Chapter 1

BALANCING ROBUSTNESS AND EFFICIENCY IN UNIFICATION-AUGMENTED CONTEXT-FREE PARSERS FOR LARGE PRACTICAL APPLICATIONS

Carolyn Penstein Rosé, University of Pittsburgh rosecp@pitt.edu
and Alon Lavie, Carnegie Mellon University alavie@cs.cmu.edu

Abstract Large practical NLP applications require robust analysis components that can effectively handle input that is disfluent or extra-grammatical. The effectiveness and efficiency of any robust parser are a direct function of three main factors: (1) Flexibility: what types of disfluencies and deviations from the grammar can the parser handle?; (2) Search: How does the parser search the space of possible interpretations, and what techniques are applied to prune the search space?; and (3) Parse Selection and Disambiguation: What methods and resources are used to evaluate and rank potential parses and sub-parses, and how does the parser cope with the extreme levels of ambiguity introduced by its flexibility parameters? In this chapter we describe our investigations on how to balance flexibility and efficiency in the context of two different robust parsers - a GLR parser and a left corner Chart parser - both based on a unification-augmented context-free grammar formalism. We demonstrate how the combination of a beam search together with ambiguity packing and statistical disambiguation provide a flexible framework for achieving a good balance between robustness and efficiency in such parsers. Our investigations are based on experimental results and comparative performance evaluations of both parsers using a grammar for the spoken language ESST (English Spontaneous Scheduling Task) domain.

1. INTRODUCTION

Large practical NLP applications require robust analysis components that can effectively handle input that is disfluent or extra-grammatical. Regardless of the specific formalism and grammar they employ, NLP sys-

tems that process free unrestricted input must be prepared to frequently encounter input that deviates from the coverage of their pre-developed grammar (and other knowledge sources). Traditional parsing algorithms that are designed to parse only completely grammatical input sentences (and fail on even the slightest deviation from the grammar) are therefore unsuitable for such applications. This problem is even more evident in systems that process spoken language. Spontaneous spoken language, whether transcribed by hand or by a speech recognizer, is often highly disfluent, with the meaningful portions of the utterance surrounded by a variety of phenomena that disrupt the grammaticality of the overall input stream. Common types of extra-grammaticality include repetitions, false starts, filled pauses, and idiosyncratic constructions found within a speaker's performance grammar but not covered by a particular system's parsing grammar. Processing such language requires an analysis component that is robust to the various types of disfluencies and can extract the most complete interpretation possible from a given input.

Researchers have approached the robust parsing problem from a variety of different directions, including symbolic (Rosé, 1997; Lavie, 1995; Ait-Mokhtar and Chanod, 1997; Neumann et al., 1997; McDonald, 1993b; McDonald, 1990; Lehman, 1989; Hipp, 1992), statistical (Bod, 1998; Miller et al., 1996; Pietra et al., 1997; Rosé and Waibel, 1997; Goodman, 1996; Magerman and Marcus, 1990; Sanker and Gorin, 1993), and connectionist (Henderson and Lane, 1998; Buo, 1996; Jain, 1991; Jain and Waibel, 1990). While statistical and connectionist approaches are inherently robust, and can often be trained automatically from labeled corpora, symbolic parsers are most capable of performing deep and detailed analysis based on linguistic principles. Thus, symbolic parsers are still highly attractive for tasks that require these detailed linguistic analyses, such as Machine Translation (MT) or Intelligent Tutoring.

To avoid failure when faced with extra-grammatical input, many robust approaches to symbolic parsing relax syntax and grammar constraints. In some cases, semantic information is used to compensate for the lack of grammaticality in order to partially or completely drive the analysis process. The earliest approaches to robust parsing involved hand-coded grammar specific heuristics for selecting a subset of analyses to extend when each input word was processed (Hobbs et al., 1991; McDonald, 1993b; McDonald, 1993a; McDonald, 1992). With these approaches, the work of making the parser robust and efficient must be redone by hand for every new grammar developed. The use of hand-coded heuristics also constrains these approaches to handle only those cases that are consistent with the simplifying assumptions represented by those heuristics.

Some recent approaches to robust parsing focus instead on shallow or partial parsing techniques (Van Noord, 1997; Abney, 1996; Ait-Mokhtar and Chanod, 1997; Worm, 1998). Rather than attempting to construct a parse covering an entire ungrammatical sentence, these approaches attempt to construct analyses for maximal contiguous portions of the input. Thus, they address the computational expense problem by limiting the flexibility of their parsing algorithms. The result is a practical form of robust parsing not involving any grammar specific heuristics, but more shallow in its resulting analysis than could possibly be achieved using other types of flexibility. For certain applications, this level of analysis may certainly be sufficient. However, if a more complete analysis is required, additional post-parsing processing is necessary, where robustness must also be applied.

