUNCTION) (PHRASE-START L-PHRASE)
                             NIL NIL NIL PSTATE)))))))
```

The `NP VP` paired-phrase action:

```
(LAMBDA (L-PHRASE R-PHRASE PSTATE)
  (COND ((AND (PHRASE-HAS-FEATURES L-PHRASE '((WH . +)))
              (PHRASE-HAS-FEATURES R-PHRASE '((EXPECT . NP) (TO . +) (HAS-TRACE))))
         (INSERT-ONTO PSTATE R-PHRASE
                      (MAKE-PHRASE :CAT 10 :FEATURES
                                   (CONS (CONS 'BOUND-TO L-PHRASE)
                                         (CONS (CONS 'HAS-TRACE '+)
                                               (CONS (CONS 'TRACE '+) NIL)))
                                   :START (PHRASE-END R-PHRASE) :END (PHRASE-END R-PHRASE)
                                   :CATEGORY 'NP)))))
```

The `VP CP` paired-phrase action:

```
(LAMBDA (L-PHRASE R-PHRASE PSTATE)
  (COND ((AND (PHRASE-HAS-FEATURES L-PHRASE '((EXPECT CP (ADVERBIAL . +))))
              (PHRASE-HAS-FEATURES R-PHRASE '((ADVERBIAL . +))))
         (LET ((R-FEATS (PHRASE-FEATURES R-PHRASE)))
           (LET ((?A (GET-FEATURE 'HAS-TRACE R-FEATS)))
             (LOWER-PHRASE L-PHRASE PSTATE)
             (LOWER-PHRASE R-PHRASE PSTATE)
             (PHRASEMAKER-VP-TAKE-ARG PSTATE 138
              (CONS (CONS 'ARG-TYPE 'CP)
                    (IF ?A (CONS (CONS 'HAS-TRACE ?A) (PHRASE-FEATURES L-PHRASE))
                             (PHRASE-FEATURES L-PHRASE)))
              (LIST R-PHRASE L-PHRASE) 'VP))))))
```

The `P VP` paired-phrase action:

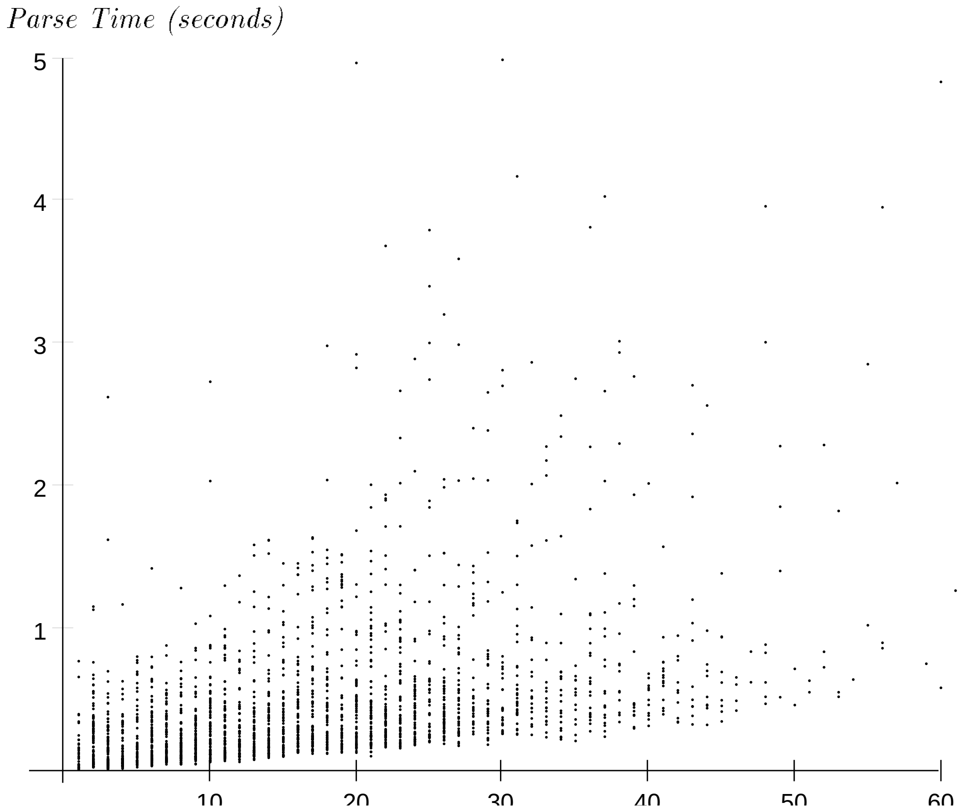
```
(LAMBDA (L-PHRASE R-PHRASE PSTATE)
  (COND ((PHRASE-HAS-FEATURES L-PHRASE '((TAKES-VP . +)))
         (LOWER-PHRASE L-PHRASE PSTATE)
         (LOWER-PHRASE R-PHRASE PSTATE)
         (MAKE-PHRASE-FROM-RULE PSTATE 145
                                (CONS (CONS 'ASSIGNS-CASE 'ACC)
                                      (APPEND (PHRASE-FEATURES L-PHRASE)
                                              (PHRASE-FEATURES R-PHRASE)))