Perhaps the most complete approach to the problem of robust symbolic parsing is *Minimum Distance Parsing* (MDP) (Lehman, 1989; Hipp, 1992). When faced with input that is extra-grammatical, the goal of an MDP parser is to find the analysis of a corresponding grammatical input that is closest in meaning to the given input. However, for practical reasons, the parser instead searches for an analysis of an input that deviates as little as possible from the original input. This is often accomplished using a "distance" metric that is based on a variety of possible parser flexibilities, such as deletions, insertions or substitutions of words in the input, with the "closest" distance being determined by using adjustable penalties. MDP was originally developed in the context of parsing programming languages, where the types of ungrammaticality were quite different than those often found in natural language. While full MDP parsers (Hipp, 1992) (Lehman, 1989) have been shown to be effective in small natural language applications with small grammars, the approach does not scale up well to large applications. As demonstrated in our previous work (Rosé, 1997), with large coverage grammars, the search space that a full MDP parser must explore expands to magnitudes that render the approach computationally infeasible. Attention has thus shifted to more limited approaches, that can consider a more restricted set of parser flexibilities, but can maintain parsing efficiency. Our research work, described in this paper, is an investigation into the tradeoffs involved in balancing robustness and efficiency in such robust symbolic parsers.

The effectiveness and efficiency of robust symbolic parsers are a direct function of three main factors: (1) Flexibility: what types of disfluencies and deviations from the grammar can the parser handle?; (2) Search: How does the parser search the space of possible interpretations, and what techniques are applied to prune the search space?; and (3) Parse

Selection and Disambiguation: What methods and resources are used to evaluate and rank potential parses and sub-parses, and how does the parser cope with the extreme levels of ambiguity introduced by its flexibility parameters? In this paper we describe our investigations on how to balance flexibility and efficiency in the context of two different robust parsers: GLR* - a robust GLR parser, and LCFLEX[1] - a left corner Chart parser. Both parsers are based on a unification-augmented context-free grammar formalism. We demonstrate how the combination of a beam search together with ambiguity packing and statistical disambiguation provide a flexible framework for achieving a good balance between robustness and efficiency in such parsers. Our investigations are based on experimental results and comparative performance evaluations of both parsers using a grammar for the spoken language ESST (English Spontaneous Scheduling Task) domain.

We begin with a brief description of the underlying grammar formalism used by both parsers, and the fundamental parsing principles of each. We then describe the flexibility features of our robust parsers. These are accompanied by detailed investigations into the performance effects of each flexibility feature. In our comparisons between our GLR parser and our LC parser we demonstrate that even with the addition of the types of flexibility we explore, our LC parser maintains its order of magnitude computational advantage over GLR in terms of speed. We also demonstrate that a great deal of robustness can be achieved while keeping the parser's efficiency at a practical level. While ambiguity is an orthogonal issue to robustness, it is an essential consideration for maintaining a practical run-time performance in robust parsers because adding flexibility necessarily increases the amount of ambiguity generated. Thus, section 9. discusses our approach to managing ambiguity and disambiguation in our robust parsers. Finally, we present our conclusions and planned future research.

All of the evaluations reported in this paper were conducted on the English Spontaneous Scheduling Domain (ESST) Grammar developed for the JANUS speech translation project (Woszcyna et al., 1993; Woszcyna et al., 1994). The grammar is primarily semantic and consists of approximately 1000 rules with an accompanying lexicon with approximately 3000 lexical entries.

2. THE GRAMMAR FORMALISM

[1]LCFLEX is freely available for research purposes. Interested persons should contact the first author at `rosecp@pitt.edu`.

$$
\begin{array}{l}
(< DECL > < -- > (< NP > < VP >) \\
\quad (((\text{x2 agr}) = (\text{x1 agr})) \\
\quad\ ((\text{x0 subject}) = \text{x1}) \\
\quad\ ((\text{x2 form}) = \text{*finite}) \\
\quad\ (\text{x0} = \text{x2})))
\end{array}
$$

Figure 1.1 **An English Grammar Rule of the GLR Parser/Compiler**

Much attention has been given in the last decade to unification based formalisms such as LFG (Kaplan and Bresnan, 1982), GPSG (Gazdar et al., 1985) and HPSG (Pollard and Sag, 1987), as declarative theories for describing the structure of natural language. A fundamental primitive of these formalisms is the unification of feature based representations. Several recent practical systems for structural analysis of natural language have been designed to support unification based grammar specifications within a general CFG framework, that allows the system to incorporate known CFG parsing algorithms as the core of the system's implementation. Two examples of such systems are the Core Language Engine (CLE) (Alshawi (ed.), 1992) and the Alvey Natural Language Tools (ANLT) (Carroll, 1993).

The underlying grammar formalism used by both GLR* and LCFlex is a unification-based grammar formalism that was originally developed for the Generalized LR Parser/Compiler (Tomita, 1990) (Tomita, 1987) at the Center for Machine Translation at Carnegie Mellon University. The formalism supports grammatical specification in an LFG framework, that consists of context-free grammar rules augmented with feature bundles that are associated with the non-terminals of the rules. For each context-free rule, the grammar formalism allows the specification of the feature structure associated with a left-hand side non-terminal of the rule as a function of the feature structures that are associated with the non-terminals on the right-hand side of the rule. At run-time, the parser computes the actual feature values of a particular left-hand side non-terminal as a by-product of the operation that creates the left-hand side non-terminal from the identified constituents on the right-hand side of the rule. Feature structure computation is, for the most part, specified and implemented via unification operations. This allows the grammar to constrain the applicability of context-free rules. The LHS constituent of a context-free rule is created only if the associated feature structure unification associated with the rule constituents is successful as well.

An example of an augmented grammar rule is shown in Figure 1.1. The variable $x0$ in the feature specification part of the rule is bound

to the left-hand side non-terminal `DECL` of the context-free part of the rule. The variables $x1$, $x2$, etc. are respectively bound to the right-hand side constituents of the rule. The feature structures that are computed at parse-time are attached to their corresponding parse node structures. Parsing with such an augmented grammar produces both c-structure and f-structure - a set of one or more parse trees, along with the feature structure associated with each of these trees.

3. THE GLR* PARSER

The GLR* parser is a robust extension of the Generalized LR (GLR) parser developed by Tomita (Tomita, 1986). Tomita's GLR parsing algorithm evolved out of the LR parsing techniques that were originally developed for parsing programming languages in the late 1960s and early 1970s (Aho and Johnson, 1974), and also follows ideas developed by Lang (Lang, 1974). LR parsers parse the input bottom-up, scanning it from left to right, and producing a rightmost derivation. LR parsers are driven by a table of parsing actions that is pre-compiled from the grammar. Tomita's GLR parsing algorithm (Tomita, 1986) extended the original LR parsing algorithm to the case of non-LR languages, where the parsing tables contain entries with multiple parsing actions. The algorithm deterministically simulates the non-determinism introduced by the conflicting actions in the parsing table by efficiently pursuing in a pseudo-parallel fashion all possible actions. The primary tool for performing this simulation efficiently is the *Graph Structured Stack* (GSS). The GSS is a data-structure that efficiently represents the multiple parsing stacks that correspond to different sequences of parsing actions. Two additional techniques, *local ambiguity packing* and *shared parse forests*, are used in order to efficiently represent the various parse trees of ambiguous sentences when they are parsed. Local ambiguity packing collects multiple sub-analyses of the same category type, all of which derive the same substring, into a single structure. Further parsing actions may then refer to the single structure that represents the entire collection of sub-parses, rather than to each of the packed sub-parses separately. Shared packed forests allow common sub-parses of different analyses to be shared via pointers.

The primary type of flexibility used in GLR* is skipping over words and segments of the input string. Two types of skipping are handled by the parser. The parser can skip over an arbitrary prefix or suffix of the input by allowing complete analyses that do not span the entire input string. We refer to this feature of the parser as *inter-analysis word skipping*. Furthermore, the parser can also skip over sequences of words

internal to an analysis. We refer to this feature as *intra-analysis word skipping*. While words skipped in both cases can be treated similarly, this distinction allows testing the parser in a mode that permits one type of skipping yet not the other. Our previous investigations demonstrate that inter-analysis skipping is much more critical than intra-analysis skipping (Rosé and Lavie, 1997), and some well known parsers have relied entirely upon this form of skipping (Mayfield et al., 1995b; Mayfield et al., 1995c; Mayfield et al., 1995a; Ward, 1989). Experiments comparing these two forms of skipping are described later in the paper.

GLR* utilizes a free skipping approach constrained by a beam search technique free of any grammar specific heuristics. The beam search keeps the parser's performance within reasonable bounds. Thus, the work for making the parser robust and efficient once completed never needs to be done again. Instead of embodying any simplifying assumptions about which analyses to extend upon processing each input word, the size of the beam bounds the growth of the number of analyses pursued in parallel. Thus, any analysis licensed by the grammar by skipping any portion of the input is, in principle, obtainable via this approach, since the beam size can always be adjusted to a sufficiently large setting. Furthermore, the approach requires no backtracking. However, the GLR parsing architecture itself has been demonstrated to be an order of magnitude less efficient than chart-based parsing architectures (Van Noord, 1997). Thus, GLR*'s run-time performance suffers because of inefficiencies in the framework in which it was developed.

4. THE LCFLEX PARSER

LCFlex is a flexible left-corner parser designed for efficient, robust interpretation. Its approach to robustness was originally inspired by our earlier GLR* parser. LCFlex extended the robustness capabilities of GLR* and supports various types of flexibility including word skipping, non-terminal inserting, and flexible unification. While the additional "flexibility features" offered by LCFlex could have just as well been implemented within GLR*, our previous experience with GLR* demonstrates that these additional degrees of flexibility render GLR* intractable (Rosé and Lavie, 1997). One attractive feature of LCFlex is that its "flexibility features" can easily be controlled by setting global parameters. Thus, it is possible to tailor the parser's behavior to the specific needs of individual applications where it is used. The more flexibility allowed within the parser, the greater the robustness, but also the more ambiguity and computational expense, both in terms of time and space. Ideally, LCFlex should be set up in order to achieve the most

desirable robustness versus efficiency trade-off. Because this trade-off is application/grammar/genre dependent, determining the most appropriate setting requires some experimentation in the current version of LCFlex. One of our current research objectives is to semi-automate this process. LCFlex's basic architecture is based on the left-corner parsing algorithm described in (Rosenkrantz and Lewis, 1970; Carroll, 1993). It uses the reflexive and transitive closure of the left-corner relation (Van Noord, 1997) both for top-down filtering and bottom-up prediction in order to limit ambiguity and thus enhance efficiency.