                                (LIST R-PHRASE L-PHRASE) 'PP))))
```

Appendix D: Parser Statistics

The following graphs represent statistics derived from two runs of the parser. The first, used in Graphs 1 through 10, was over 1300 sentences selected randomly from throughout the LOB. The second, used in Graphs 11 through 14, was on a 600 sentences subset of the 1300 and was performed without any lowering.

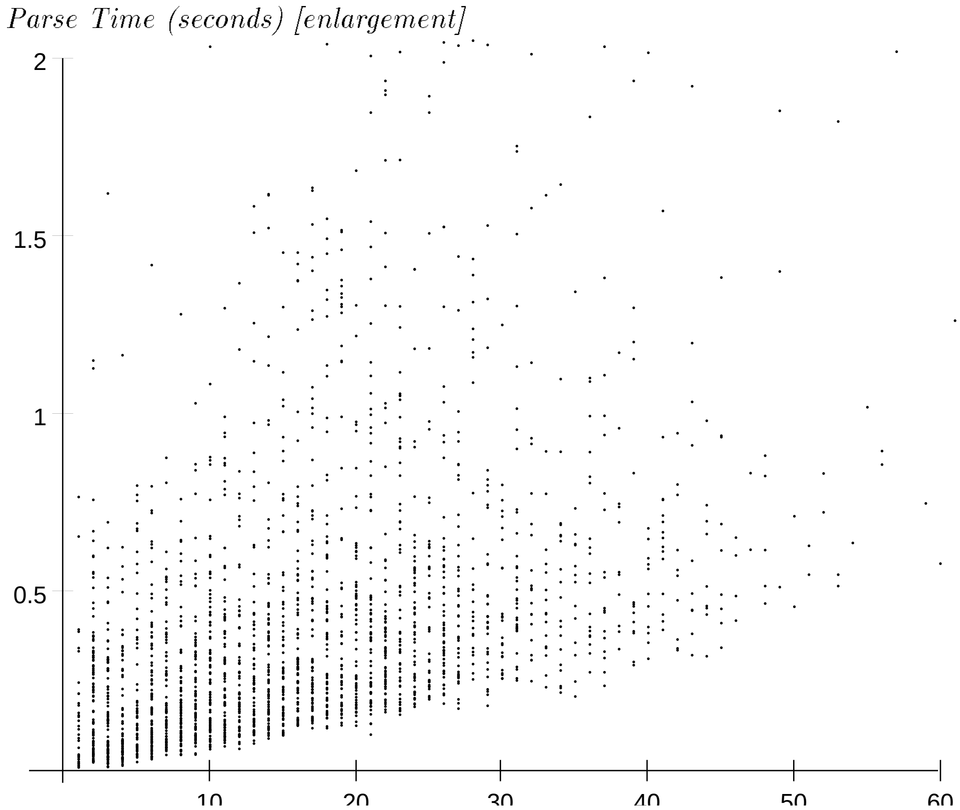
All code is pure Common Lisp, running on a Symbolics 3600 series Lisp Machine. As much effort as possible was expended to make the timing statistics independent of garbage collection and other background processes, but the reported parse times of individual sentence can still vary significantly even if parsed back to back.

So as to provide results more directly comparable to those generally presented in the field, punctuation marks not combined into larger phrases in the output parse have been subtracted from the total sentence length, number of output phrases, and total numbers of items and phrases created.



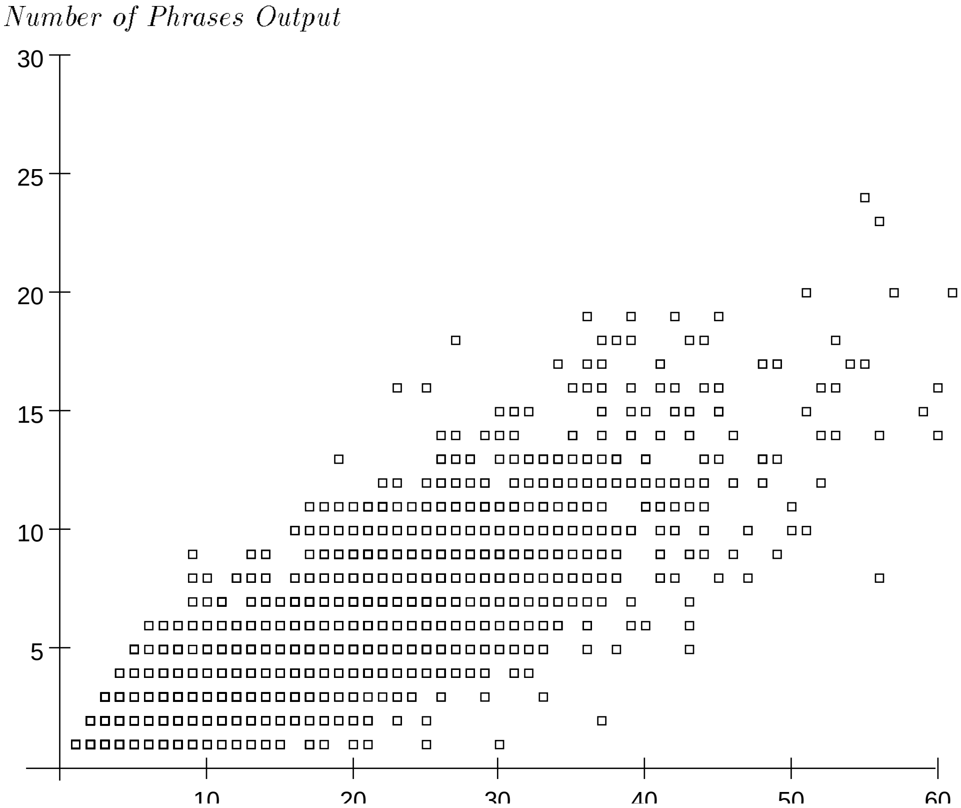
Graph 1: *Number of Words in Sentence*

The average time spent by the parser on each word is .039 seconds, with a standard deviation of .07 seconds. The average time per sentence is .56 seconds (average sentence length of 17.4 words).



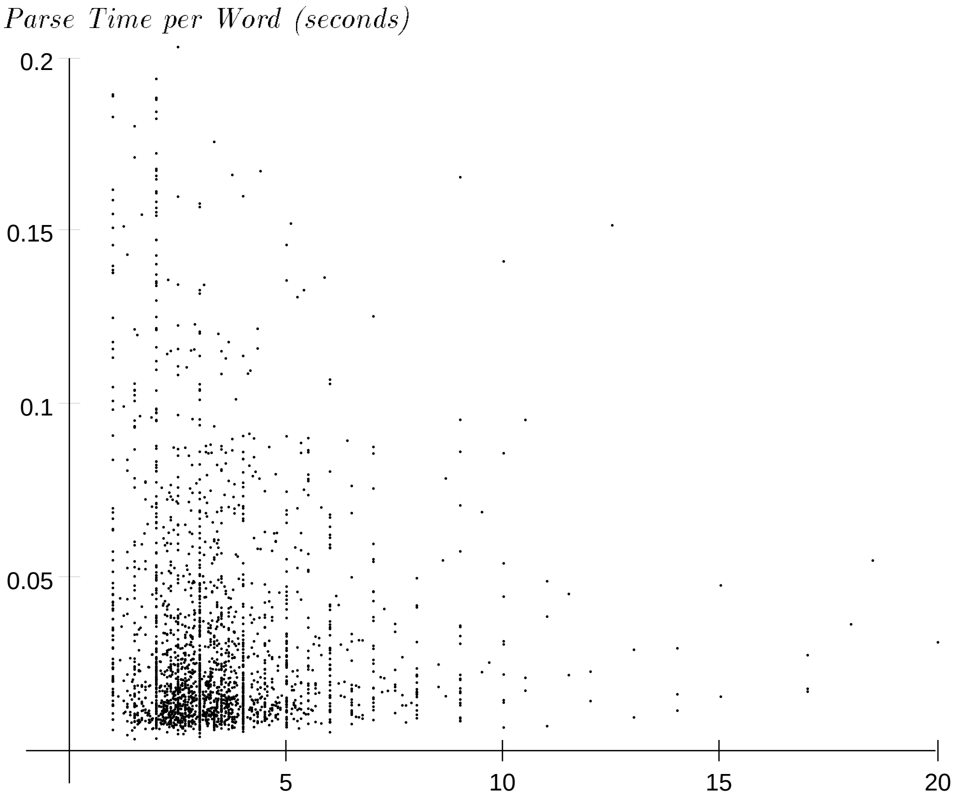
Graph 2: *Number of Words in Sentence*

Graph 2 is a blowup of Graph 1.



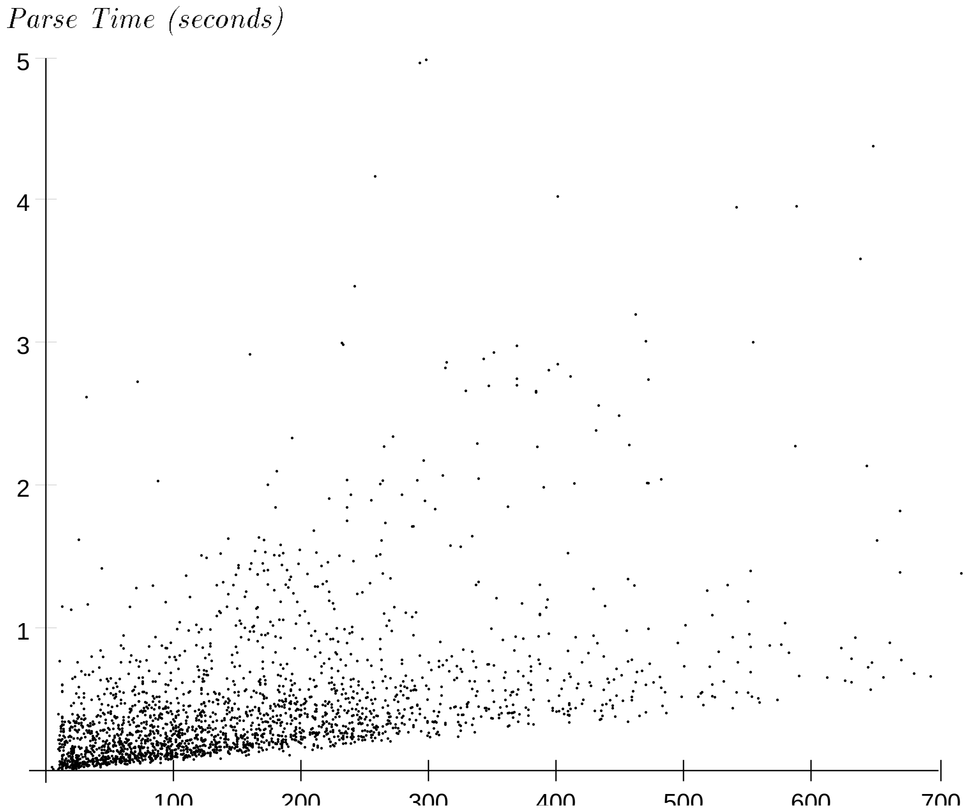
Graph 3: *Number of Words in Sentence*

Graph 3 provides a feeling for how well the parser breaks up sentences. Unfortunately, many data points overlay each other. The average length of a phrase produced is 3.51 words, with a standard deviation of 2.08 words.



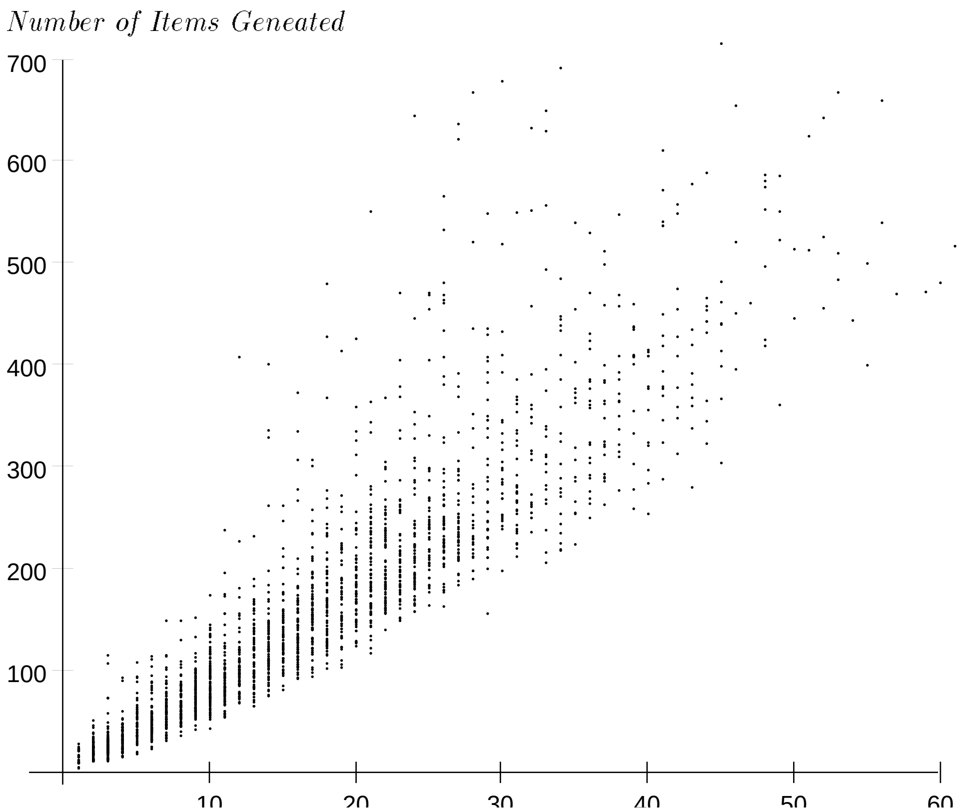
Graph 4: *Averge Length of Output Phrase in Sentence*

Graph 4 shows that the parser does not necessarily expend any great amount more effort per word in sentences which it breaks up into large pieces.