LCFlex utilizes a beam skipping technique similar to the one previously developed within the GLR* parser. Like GLR*, LCFlex searches for the maximal parsable subsequence of a sentence. Thus, it is a grammar-independent approach to robust parsing of full sentences rather than a partial or shallow parsing technique (Abney, 1996; Ait-Mokhtar and Chanod, 1997; Worm, 1998). LCFlex is implemented within the left-corner parsing architecture, which is an order of magnitude faster than the GLR architecture. Even with its added flexibility, it maintains the computational advantage of chart-based parsers over LR based parsers. Additionally, it incorporates the same efficiency measures that were a part of the GLR* parser. It uses a standard ambiguity packing algorithm augmented with a pruning step similar to that introduced in (Lavie, 1995). This pruning step eliminates packed analyses that cover a strictly subsumed subset of the portion of the sentence covered by another packed analysis. For ambiguity resolution, LCFlex uses an adapted version of the statistical disambiguation approach discussed in (Lavie, 1995; Carroll and Briscoe, 1993).

In addition to the flexibility that was an integral part of the GLR* parser, LCFlex embodies several other types of flexibility. In addition to skipping, it also has the ability to consider insertion of missing words and categories, which can also be thought of as partial rule matching. Whereas general insertion has been demonstrated to be impractical in realistic sized applications(Rosé, 1997; Rosé and Lavie, 1997), used sparingly it can be useful in applications such as language tutors and grammar checkers that must not only arrive at an analysis for an extra-grammatical input sentence, but also determine the cause of the extra-grammatical (Schneider and McCoy, 1998). LCFlex can be set to allow specific combinations of insertions. It also has the ability to perform flexible unification. Where complete flexible unification is too inefficient, setting aside a small set of features as "soft-features" can be useful particularly in language learning environments where certain syntactic features tend to be acquired later than others. LCFlex's final flexibility feature is its two-level parsing mode that allows it to construct

a partial analysis at a constrained level of flexibility and then combine the fragments of the partial parse by running the parser again at a far less constrained level of flexibility.

5. WORD SKIPPING

Because spoken language in particular is littered with repetitions, false starts, filled pauses, and other noise surrounding islands of parsable language, we have found word skipping to be the most essential form of robustness. Thus, word skipping is featured prominently both in GLR* and in our more recent LCFlex parser. In this section we describe how our word skipping approach constrained by a beam search technique is implemented in these two alternative parsing architectures, thus demonstrating the architecture independence of the approach. Our end-to-end translation evaluation demonstrates that our beam skipping approach effectively maximizes the parser's robustness while keeping the parser's run time performance practical. Furthermore, we demonstrate that LCFlex's implementation of beam skipping maintains the computational advantage of LC parsers over GLR parsers.

5.1 WORD SKIPPING IN GLR*

GLR* accommodates intra-analysis skipping by manipulating the underlying stack structure of the parser. Whereas the standard GLR parser attaches (shifts) each processed input word to the state at the top of the stack structure, GLR* allows processing the new word also from states that are internal to the stack structure. Shifting an input symbol from an internal state is equivalent to skipping the words of the input that were encountered after the parser reached that state and prior to the current word that is being shifted. For a detailed description of how this process is performed, the reader is referred to (Lavie, 1995).

Allowing the parser to consider all possible internal skipping combinations is computationally infeasible due to the explosion in ambiguity. Thus, GLR* uses a beam search technique to limit the set of words that can be skipped throughout the parsing process. Since the purpose of GLR* is to find maximal (or close to maximal) parsable subsets of the input, applying an adequate beam should allow the flexibility necessary to recover the desired parse. The beam search restricts the parser to pursuing a "beam" of a fixed size k of parsing actions, which are selected according to a criterion that locally minimizes the number of words skipped. There exists a direct tradeoff between the amount of search (and word skipping) that the parser is allowed to pursue and the time and space performance of the parser itself. For a given grammar,

one must determine the smallest possible setting of the beam parameter that allows the parser to find the desired parses in an overwhelming majority of cases.

5.2 WORD SKIPPING IN LCFLEX

Like GLR*, LCFlex is able to skip portions of its input sentences in order to search for the maximal portion for which it can construct an analysis. It allows the user to control how freely it is permitted to skip in searching for the maximal parsable subset of an input utterance. Like GLR*, LCFlex incorporates both inter-analysis skipping and intra-analysis skipping, and these two forms of skipping may be used either separately or together.

Inter-analysis skipping in LCFlex allows an analysis to begin or end at *any* vertex in the chart. In this way, the parser constructs analyses for every contiguous portion of the input where it is possible to do so. Inter-analysis skipping is accomplished in LCFlex by modifying its prediction mechanisms. Normally active edges are created wherever their corresponding left hand side category is predicted by LCFlex's top-down and bottom-up prediction mechanisms. When inter-analysis skipping is enabled, the goal category (or categories) are predicted at each vertex in the chart in addition to the categories predicted based on the active edges ending at those vertices. This allows full analyses to begin at any vertex in the chart. When the whole utterance has been processed, analyses can be extracted from the chart by searching for inactive edges with one of the goal categories as their category. When inter-analysis skipping is turned on, analyses that begin and end at any vertex are returned, rather than only the analyses that begin with the first word and end with the last word. From the extracted list of analyses it is possible to return the subset of largest analyses that span the maximum subset of the utterance. In this way, LCFlex can perform like other well-known partial parsers.

Often inter-analysis skipping is inadequate because it fails to handle the disfluencies that appear *within* an analysis. To handle these cases, the second type of skipping, *intra-analysis skipping*, allows the parser to skip words *within* an analysis. Intra-analysis skipping in LCFlex is accomplished by copying references to active edges forward in the chart, in effect skipping over all of the words in between the current vertex and the vertices where the edges were copied from. Before each word is parsed, active edges from some number of previous vertices are copied forward to the current vertex. The exact number of previous vertices from which active edges are copied is determined by the parser's two

different beams, described below, that bound the number of analyses pursued by the parser in parallel. An alternative to copying active edges forward would have been to search backwards in the chart for active edges to extend each time an inactive edge is inserted. This would have appeared superficially more similar to the approach taken in GLR*. However, this makes it necessary to do the work of skipping over the same portion of the sentence each time an inactive edge of a particular category is inserted into the chart beginning at the same vertex. Approaching the problem of intra-analysis skipping by copying active edges forward allows the parser to do the work of skipping each subset of the sentence only once rather than multiple times. Note also that intra-analysis skipping could have also been accomplished by copying active edges forward in the chart, allowing us to handle both types of skipping with a single mechanism. However, our intra-analysis skipping approach introduces far less ambiguity than copying active edges forward in the chart. Additionally, our experience has revealed that even the noisiest input sentences tend to be composed of mostly grammatical islands surrounded by noise. If we treat these two types of skipping separately, we can achieve the same robustness with a lower threshold on active edge copying and thus less computational cost.

We have experimented with two types of beams in LCFLex to control the amount of skipping. The first type of beam, *maximum of n beam*, allows the user to specify how many words can be skipped within an single analysis. The second type of beam, *n-best beam* allows the user to specify a number that constrains the number of analyses it pursues in parallel, as in GLR*. It is not a true n-best beam in the sense that it limits the absolute number of analyses pursued in parallel. Instead, it approximates this by bounding the number of active edges that are allowed to end at any one vertex in the chart. These two different beams can be used separately or together. Thus, LCFLex allows a great deal of control over its skipping behavior.

5.3 EVALUATING WORD SKIPPING

We conducted a series of empirical tests to evaluate the effect of the word skipping flexibility allowed by the parser on its runtime performance. Our evaluation here demonstrates that LCFLex maintains the order of magnitude computational advantage of left-corner parsers over GLR parsers. For both GLR* and LCFLex, the amount of word skipping flexibility is controlled via the parser's search beam. We compared the alternative settings of the two parsers based both on parse times and on translation quality of text produced automatically by our gen-

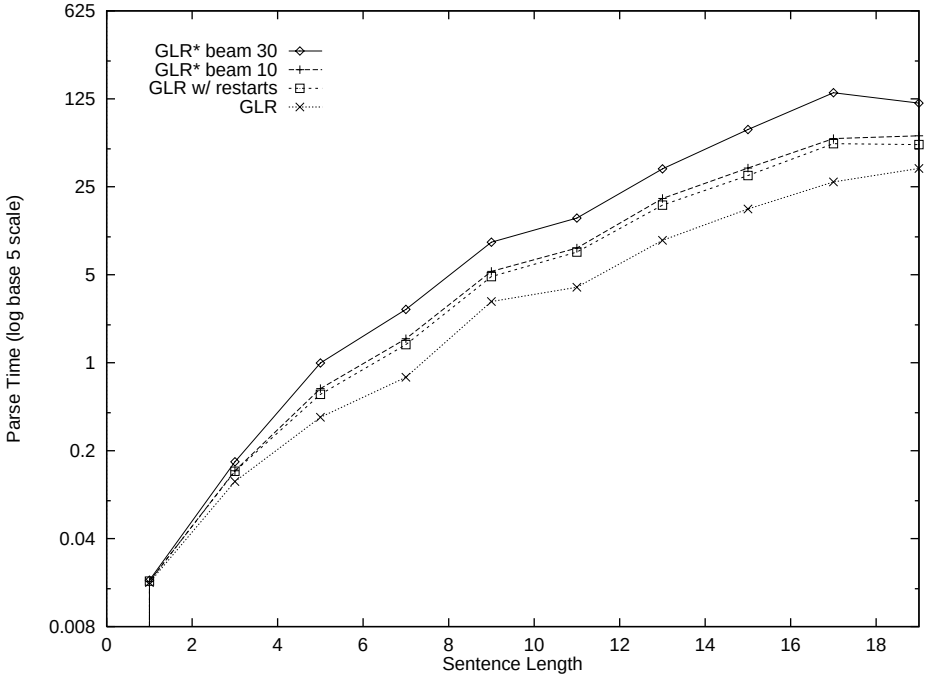


Figure 1.2 **Avg Parse Time vs. Sentence Length for GLR*, Various Skipping Settings**

	Bad	OK	Perfect	Total Acceptable
No Robustness	47.16	11.67	41.17	52.84 (60.39)
Restarts	32.83	21.00	46.17	67.17 (76.77)
GLR* beam 10	31.50	21.00	47.50	68.50 (78.29)
GLR* beam 30	30.33	19.50	50.17	69.67 (79.62)
LCFLEX 2 Level	29.00	20.50	50.50	71.00 (81.14)
LCFLEX Skip 3	26.67	20.00	53.33	73.33 (83.80)

Table 1.1 **Translation Quality with Alternative Parsers**

eration component from the parser output. In all cases, LCFLEX performs at least an order of magnitude more efficiently than GLR* set to a roughly equivalent flexibility setting. The complete test set contains 607 sentences of transcribed spontaneous speech from the ESST task. The distribution of sentence lengths is found in Figure 1.5. Of these 607 sentences, a subset of 200 sentences was selected for grading. These 200 sentences were contiguous sentences from five dialogues in