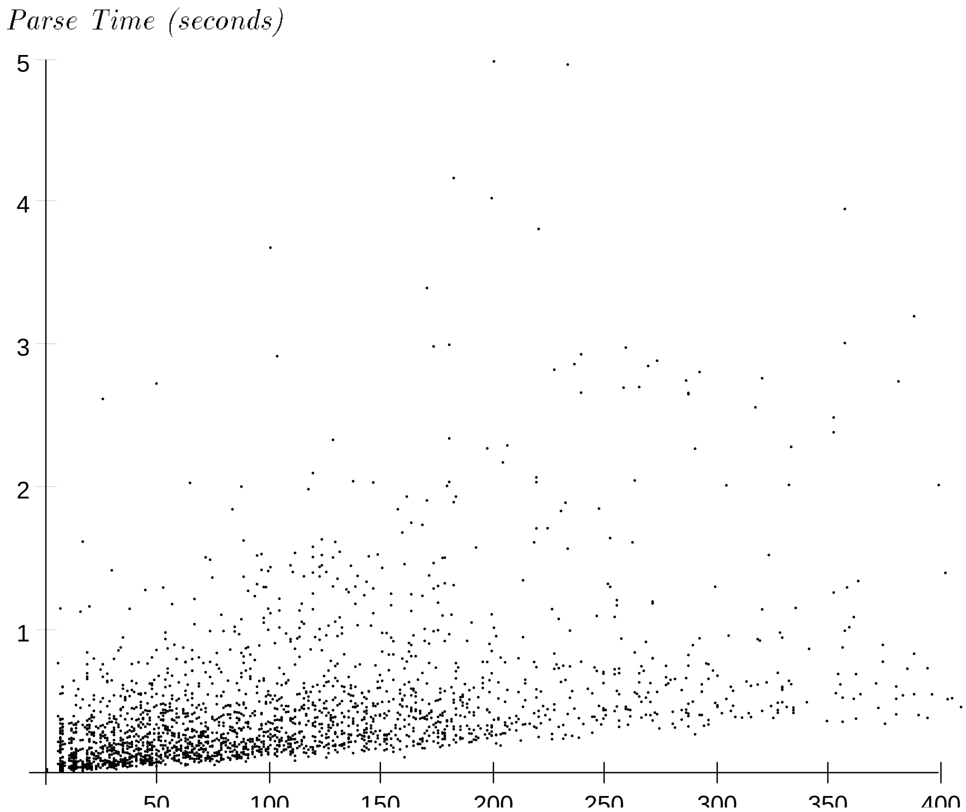
Graph 5: *Number of Items Generated*

In Graphs 5 and 6, the number of items generated refers to the total number of rule states and phrases generated over the parse.

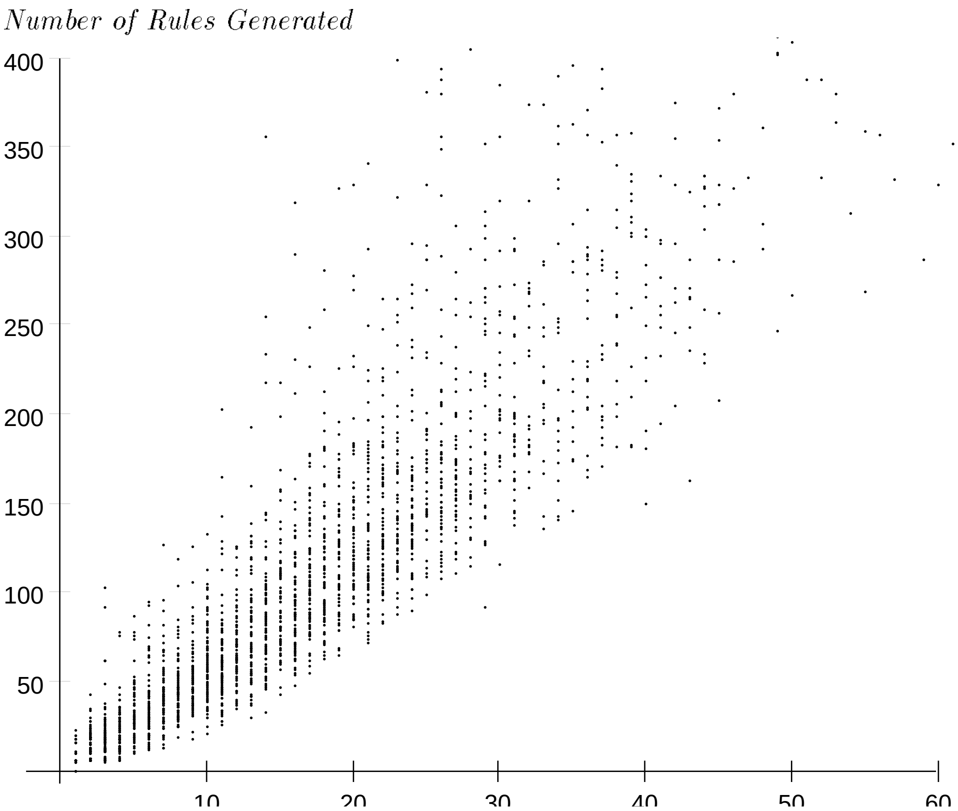


Graph 6: *Number of Words in Sentence*

A polynomial fit of Graph 6 gives $i = 8.36n^{1.038}$, essentially a linear fit.

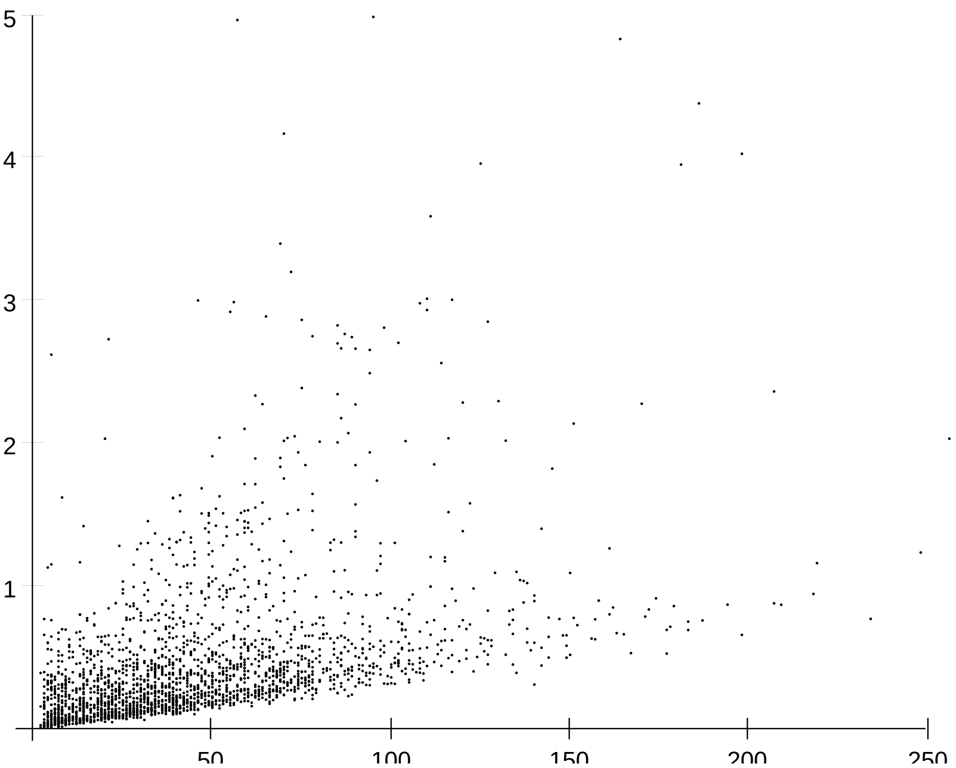


Graph 7: *Number of Rules Generated*

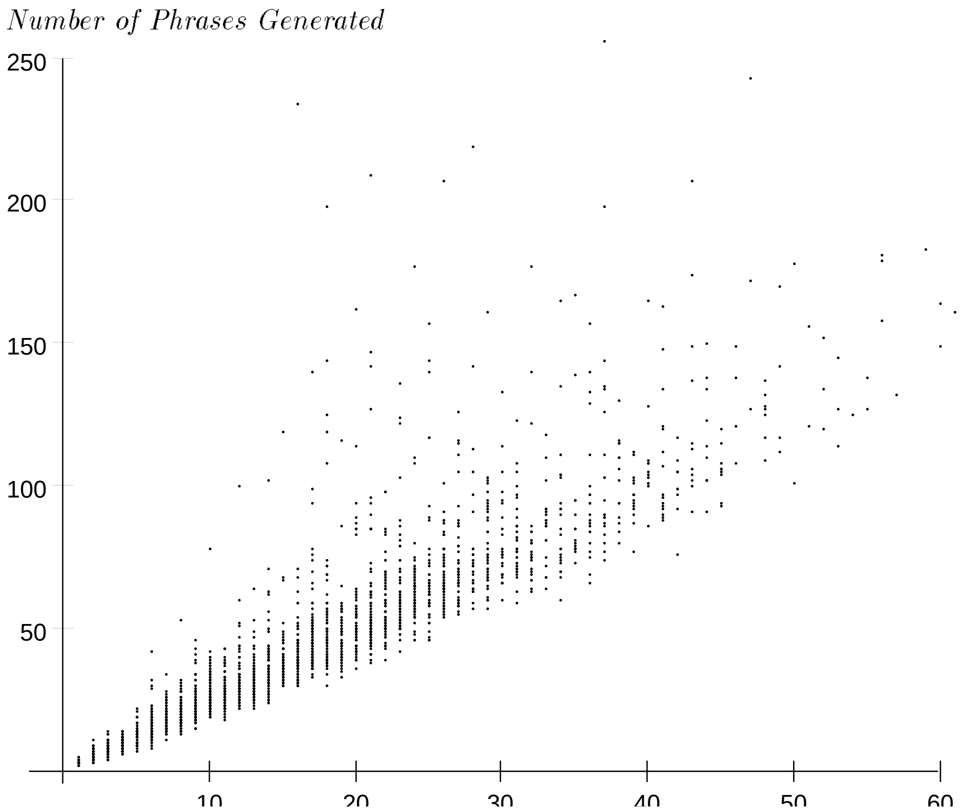


Graph 8: *Number of Words in Sentence*

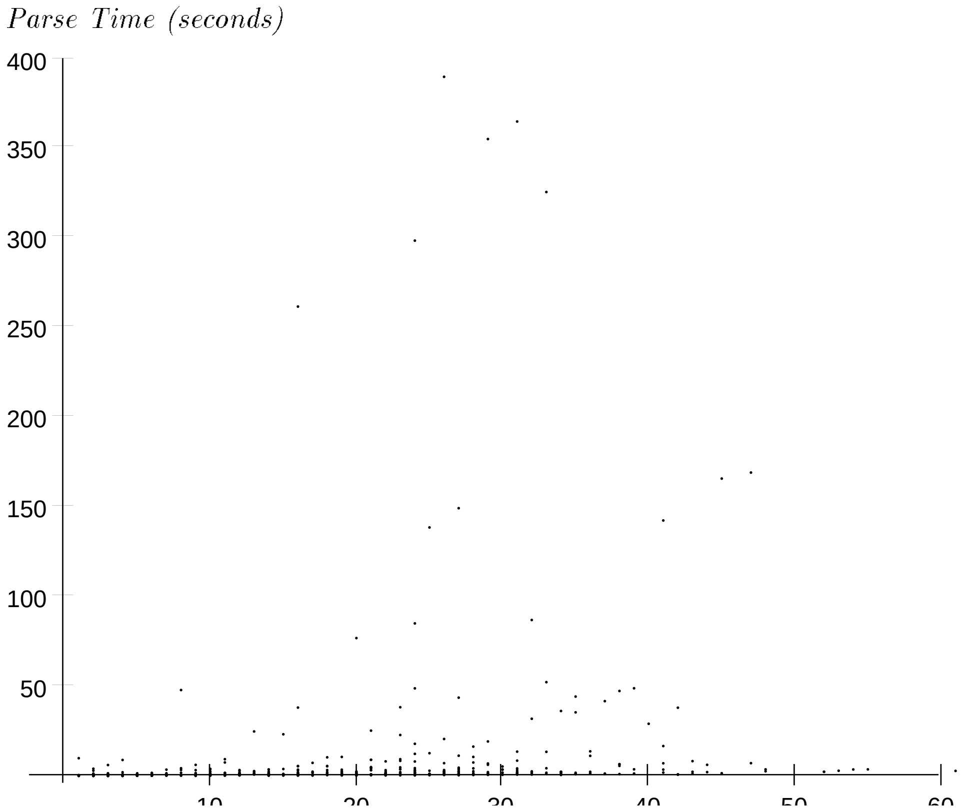
Parse Time (seconds)



Graph 9: *Number of Phrases Generated*

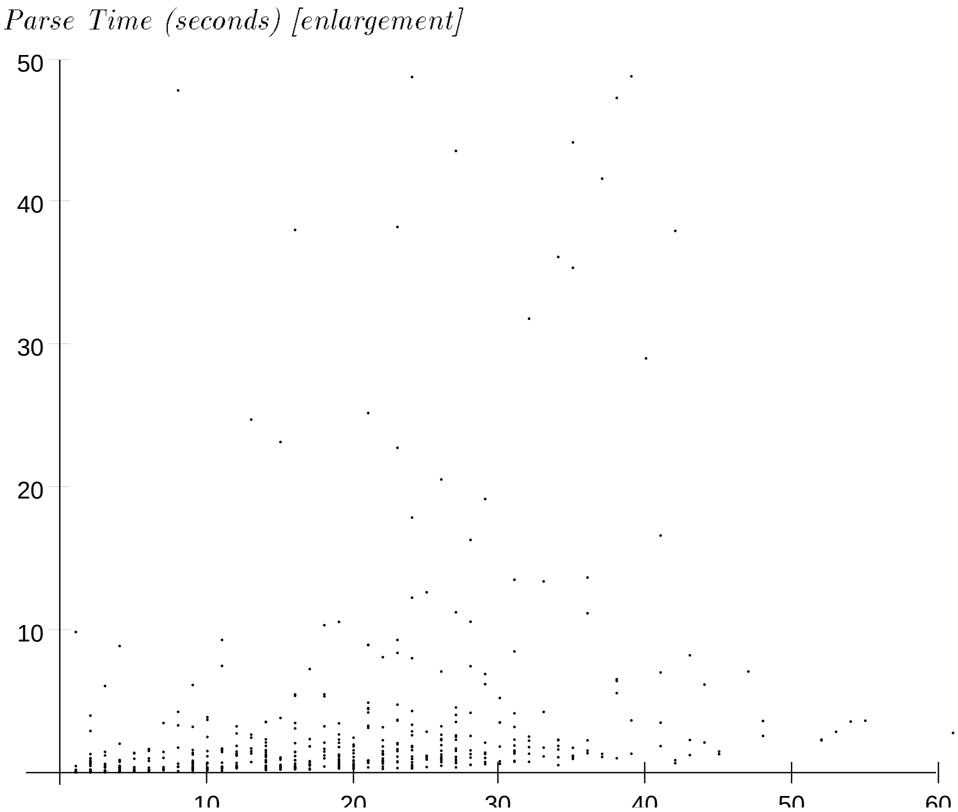


Graph 10: *Number of Words in Sentence*

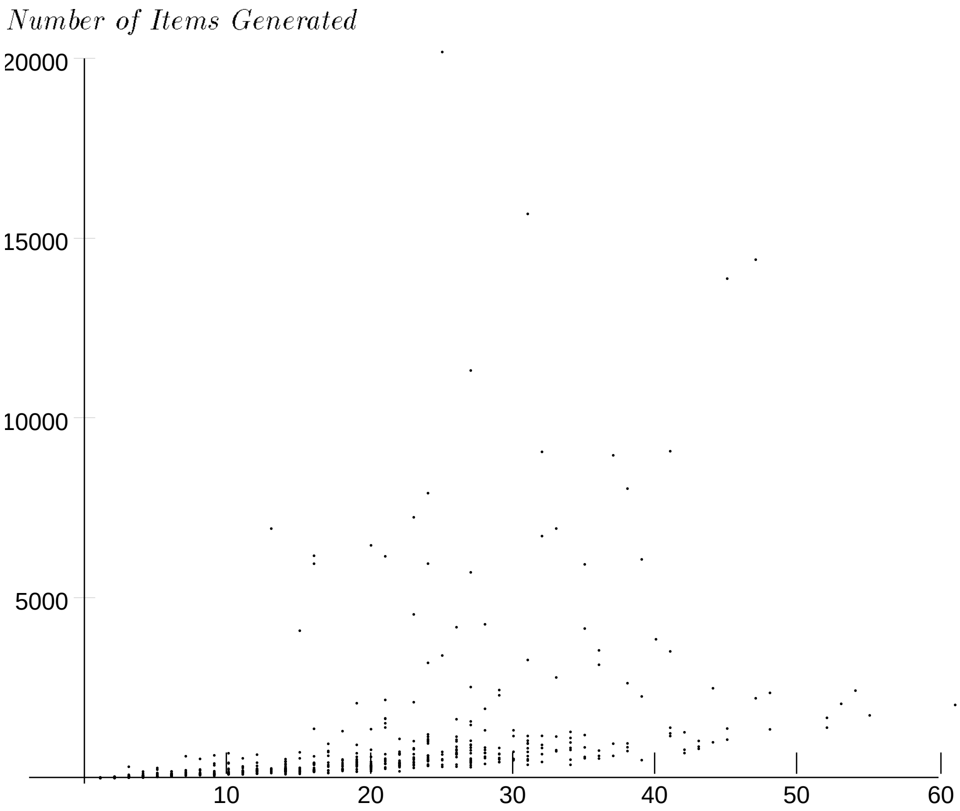


Graph 11: *Number of Words in Sentence*

Graphs 11 through 14 represent the parser run on 600 sentences without any lowering of phrases taking place. The average time per word drastically increases to .6 seconds with a standard deviation of 4.5 seconds. The average total time per sentence increases to 12.7 seconds.



Graph 12: *Number of Words in Sentence*



Graph 13: *Number of Words in Sentence*

Graph 14: *Number of Words in Sentence*

Appendix E: Sample Parses

The following parses are unretouched and unedited. All were parsed with the grammar presented in Appendix B.

10.1 Examples from Harrison's Thesis

The following sentences are the first 20 (of 75) test sentences from [Harrison 1988]. Presented with the sentences are the times Harrison's parser took, and the number of parses his parser produced. In some cases the second number is enormous. Two periods, .. indicate that the trees shown do not reflect parenthesis structure because of space constraints.

Outside of for a small few specially defined words not found in the LOB, all words were looked up in the LOB dictionary, and disambiguated.