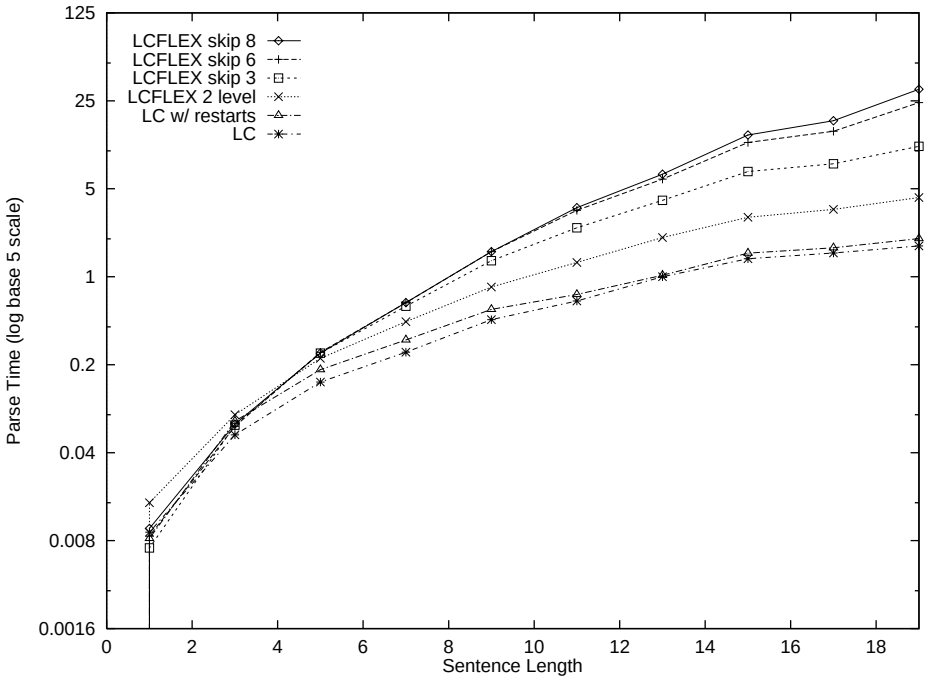


Figure 1.3 **Avg Parse Time vs. Sentence Length for** LCFLEX**, Various Skipping Settings**

our ESST corpus. We selected this set of contiguous sentences in order to ensure a realistic distribution of sentence lengths and in-domain vs. out-of-domain sentences.

Each translation was graded by three independent judges, receiving a grade of Perfect, OK, or Bad. For each strategy, the number of sentences assigned each grade was totaled for each grader. These totals were then averaged to arrive at the percentages presented in Table 1.1. Of the 200 sentences in the test set, 25 were determined to be out of domain. Thus, the percentages in parentheses in the final column are percentages of acceptable translations out of the in-domain sentences.

In the case of GLR*, we tested the parser in four different settings: (1) "GLR": with no skipping allowed - this is equivalent to the original (non-robust) GLR parser; (2) "GLR with restarts": allowing only inter-analysis skipping (of prefixes and suffixes of the input string); (3) "GLR* beam 10": allowing inter-analysis skipping with the parser using a search beam of size 10; and (4) "GLR* beam 30": as in (3), but with a search beam of 30. Average parse times as a function of sentence length for all four tests appear in Figure 1.2. As expected, greater levels of skipping

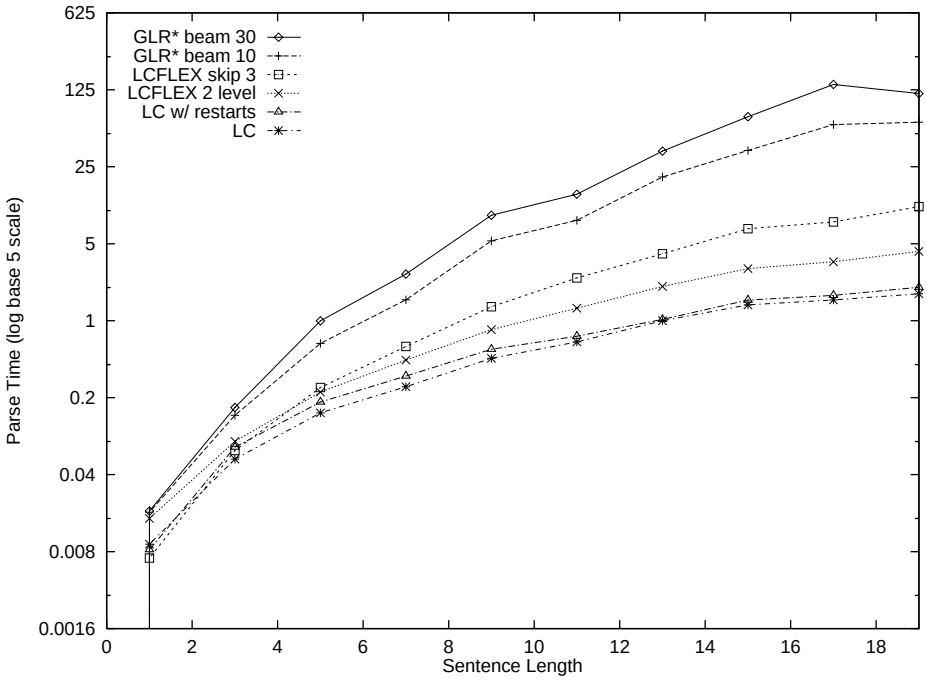


Figure 1.4 **Avg Parse Time vs. Sentence Length for** LCFlex **compared with GLR***

flexibility result in increasing parse times. It is interesting to note that GLR* with a beam of 10 is only slightly more time consuming than just allowing restarts, while already allowing a significant amount of intra-analysis skipping. In fact, our experiments indicated that GLR* with a beam of 10 was usually flexible enough to find a correct analysis when such an analysis can be recovered with word skipping (see also the translation quality evaluation results in Figure 1.1). A beam of 30 already results in significantly longer parse times, approaching infeasible levels for online parsing with sentences of length greater than 15 words.

For LCFlex, we conducted a similar set of experiments, with varying amounts of skipping flexibility. In LCFlex, the amount of skipping flexibility is controlled by a global skip parameter, which determines the total number of words allowed to be skipped in the process of obtaining an analysis. We tested the parser with five different settings: (1) "LC": no skipping allowed - equivalent to a non-robust left-corner parser; (2) "LC with restarts": allowing only inter-analysis skipping (of prefixes and suffixes of the input string); (3) "LCFlex skip 3": a maximum of 3 words can be skipped within an analysis; (4) "LCFlex skip 6":

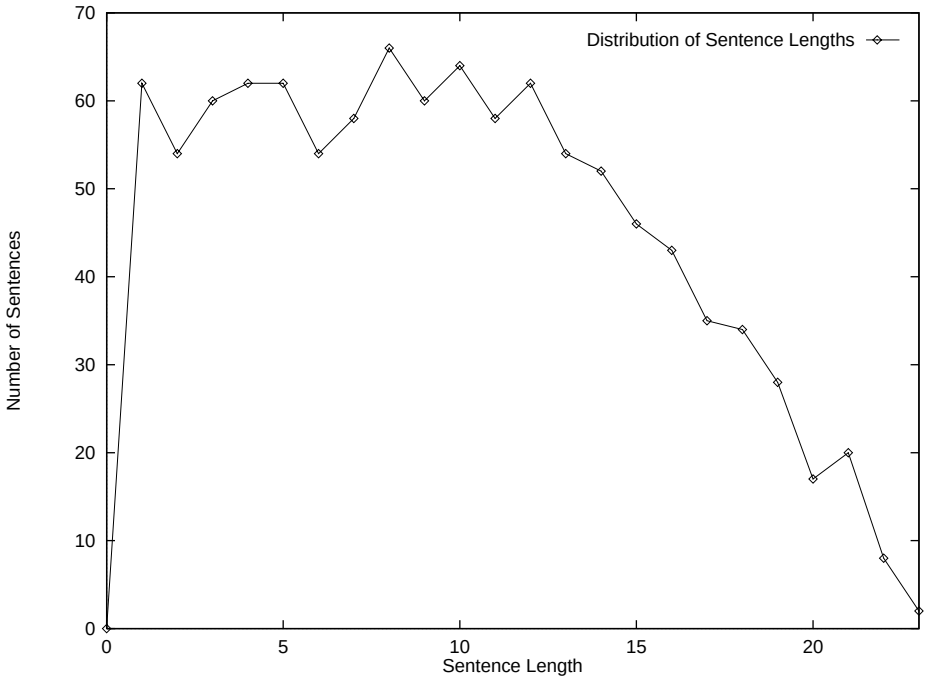


Figure 1.5 **Distribution of Sentence Lengths**

similar, with a maximum of 6 skipped words; and (5) "LCFLEX skip 8": similar, with a maximum of 8 skipped words. In addition to our beam comparison tests, we also tested LCFLEX in its two level parsing mode discussed below in Section 8. in order to demonstrate the increased efficiency possible with that mode.

Our purpose in testing different beam settings for both parsers was to demonstrate that LCFLEX maintains the computational advantage of left-corner chart parsers over GLR parsers even as the flexibility in both parsers is increased. Because GLR*'s beam is based on the nature of its Graph Structured Stack, which is not a part of LCFLEX, it is impossible to constrain the two parsers to skip exactly the same amount or in exactly the same places. However, in practice, setting LCFLEX's maximum skip beam to 3 causes it to closely mimic GLR*'s beam of 30. Setting its maximum skip beam to 6 or 8 causes it to allow more skipping than GLR* with a beam of 30. While it would have been possible to re-implement GLR*'s beam to control skipping based on number of words skipped, as in LCFLEX, our informal investigations indicate that controlling GLR*'s skipping in this way causes it to parse far less efficiently than with its current beam. Thus, our current results are suf-