```
1. The assembly language provides a means for writing a program without
   having to be concerned with actual memory addresses . (4.41 secs, 1560 parses)

> (IP (NP (DET "The") (N "assembly") (N "language"))
      (I-BAR (I-0)
             (VP (V "provides") (NP (DET "a") (N "means"))
                 (PP (P "for") (NP (I-BAR (I-0) (VP (V "writing")
                                                    (NP (DET "a") (N "program")))))))))
> (P "without")
> (I-BAR (I-0) (VP (HAVE "having")))
> (I-BAR (I-0 (TO "to") (IS "be")) (VP (V "concerned")))
> (PP (P "with") (NP (AP (JJ "actual")) (N "memory") (N "addresses")))
> (*PER ".")

Time: 0.184 seconds.

2. Labels can be assigned to a particular instruction step in a source
   program to identify that step as an entry point for use in subsequent
   instructions . (1.61 secs, 34 parses)

> (IP (NP (N "Labels"))
      (I-BAR (I-0 (MD "can") (IS "be"))
```

```
            (VP (V "assigned") (NP)
                (PP (P "to") (NP (DET "a") (AP (JJ "particular"))
                             (N "instruction") (N "step"))))))
> (PP (P "in") (NP (DET "a") (N "source") (N "program")))
> (I-BAR (I-0 (TO "to"))
        (VP (V "identify") (NP (DET "that") (N "step"))
            (PP (P "as") (NP (DET "an") (N "entry") (N "point")))))
> (PP (P "for") (NP (N "use")))
> (PP (P "in") (NP (AP (JJ "subsequent")) (N "instructions")))
> (*PER ".")

Time: 0.207 seconds.

3. The assembly language also includes assembler directives that
   supplement the machine instructions . (.33 secs, 1 parse)

> (IP (NP (DET "The") (N "assembly") (N "language"))
      (I-BAR (I-0 (ADV "also"))
            (VP (V "includes")
                (NP (NP (N "assembler") (N "directives"))
                    (CP (COMP "that")
                        (IP (NP)
                            (I-BAR (I-0) (VP (V "supplement")
                                             (NP (DET "the") (N "machine")
                                                 (N "instructions")))))))))
> (*PER ".")

Time: 0.107 seconds.

4. A pseudo-op is a statement which is not translated into a machine
   instruction . (.6 secs, 1 parse)

> (IP (NP (DET "A") (N "pseudo-op"))
      (I-BAR (I-0)
            (VP (IS "is")
                (NP (NP (DET "a") (N "statement"))
                    (CP (COMP "which")
                        (IP (NP)
                            (I-BAR (I-0 (IS "is") (XNOT "not")) ..
        (VP (V "translated") (NP)
            (PP (P "into") (NP (DET "a") (N "machine") (N "instruction"))))))))))
> (*PER ".")

Time: 0.116 seconds.

5. Each statement is written on a single line , and it may consist of
   four entries . (.38 secs, 1 parse)

> (IP (NP (DET "Each") (N "statement")) (I-BAR (I-0 (IS "is")) (VP (V "written") (NP))))
> (PP (P "on") (NP (DET "a") (AP (JJ "single")) (N "line")))
> (*CMA ",")
> (CC "and")
> (IP (NP (PRO "it"))
      (I-BAR (I-0 (MD "may"))
            (VP (V "consist") (PP (P "of") (NP (NUM (CD "four")) (N "entries"))))))
> (*PER ".")

Time: 0.097 seconds.
```

```
6. The source program is processed by the assembler to obtain a machine
   language . (.56 secs, 4 parses)

> (IP (NP (DET "The") (N "source") (N "program"))
      (I-BAR (I-0 (IS "is") (PP (P "by") (NP (DET "the") (N "assembler"))))
             (VP (V "processed") (NP))))
> (I-BAR (I-0 (TO "to")) (VP (V "obtain") (NP (DET "a") (N "machine") (N "language"))))
> (*PER ".")

Time: 0.101 seconds.

7. The packet transport mechanism provided by Ethernet has been used to
   build systems which can be local computer networks . (.66 secs, 2 parses)

> (IP (NP (NP (DET "The") (N "packet") (N "transport") (N "mechanism"))
          (I-BAR (I-0) (VP (V "provided") (PP (P "by") (NP (PROP (NAME "Ethernet")))))))
      (I-BAR (I-0 (HAVE "has") (IS "been"))
             (VP (V "used") (NP)
                 (I-BAR (I-0 (TO "to"))
                        (VP (V "build") ..
        (NP (NP (N "systems"))
            (CP (COMP "which")
                (IP (NP) (I-BAR (I-0 (MD "can")) (VP (IS "be") (AP (JJ "local")))))))
        (NP (N "computer") (N "networks")))))))
> (*PER ".")

Time: 0.327 seconds.

8. Switching of packets to their destinations on the Ether is distributed
   among the receiving stations using packet address recognition . (.96 secs, 8 parses)

> (NP (NP (N "Switching")) (PP (P "of") (NP (N "packets"))))
> (PP (P "to") (NP (DET "their") (N "destinations")))
> (PP (P "on") (NP (DET "the") (N "Ether")))
> (I-BAR (I-0 (IS "is"))
         (VP (V "distributed") (NP)
             (PP (P "among")
                 (NP (NP (DET "the") (N "receiving") (N "stations"))
                     (I-BAR (I-0) (VP (V "using") ..
                                (NP (N "packet") (N "address") (N "recognition"))))))))
> (*PER ".")

Time: 0.137 seconds.

9. A model for estimating performance under heavy loads is included for
   completeness . (.61 secs, 4 parses)

> (NP (DET "A") (N "model"))
> (PP (P "for")
      (IP (NP (I-BAR (I-0)
                     (VP (V "estimating") (NP (N "performance"))
                         (PP (P "under") (NP (AP (JJ "heavy")) (N "loads"))))))
          (I-BAR (I-0 (IS "is")) (VP (V "included") (NP)))))
> (PP (P "for") (NP (N "completeness")))
> (*PER ".")

Time: 0.089 seconds.

10. It is a text book for students who are beginning graduate work in
```

```
    computer science . (1.14 secs, 9 parses)

> (IP (NP (PRO "It")) (I-BAR (I-0) (VP (IS "is") (NP (DET "a") (N "text") (N "book")))))
> (PP (P "for")
      (NP (NP (N "students"))
          (CP (COMP "who")
              (IP (NP) (I-BAR (I-0 (IS "are"))
                              (VP (V "beginning") (NP (N "graduate") (N "work"))))))))
> (PP (P "in") (NP (N "computer") (N "science")))
> (*PER ".")

Time: 0.129 seconds.

11. It includes exercises designed to help the student master a body of
    techniques . (.45 secs, 4 parses)

> (IP (NP (PRO "It"))
      (I-BAR (I-0) (VP (V "includes")
                       (NP (NP (N "exercises")) (I-BAR (I-0) (VP (V "designed")))))))
> (I-BAR (I-0 (TO "to")) (VP (V "help") (NP (DET "the") (N "student") (N "master"))))
> (NP (NP (DET "a") (N "body")) (PP (P "of") (NP (N "techniques"))))
> (*PER ".")

Time: 0.178 seconds.

12. It is a practical guide for people who are building computer systems
    that deal with natural language . (1.26 secs, 30 parses)
> (IP (NP (PRO "It"))
      (I-BAR (I-0) (VP (IS "is") (NP (DET "a") (AP (JJ "practical")) (N "guide")))))
> (PP (P "for")
      (NP (NP (N "people"))
          (CP (COMP "who")
              (IP (NP)
                  (I-BAR (I-0 (IS "are")) ..
     (VP (V "building")
         (NP (NP (N "computer") (N "systems"))
             (CP (COMP "that") (IP (NP) (I-BAR (I-0) (VP (V "deal")))))
         (PP (P "with") (NP (AP (JJ "natural")) (N "language")))))))))
> (*PER ".")

Time: 0.228 seconds.

13. It is not structured as a how-to book , but it describes the relevant techniques in
    detail, and it includes an extensive outline of English grammar . (.76 secs, 2 parses)

> (IP (NP (PRO "It")) (I-BAR (I-0 (XNOT "not")) (VP (IS "is") (AP (JJ "structured")))))
> (PP (P "as") (NP (DET "a") (AP (JJ "how-to")) (N "book")))
> (*CMA ",")
> (CC "but")
> (IP (NP (PRO "it"))
      (I-BAR (I-0)
             (VP (V "describes")
                 (NP (DET "the") (AP (JJ "relevant")) (N "techniques"))
                 (PP (P "in") (NP (N "detail"))))))
> (*CMA ",")
> (CC "and")
> (IP (NP (PRO "it"))
      (I-BAR (I-0)
             (VP (V "includes")
```

```
                 (NP (NP (DET "an") (AP (JJ "extensive")) (N "outline"))
                     (PP (P "of") (NP (AP (JJ "English")) (N "grammar")))))))
> (*PER ".")

Time: 0.205 seconds.

14. It is a reference source with many pointers into the literature of
    linguistics . (.43 secs, 3 parses)

> (IP (NP (PRO "It"))
      (I-BAR (I-0) (VP (IS "is") (NP (DET "a") (N "reference") (N "source")))))
> (PP (P "with") (NP (AP "many") (N "pointers")))
> (PP (P "into") (NP (NP (DET "the") (N "literature"))
                     (PP (P "of") (NP (N "linguistics")))))
> (*PER ".")

Time: 0.083 seconds.

15. I have attempted to introduce a wide variety of material to provide newcomers with
    broad access to the field . (.78 secs, 13 parses)

> (IP (NP (PRO "I"))
      (I-BAR (I-0 (HAVE "have"))
             (VP (V "attempted")
                 (I-BAR (I-0 (TO "to"))
                        (VP (V "introduce")
                            (NP (DET "a") (AP (JJ "wide")) (N "variety"))
                            (PP (P "of") (NP (N "material"))))))))
> (I-BAR (I-0 (TO "to"))
         (VP (V "provide") (NP (N "newcomers"))
             (PP (P "with") (NP (AP (JJ "broad")) (N "access")))))
> (PP (P "to") (NP (DET "the") (N "field")))
> (*PER ".")

Time: 0.223 seconds.

16. Each chapter includes suggestions for further reading , and there is an extensive
    bibliography . (.61 secs, 6 parses)

> (IP (NP (DET "Each") (N "chapter"))
      (I-BAR (I-0) (VP (V "includes") (NP (N "suggestions")))))
> (PP (P "for") (NP (AP (JJ "further")) (N "reading")))
> (*CMA ",")
> (CC "and")
> (IP (NP (PRO "there"))
      (I-BAR (I-0) (VP (IS "is")
                       (NP (DET "an") (AP (JJ "extensive")) (N "bibliography")))))
> (*PER ".")

Time: 0.104 seconds.

17. However , I have tried to limit the references to easily available
    material . (.28 secs, 2 parses)

> (ADV "However")
> (*CMA ",")
> (IP (NP (PRO "I")) (I-BAR (I-0 (HAVE "have")) (VP (V "tried"))))
> (I-BAR (I-0 (TO "to"))
             (VP (V "limit") (NP (DET "the") (N "references"))
```

```
                 (PP (P "to")
                     (NP (AP (ADV "easily") (JJ "available")) (N "material")))))
> (*PER ".")

Time: 0.675 seconds.
```

18. In looking at language as a cognitive process , we deal with issues that have been the focus of linguistic study for many years , and this book includes insights gained from these studies . (3.18 secs, 80 parses)

```
> (PP (P "In") (NP (I-BAR (I-0) (VP (V "looking") (PP (P "at") (NP (N "language")))))))
> (PP (P "as") (NP (DET "a") (AP (JJ "cognitive")) (N "process")))
> (*CMA ",")
> (IP (NP (PRO "we"))
      (I-BAR (I-0) ..
         (VP (V "deal")
             (PP (P "with")
                 (NP (NP (NP (N "issues"))
                         (CP (COMP "that")
                             (IP (NP)
                                 (I-BAR (I-0 (HAVE "have"))
                                      (VP (IS "been") (NP (DET "the") (N "focus")))))))
                     (PP (P "of") (NP (AP (JJ "linguistic")) (N "study"))))))))
> (PP (P "for")
      (IP (NP (NP (AP "many") (N "years")) (*CMA ",") (CC "and")
              (NP (DET "this") (N "book")))
          (I-BAR (I-0)
                 (VP (V "includes")
                     (NP (NP (N "insights")) (I-BAR (I-0) (VP (V "gained"))))))))
> (PP (P "from") (NP (DET "these") (N "studies")))
> (*PER ".")

Time: 0.437 seconds.
```

19. In forty years , since digital computers were developed , people have programmed them to perform many activities that we think of as requiring some form of intelligence . (2.5 secs, 32 parses)

```
> (PP (P "In") (NP (NUM (CD "forty")) (N "years")))
> (*CMA ",")
> (CP (COMP "since")
      (IP (NP (AP (JJ "digital")) (N "computers"))
          (I-BAR (I-0 (IS "were")) (VP (V "developed") (NP)))))
> (*CMA ",")
> (IP (NP (N "people"))
      (I-BAR (I-0 (HAVE "have")) (VP (V "programmed") (NP (PRO "them")))))
> (I-BAR (I-0 (TO "to") (ADV "of"))
         (VP (V "perform")
             (NP (NP (AP "many") (N "activities"))
                 (CP (COMP "that")
                     (IP (NP (PRO "we")) (I-BAR (I-0) (VP (V "think") (NP))))))))
> (PP (P "as")
      (NP (I-BAR (I-0)
                 (VP (V "requiring") (NP (DET "some") (N "form"))
                     (PP (P "of") (NP (N "intelligence")))))))
> (*PER ".")

Time: 0.325 seconds.
```

```
20. Our study of the mental processes involved in language draws heavily on concepts that
    have been developed in the area called artificial intelligence. (1.48 secs, 20 parses)

> (NP (NP (DET "Our") (N "study"))
      (PP (P "of") (NP (DET "the") (AP (JJ "mental")) (N "processes"))))
> (I-BAR (I-0) (VP (V "involved")))
> (PP (P "in") (NP (N "language") (N "draws")))
> (ADV "heavily")
> (PP (P "on")
      (NP (NP (N "concepts"))
          (CP (COMP "that")
              (IP (NP) (I-BAR (I-0 (HAVE "have") (IS "been"))
                              (VP (V "developed") (NP)))))))
> (PP (P "in")
      (NP (NP (DET "the") (N "area"))
          (I-BAR (I-0) (VP (V "called")
                           (NP (AP (JJ "artificial")) (N "intelligence"))))))
> (*PER ".")

Time: 0.277 seconds.
```

10.2 Examples from the LOB

The following 20 sentences are the the beginning of the LOB. They were parsed without using the LOB pretags,*i.e.*, disambiguation was performed.

```
1. *' stop electing life peers **' .

> (* "*'")
> (I-BAR (I-0) (VP (V "stop")
                   (I-BAR (I-0) (VP (V "electing") (NP (N "life") (N "peers"))))))
> (** "**'")
> (*PER ".")

Time: 0.137 seconds.

2. a move to stop Mr Gaitskell from nominating any more labour life peers is to be made at
   a meeting of labour MPs tomorrow .

> (NP (DET "a") (N "move"))
> (I-BAR (I-0 (TO "to"))
         (VP (V "stop") (NP (PROP (N "Mr") (NAME "Gaitskell")))
            (PP (P "from")
                (NP
                  (I-BAR (I-0 (ADV "any"))
                         (VP (V "nominating")
                             (NP (AP "more") (N "labour") (N "life") (N "peers"))))))))
> (I-BAR (I-0 (NP (N "tomorrow")))
         (VP (IS "is")
             (I-BAR (I-0 (TO "to") (IS "be"))
                    (VP (V "made") (NP)
                        (PP (P "at") (NP (NP (DET "a") (N "meeting"))
                                         (PP (P "of") (NP (N "labour") (N "MPs")))))))))
> (*PER ".")
```

```
Time: 0.219 seconds.

3. Mr Michael Foot has put down a resolution on the subject and he is to be backed by
   Mr Will Griffiths , MP for Manchester Exchange .

> (IP (NP (PROP (N "Mr") (NAME "Michael") (NAME "Foot")))
      (I-BAR (I-0 (HAVE "has") (RP "down"))
             (VP (V "put")
                 (NP (DET "a") (N "resolution"))
                 (PP (P "on") (NP (DET "the") (N "subject"))))))
> (CC "and")
> (IP (NP (PRO "he"))
      (I-BAR (I-0)
             (VP (IS "is")
                 (I-BAR
                   (I-0 (TO "to") (IS "be")
                        (PP (P "by") (NP (PROP (N "Mr") (NAME "Will")
                                                (NAME "Griffiths")))))
                   (VP (V "backed") (NP))))))
> (*CMA ",")
> (NP (N "MP"))
> (PP (P "for") (NP (PROP (NAME "Manchester") (NAME "Exchange"))))
> (*PER ".")

Time: 0.273 seconds.

4. though they may gather some left-wing support , a large majority of labour MPs are
   likely to turn down the Foot-Griffiths resolution .

> (CP (COMP "though")
      (IP (NP (PRO "they"))
          (I-BAR (I-0 (MD "may"))
                 (VP (V "gather")
                     (NP (DET "some") (AP (JJ "left-wing")) (N "support"))))))
> (*CMA ",")
> (IP
     (NP (NP (DET "a") (AP (JJ "large")) (N "majority"))
         (PP (P "of") (NP (N "labour") (N "MPs"))))
     (I-BAR (I-0 (ADV "likely"))
            (VP (IS "are")
                (I-BAR (I-0 (TO "to") (RP "down"))
                       (VP (V "turn")
                           (NP (DET "the") (PROP (NAME "Foot-Griffiths"))
                               (N "resolution")))))))
> (*PER ".")

Time: 0.199 seconds.

5. Mr Foot's line will be that as labour MPs opposed the government bill which brought
   life peers into existence , they should not now put forward nominees .

> (NP (NP (PROP (N "Mr") (NAME "Foot's"))) (NP (N "line")))
> (I-BAR (I-0 (MD "will")) (VP (IS "be")))
> (COMP "that")
> (PP (P "as")
      (NP (NP (N "labour") (N "MPs"))
          (I-BAR (I-0)
                 (VP (V "opposed")
```

```
                    (NP (NP (DET "the") (N "government") (N "bill"))
                        (CP (COMP "which")
                            (IP (NP)
                                (I-BAR (I-0)
                                       (VP (V "brought") (NP (N "life") (N "peers")))))))
                    (PP (P "into") (NP (N "existence")))))))
> (*CMA ",")
> (IP (NP (PRO "they"))
      (I-BAR (I-0 (MD "should") (XNOT "not") (ADV "now") (ADV "forward"))
             (VP (V "put") (NP (N "nominees")))))
> (*PER ".")

Time: 0.393 seconds.
```

6. he believes that the House of Lords should be abolished and that labour should not take any steps which would appear to *' prop up **' an out-dated institution .

```
> (IP (NP (PRO "he"))
      (I-BAR (I-0)
             (VP (V "believes")
                 (CP (COMP "that")
                     (IP (NP (NP (DET "the") (N "House")) (PP (P "of") (NP (N "Lords"))))
                         (I-BAR (I-0 (MD "should") (IS "be"))
                                (VP (V "abolished") (NP))))))))
> (CC "and")
> (IP (NP (DET "that") (N "labour"))
      (I-BAR (I-0 (MD "should") (XNOT "not"))
             (VP (V "take") (NP (DET "any") (N "steps"))
                 (CP (COMP "which")
                     (IP (NP) (I-BAR (I-0 (MD "would")) (VP (V "appear"))))))))
> (P "to")
> (* "*'")
> (I-BAR (I-0 (RP "up")) (VP (V "prop")))
> (** "**'")
> (NP (DET "an") (AP (JJ "out-dated")) (N "institution"))
> (*PER ".")

Time: 0.351 seconds.
```

7. since 1958 , 13 labour life peers and peeresses have been created .

```
> (CP (COMP "since")
      (IP
        (NP (NP (NUM (CD "1958"))) (*CMA ",")
            (NP (NUM (CD "13")) (N "labour") (N "life") (N "peers"))
            (CC "and") (NP (N "peeresses")))
        (I-BAR (I-0 (HAVE "have") (IS "been")) (VP (V "created") (NP)))))
> (*PER ".")

Time: 0.151 seconds.
```

8. most labour sentiment would still favour the abolition of the House of Lords , but while it remains labour has to have an adequate number of members .

```
> (IP (NP (AP "most") (N "labour") (N "sentiment"))
      (I-BAR (I-0 (MD "would") (ADV "still"))
             (VP (V "favour")
                 (NP (NP (DET "the") (N "abolition"))
                     (PP (P "of") (NP (DET "the") (N "House"))))
```

```
                (PP (P "of") (NP (N "Lords"))))))
> (*CMA ",")
> (CC "but")
> (CP (COMP "while") (IP (NP (PRO "it")) (I-BAR (I-0) (VP (V "remains")))))
> (CP
    (IP (NP (N "labour"))
        (I-BAR (I-0)
               (VP (HAVE "has")
                   (I-BAR (I-0 (TO "to"))
                          (VP (HAVE "have")
                              (NP (NP (DET "an") (AP (JJ "adequate")) (N "number"))
                                  (PP (P "of") (NP (N "members")))))))))
> (*PER ".")

Time: 0.320 seconds.

9. Africans drop rivalry to fight Sir Roy .

> (IP (NP (N "Africans")) (I-BAR (I-0) (VP (V "drop") (NP (N "rivalry")))))
> (I-BAR (I-0 (TO "to")) (VP (V "fight") (NP (PROP (N "Sir") (NAME "Roy")))))
> (*PER ".")

Time: 0.142 seconds.

10. the two rival African nationalist parties of Northern Rhodesia have agreed to get
    together to face the challenge from Sir Roy Welensky , the federal Premier .

> (NP (DET "the") (NUM (CD "two")) (N "rival"))
> (IP (NP (NP (AP (JJ "African") (JJ "nationalist")) (N "parties"))
          (PP (P "of") (NP (PROP (NAME "Northern") (NAME "Rhodesia")))))
      (I-BAR (I-0 (HAVE "have")) (VP (V "agreed") (NP))))
> (I-BAR (I-0 (TO "to") (ADV "together"))
         (VP (V "get")
             (I-BAR (I-0 (TO "to"))
                    (VP (V "face") (NP (DET "the") (N "challenge"))
                        (PP (P "from")
                            (NP (NP (PROP (N "Sir") (NAME "Roy") (NAME "Welensky")))
                                (*CMA ",")
                                (NP (DET "the") (AP (JJ "federal")) (N "Premier"))
                                (*PER "."))))))

Time: 0.311 seconds.

11. delegates from Mr Kenneth Kaunda's united national independence party ( 280,000
    members ) and Mr Harry Nkumbula's African national congress ( 400,000 ) will meet
    in London today to discuss a common course of action .

> (NP (N "delegates"))
> (PP (P "from")
      (NP (NP (PROP (N "Mr") (NAME "Kenneth") (NAME "Kaunda's")))
          (NP (AP (JJ "united") (JJ "national")) (N "independence") (N "party"))))
> (*LPR "(")
> (NP (NUM (CD "280,000")) (N "members"))
> (*RPR ")")
> (CC "and")
> (NP (NP (PROP (N "Mr") (NAME "Harry") (NAME "Nkumbula's")))
      (NP (AP (JJ "African") (JJ "national")) (N "congress")))
> (*LPR "(")
> (NP (NUM (CD "400,000")))
```

```
> (*RPR ")")
> (I-BAR (I-0 (MD "will")) (VP (V "meet")))
> (PP (P "in") (NP (PROP (NAME "London"))))
> (NP (N "today"))
> (I-BAR (I-0 (TO "to"))
         (VP (V "discuss") (NP (DET "a") (AP (JJ "common")) (N "course"))
  (PP (P "of") (NP (N "action")))))
> (*PER ".")
```

Time: 0.209 seconds.

12. Sir Roy is violently opposed to Africans getting an elected majority in Northern Rhodesia , but the colonial Secretary , Mr Iain Macleod , is insisting on a policy of change .

```
> (IP (NP (PROP (N "Sir") (NAME "Roy")))
      (I-BAR (I-0 (IS "is") (ADV "violently"))
             (VP (V "opposed") (NP)
                 (PP (P "to")
                     (NP
                       (NP (NP (N "Africans"))
                           (I-BAR (I-0)
                                  (VP (V "getting")
                                      (NP (DET "an") (AP (JJ "elected")) (N "majority"))
                                      (PP (P "in")
                                          (NP (NP (PROP (NAME "Northern")
                                                        (NAME "Rhodesia"))))))))))))
> (IP (NP (NP (DET "the") (AP (JJ "colonial")) (N "Secretary"))
          (*CMA ",")
          (PROP (N "Mr") (NAME "Iain") (NAME "Macleod")) (*CMA ","))
      (I-BAR (I-0 (IS "is") (RP "on")) (VP (V "insisting"))))
> (NP (NP (DET "a") (N "policy")) (PP (P "of") (NP (N "change"))))
> (*PER ".")
```

Time: 0.583 seconds.

13. sir Roy's united federal party is boycotting the London talks on the protectorate's future .

```
> (IP (NP (NP (PROP (N "sir") (NAME "Roy's")))
          (NP (AP (JJ "united") (JJ "federal")) (N "party")))
      (I-BAR (I-0 (IS "is"))
             (VP (V "boycotting") (NP (DET "the") (PROP (NAME "London")) (N "talks")))))
> (PP (P "on") (NP (NP (DET "the") (N "protectorate's")) (NP (N "future"))))
> (*PER ".")
```

Time: 0.165 seconds.

14. said Mr Nkumbula last night :

```
> (I-BAR (I-0) (VP (V "said")
                   (NP (PROP (N "Mr") (NAME "Nkumbula")) (N "last") (N "night"))))
> (*COL ":")
```

Time: 0.059 seconds.

15. *' we want to discuss what to do if the British government gives in to Sir Roy and the talks fall through .

```
> (* "*'")
> (IP (NP (PRO "we")) (I-BAR (I-0) (VP (V "want"))))
> (I-BAR (I-0 (TO "to")) (VP (V "discuss")))
> (IP (NP (PRO "what")) (I-BAR (I-0 (TO "to")) (VP (DID "do") (NP))))
> (CP (COMP "if")
      (IP (NP (DET "the") (AP (JJ "British")) (N "government"))
          (I-BAR (I-0 (RP "in"))
                 (VP (V "gives")
                     (PP (P "to")
                         (NP (NP (PROP (N "Sir") (NAME "Roy")))
                             (CC "and") (NP (DET "the") (N "talks"))))))))
> (I-BAR (I-0 (RP "through")) (VP (V "fall")))
> (*PER ".")
```

Time: 0.197 seconds.

16. there are bound to be demonstrations . **'

```
> (IP (NP (PRO "there")) (I-BAR (I-0 (IS "are")) (VP (V "bound") (NP))))
> (I-BAR (I-0 (TO "to")) (VP (IS "be") (NP (N "demonstrations"))))
> (*PER ".")
> (** "**'")
```

Time: 0.066 seconds.

17. yesterday Sir Roy's chief aide , Mr Julius Greenfield , telephoned his chief a report on his talks with Mr Macmillan at Chequers .

```
> (NP (N "yesterday"))
> (IP (NP (NP (PROP (N "Sir") (NAME "Roy's")))
          (NP (NP (N "chief") (N "aide")) (*CMA ",")
              (PROP (N "Mr") (NAME "Julius") (NAME "Greenfield")) (*CMA ",")))
      (I-BAR (I-0) (VP (V "telephoned") (NP (DET "his") (N "chief"))
                       (NP (DET "a") (N "report")))))
> (PP (P "on") (NP (DET "his") (N "talks")))
> (PP (P "with") (NP (PROP (N "Mr") (NAME "Macmillan"))))
> (PP (P "at") (NP (PROP (NAME "Chequers"))))
> (*PER ".")
```

Time: 0.477 seconds.

18. Mr Macleod went on with the conference at Lancaster House despite the crisis which had blown up .

```
> (IP (NP (PROP (N "Mr") (NAME "Macleod")))
      (I-BAR (I-0 (RP "on"))
             (VP (V "went") (PP (P "with") (NP (DET "the") (N "conference"))))))
> (PP (P "at") (NP (PROP (NAME "Lancaster") (N "House"))))
> (PP (P "despite")
      (NP (NP (DET "the") (N "crisis"))
          (CP (COMP "which")
              (IP (NP) (I-BAR (I-0 (HAVE "had") (RP "up")) (VP (V "blown")))))))
> (*PER ".")
```

Time: 0.162 seconds.

19. he has now revealed his full plans to the Africans and liberals attending .

```
>  (IP (NP (PRO "he"))
```

```
      (I-BAR (I-0 (HAVE "has") (ADV "now"))
             (VP (V "revealed") (NP (DET "his") (AP (JJ "full")) (N "plans")))))
>  (PP (P "to")
      (NP (NP (NP (DET "the") (N "Africans")) (CC "and") (NP (N "liberals")))
          (I-BAR (I-0) (VP (V "attending")))))
==>  (*PER ".")

Time: 0.101 seconds.

20. these plans do not give the Africans the overall majority they are seeking .

>  (IP (NP (DET "these") (N "plans"))
      (I-BAR (I-0 (DID "do") (XNOT "not"))
             (VP (V "give")
                 (NP (DET "the") (N "Africans"))
                 (NP (DET "the") (N "overall") (N "majority")))))
>  (IP (NP (PRO "they")) (I-BAR (I-0 (IS "are")) (VP (V "seeking") (NP))))
>  (*PER ".")

Time: 0.117 seconds.
```

Appendix F: Sample Multivalued Disambiguations

The following 20 disambiguations (Method D, $F = .01$ without oracle) are taken randomly from the LOB. In each case, first the sentence is presented (the words and their dictionary entries), and then the disambiguation is presented as a list of `(pretag: disambiguation-categories)`. The order of `disambiguation-categories` reflects the disambiguator's evaluation of relative likelihood, first being most likely.

265 words are disambiguated (1.19 categories per word returned) with two errors, about twice what would be expected, though one (sentence 7) is a mistagging and the other (sentence 12) is a close call. In 10 cases multiple categories are returned and the correct category is not the first in the listing. These are instances where multivaluedness has prevented errors.

```
1. Sent: (*: *) (I: NC PP1A ZZ) (DO: DO NN) (N: XNOT NC ZZ) (THINK: VB)
         (SO: NP PN RB QL CS TO JJB) (,: *CMA) (**: **) (MORAG: NP) (REPLIED:
         VBD VBN) (.: *PER)
   Disambig: (*: *) (PP1A: PP1A) (DO: DO) (XNOT: XNOT) (VB: VB) (PN: RB CS
             PN QL) (*CMA: *CMA) (**: **) (NP: NP) (VBD: VBD) (*PER: *PER)

2. Sent: (*: *) (IS: BEZ) (IT: PP3 NC) (A: ZZ AT) (HOSTEL: NN) (?: *QUE) (**: **)
   Disambig: (*: *) (BEZ: BEZ) (PP3: PP3) (AT: AT) (NN: NN) (*QUE: *QUE) (**: **)

3. Sent: (HE: PP3A) (FOUND: VBN VBD VB JJ) (IT: PP3 NC) (EASIER: JJR RBR)
         (TO: TO IN NC RP NN) (EVADE: VB) (HER: PP$ PP3O) (ADVANCES: NNS VBZ)
         (NOW: RN CS) (THAT: CS DT WPR QL) (HE: PP3A) (KNEW: VBD) (THAT: CS DT
         WPR QL) (SHE: PP3A) (WAS: BEDZ) (BEING: BEG NN) (TRANSFERRED: VBN VBD)
         (.: *PER)
   Disambig: (PP3A: PP3A) (VBD: VBD) (PP3: PP3) (JJR: JJR RBR) (TO: TO)
             (VB: VB) (PP$: PP$ PP3O) (NNS: NNS) (CS: RN CS) (CS: CS) (PP3A: PP3A)
             (VBD: VBD) (CS: CS) (PP3A: PP3A) (BEDZ: BEDZ) (BEG: BEG) (VBN: VBN)
             (*PER: *PER)

4. Sent: (HE: PP3A) (LAUGHED: VBD VBN) (.: *PER)
```

Disambig: (PP3A: PP3A) (VBD: VBD) (*PER: *PER)

5. Sent: (FIELD: NP VB NN) (LEFT: VBN VBD NN JJ RB) (ABOUT: IN RP RB) (TEN: CD) (MINUTES: NNS) (AFTER: IN CS JJB RI) (THE: ATI) (TWO: CD) (DETECTIVES: NNS) (.: *PER)
   Disambig: (NP: NN NP) (VBD: VBN NN VBD JJ) (RB: IN RB) (CD: CD) (NNS: NNS) (IN: IN) (ATI: ATI) (CD: CD) (NNS: NNS) (*PER: *PER)

6. Sent: (A: ZZ AT) (SMALL: NP JJ NN) (BOY: NP NN UH) (IN: IN RP RB TO CS NN NC) (A: ZZ AT) (BLUE: NP JJ NN) (BLAZER: NN) (WAS: BEDZ) (WALKING: VBG NN) (ALONG: IN RP) (THE: ATI) (PAVEMENT: NN) (.: *PER)
   Disambig: (AT: AT) (JJ: JJ) (NN: NN) (IN: IN) (AT: AT) (JJ: JJ NN) (NN: NN) (BEDZ: BEDZ) (VBG: VBG) (IN: IN RP) (ATI: ATI) (NN: NN) (*PER: *PER)

7. Sent: (A: ZZ AT) (VOICE: NP NN VB) (,: *CMA) (JUST: NP RB JJ) (A: ZZ AT) (LITTLE: NP AP JJ RB NC) (LOUDER: JJR RBR) (THAN: CS IN) (PERHAPS: RB) (WAS: BEDZ) (INTENDED: VBN VBD JJ) (,: *CMA) (FLOATED: VBD) (TOWARDS: IN) (HER: PP$ PP3O) (.: *PER)
   Disambig: (AT: AT) (NN: NN) (*CMA: *CMA) (RB: RB) (RB: AT) (RB: RB JJ AP) (JJR: JJR RBR) (CS: IN CS) (RB: RB) (BEDZ: BEDZ) (VBN: VBN JJ) (*CMA: *CMA) (VBD: VBD) (IN: IN) (PP3O: PP3O PP$) (*PER: *PER)

8. Sent: (I: NC PP1A ZZ) (SHIFTED: VBD VBN) (HIS: PP$ PP$$) (HEAD: NP NPL NN JJB VB) (TO: TO IN NC RP NN) (MY: PP$) (KNEE: NN) (.: *PER)
   Disambig: (PP1A: PP1A) (VBD: VBD) (PP$: PP$) (NN: NN) (IN: IN) (PP$: PP$) (NN: NN) (*PER: *PER)

9. Sent: (THIS: DT QL) (IS: BEZ) (THE: ATI) (WORST: JJT RBT) (PATCH: NN VB) (.: *PER)
   Disambig: (DT: DT) (BEZ: BEZ) (ATI: ATI) (JJT: JJT) (NN: NN) (*PER: *PER)

10. Sent: (SHE: PP3A) (MADE: VBN VBD) (HIM: PP3O) (HANG: NP VB NN) (HIS: PP$ PP$$) (PAINTINGS: NNS) (ON: NP IN RP) (THE: ATI) (WALLS: NP NNS) (AND: CC) (COULD: MD) (SCARCELY: RB) (BELIEVE: VB) (THE: ATI) (BRILLIANT: JJ) (TRANSFORMATION: NN) (.: *PER)
    Disambig: (PP3A: PP3A) (VBD: VBD) (PP3O: PP3O) (VB: VB) (PP$: PP$) (NNS: NNS) (IN: IN) (ATI: ATI) (NNS: NNS) (CC: CC) (MD: MD) (RB: RB) (VB: VB) (ATI: ATI) (JJ: JJ) (NN: NN) (*PER: *PER)

11. Sent: (*: *) (DO: DO NN) (WHAT: WDT) (HE: PP3A) (SAYS: VBZ) (,: *CMA) (HE: PP3A) ('S: BEZ PP1OS HVZ) (GETTING: VBG NN) (SELFISH: JJ) (ENOUGH: QLP DTI RB) (.: *PER) (**: **)
    Disambig: (*: *) (DO: DO) (WDT: WDT) (PP3A: PP3A) (VBZ: VBZ) (*CMA: *CMA) (PP3A: PP3A) (BEZ: BEZ) (VBG: VBG) (JJ: JJ) (QLP: QLP RB) (*PER: *PER) (**: **)

12. Sent: (LINDA: NP) (LOOKED: VBD VBN) (AT: IN RB NC) (MRS: NPT) (BATLEY: NP) (WHO: CS WP) (WAS: BEDZ) (NOW: RN CS) (THRUSTING: VBG JJ) (THE: ATI) (CHINA: NP NN) (NOISILY: RB) (INTO: IN) (THE: ATI) (CUPBOARD: NN) (.: *PER)
    Disambig: (NP: NP) (VBD: VBD) (IN: IN) (NPT: NPT) (NP: NP) (WPR: CS WP) (BEDZ: BEDZ) (RN: RN) (VBG: VBG) (ATI: ATI) (NN: NN NP) (RB: RB) (IN: IN) (ATI: ATI) (NN: NN) (*PER: *PER)

13. Sent: (*: *) (WERE: BED) (YOU: PP2) (INSIDE: IN RP JJB NN) (THE: ATI) (HOUSE: NPL NN VB) (?: *QUE) (**: **)
    Disambig: (*: *) (BED: BED) (PP2: PP2) (IN: IN RP) (ATI: ATI) (NN: NN) (*QUE: *QUE) (**: **)

```
14. Sent: (THE: ATI) (TABLE: NN) (WAS: BEDZ) (SET: VBN VB NN VBD JJ) (NOW:
          RN CS) (WITH: IN RI NC) (A: ZZ AT) (WHITE: NP JJ NN) (CLOTH: NN) (AND:
          CC) (WAS: BEDZ) (LADEN: VBN) (WITH: IN RI NC) (FOOD: NN) (AND: CC)
          (SHE: PP3A) (SAID: VBD VBN JJ) (APOLOGETICALLY: RB) (:: *COL IN)
    Disambig: (ATI: ATI) (NN: NN) (BEDZ: BEDZ) (VBN: VBN NN) (RN: RN CS)
              (IN: IN) (AT: AT) (JJ: JJ NN) (NN: NN) (CC: CC) (BEDZ: BEDZ) (VBN:
              VBN) (IN: IN) (NN: NN) (CC: CC) (PP3A: PP3A) (VBD: VBD) (RB: RB)
              (*COL: *COL IN)

15. Sent: (JUDY: NP) (KNEW: VBD) (THAT: CS DT WPR QL) (IF: CS) (SHE: PP3A)
          (MADE: VBN VBD) (THE: ATI) (SLIGHTEST: JJT) (MOVE: NN VB) (TOWARDS:
          IN) (CHARLES: NP NC) (SHE: PP3A) (WOULD: MD) (BE: BE) (IN: IN RP RB TO
          CS NN NC) (HIS: PP$ PP$$) (ARMS: NNS)
    Disambig: (NP: NP) (VBD: VBD) (CS: CS) (CS: CS) (PP3A: PP3A) (VBD:
              VBD) (ATI: ATI) (JJT: JJT) (NN: NN VB) (IN: IN) (NP: NP) (PP3A: PP3A)
              (MD: MD) (BE: BE) (IN: IN) (PP$: PP$) (NNS: NNS)

16. Sent: (*: *) (YOU: PP2) (SEE: VB NC) (,: *CMA) (IT: PP3 NC) ('S: BEZ
          PP1OS HVZ) (ALL: ABN RB JJ) (VERY: QL AP) (WELL: JJ RB UH NN) (FOR: IN
          CS RB NC) (YOU: PP2) (;: *SEM) (YOU: PP2) (CAN: MD NN) (GO: VB NN NC)
          (FOR: IN CS RB NC) (WALKS: NNS VBZ) (BY: IN RB RP JJB NC CD1)
          (YOURSELF: PPL) (,: *CMA) (EXPLORE: VB) (ANYWHERE: RB) (YOU: PP2)
          (LIKE: IN VB JJB CS NN RI) (.: *PER)
    Disambig: (*: *) (PP2: PP2) (VB: VB) (*CMA: *CMA) (PP3: PP3) (BEZ:
              BEZ) (RB: RB ABN) (QL: QL) (JJ: RB JJ) (IN: IN CS) (PP2: PP2) (*SEM:
              *SEM) (PP2: PP2) (MD: MD) (VB: VB) (IN: IN) (NNS: NNS) (IN: IN) (PPL:
              PPL) (*CMA: *CMA) (VB: VB) (RB: RB) (PP2: PP2) (VB: VB IN NN) (*PER:
              *PER)

17. Sent: (*: *) (IS: BEZ) (THERE: EX RN) (ANYTHING: PN RB) (IN: IN RP RB
          TO CS NN NC) (YOUR: PP$) (BOOK: NN VB) (THAT: CS DT WPR QL) (PEOPLE:
          NNS NN) (HAVE: HV) (N: XNOT NC ZZ) (SAID: VBD VBN JJ) (ABOUT: IN RP
          RB) (ME: NP PP1O NC PP$) (AND: CC) (BELIEVED: VBD VBN) (ABOUT: IN RP
          RB) (ME: NP PP1O NC PP$) (?: *QUE) (**: **)
    Disambig: (*: *) (BEZ: BEZ) (EX: EX RN) (PN: PN RB) (IN: IN) (PP$:
              PP$) (NN: NN) (WPR: CS WPR DT) (NNS: NNS) (HV: HV) (XNOT: XNOT ZZ)
              (VBN: VBN VBD) (IN: IN) (PP1O: PP1O NP) (CC: CC) (VBN: VBD VBN) (IN:
              IN) (PP1O: PP1O) (*QUE: *QUE) (**: **)

18. Sent: (*: *) (MY: PP$) (ONLY: RB AP NC CC) (CRITICISM: NN) (.: *PER) (**: **)
    Disambig: (*: *) (PP$: PP$) (AP: AP) (NN: NN) (*PER: *PER) (**: **)

19. Sent: (SHE: PP3A) (HAD: HVD HVN) (AVOIDED: VBN VBD) (,: *CMA) (IT: PP3
          NC) (IS: BEZ) (TRUE: JJ RB) (,: *CMA) (ANY: DTI RB) (OPEN: JJ NN VB)
          (SCANDAL: NN) (.: *PER)
    Disambig: (PP3A: PP3A) (HVD: HVD) (VBN: VBN) (*CMA: *CMA) (PP3: PP3)
              (BEZ: BEZ) (JJ: JJ RB) (*CMA: *CMA) (DTI: DTI RB) (JJ: JJ NN) (NN: NN)
              (*PER: *PER)

20. Sent: (IT: PP3 NC) (IS: BEZ) (THE: ATI) (KIND: NP NN JJ RB) (OF: IN RB
          HV) (QUESTION: NN VB) (THAT: CS DT WPR QL) (USUALLY: RB) (A: ZZ AT)
          (NOVELIST: NN) (RESENTS: VBZ) (;: *SEM) (RESENTS: VBZ) (BECAUSE: CS
          IN) (IT: PP3 NC) (IS: BEZ) (IMPOSSIBLE: JJ) (TO: TO IN NC RP NN)
          (REPLY: NN VB) (HONESTLY: RB) (.: *PER)
    Disambig: (PP3: PP3) (BEZ: BEZ) (ATI: ATI) (NN: NN) (IN: IN) (NN: NN)
              (WPR: CS WPR DT) (RB: RB) (AT: AT) (NN: NN) (VBZ: VBZ) (*SEM: *SEM)
              (VBZ: VBZ) (CS: CS IN) (PP3: PP3) (BEZ: BEZ) (JJ: JJ) (TO: TO IN) (VB:
              VB NN) (RB: RB) (*PER: *PER)
```