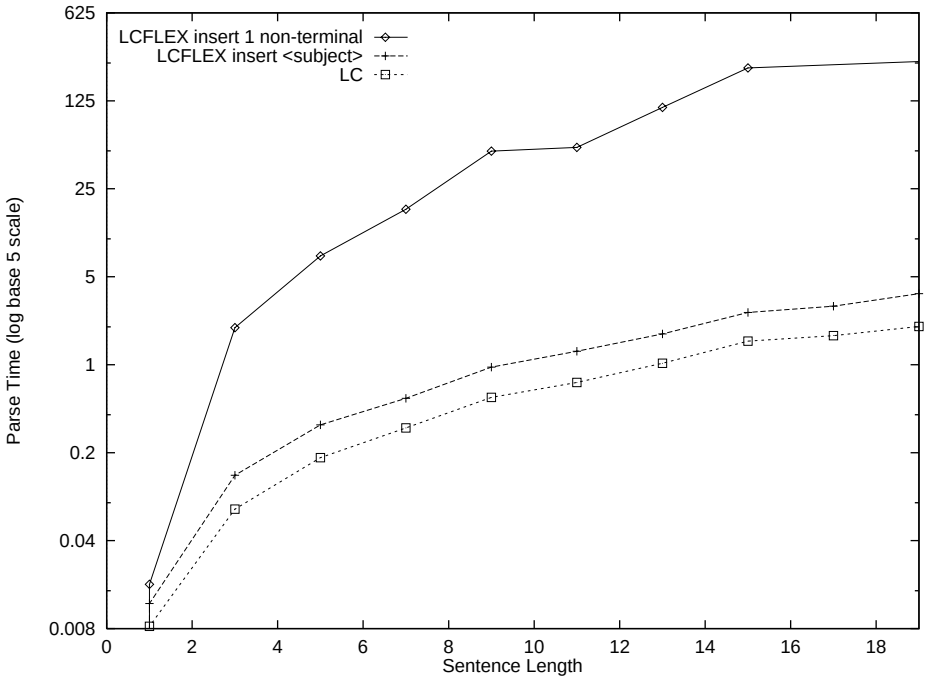


Figure 1.6 **Avg Parse Time vs. Sentence Length for** LCFlex**, Various Insertion Settings**

ficient to demonstrate that LCFlex does maintain the computational advantage of left-corner parsers over GLR parsers.

Average parse times as a function of sentence length for all tests appear in Figure 1.3. As expected, greater levels of skipping flexibility result in increasing parse times. However, for LCFlex, inter-analysis skipping does not result in a very significant increase in parse time, while already providing a good deal of robustness. Two level parsing performs slightly slower while achieving another sizable increase in robustness. While a skip parameter value of 3 does incur a significant time cost, parse times remain feasible for even relatively long sentences. Increasing the skip parameter above 3 did not improve the overall translation quality produced.

Note that with all of the above settings, LCFlex is at least an order of magnitude faster than GLR* in comparable settings. (See Figure 1.4.) LCFlex with a skip value of 8 is about similar in speed to GLR with no skipping at all.

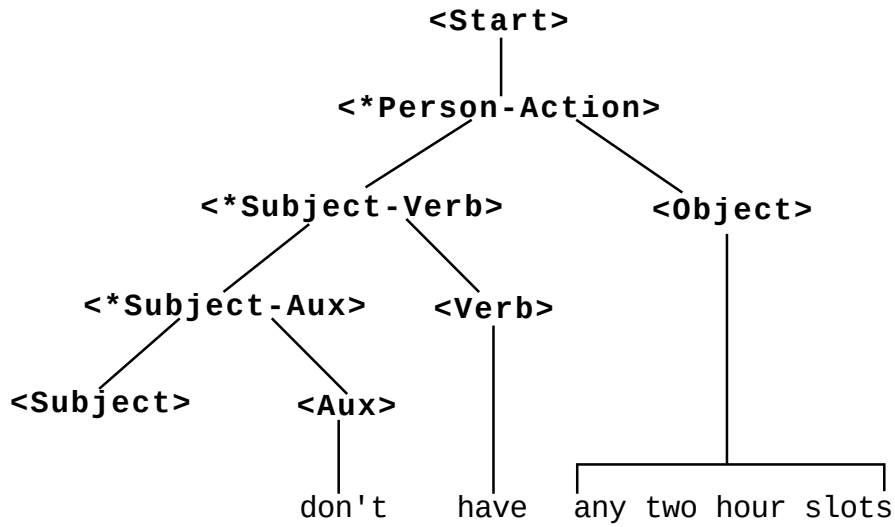
6. INSERTING IN LCFLEX

While in our experience, skipping is the most important flexibility feature for efficient robust parsing, under certain circumstances it is still not enough. Such is the case when a speaker uses an abbreviated form that is not grammatical, for example when a determiner is dropped in informal language. (i.e., "cat at nine o'clock" to indicate the location of an approaching cat). While such constructions can be modeled within the grammar, doing so leads to an unnecessary explosion in ambiguity. In the case of determiner dropping, two analyses would always be constructed for every noun phrase normally requiring a determiner, one including the determiner and one omitting it. Even in cases where a correct analysis is constructed, the incorrect one (without the determiner) would be constructed as well, doubling the number of analyses that contain a representation for the associated noun phrase. If these types of ungrammaticalities are handled within the parser instead, for instance by allowing the parser to insert words or non-terminals in order to complete a partial rule match, then two analyses will still be constructed for the noun phrase in question, but the one with the greater error can eventually be pruned. Because the parser is aware of the fact that the analysis without a determiner is less correct than the one with a determiner, it can prevent the explosion of ambiguity that arises from modeling this type of ungrammaticality within the grammar.

6.1 LCFLEX'S LIMITED INSERTIONS

While insertions have been constrained in previous approaches by limiting the number of inserted non-terminals, as with Lehman's prioritized agenda mechanism (Lehman, 1989), LCFlex allows the user to control the number of occurrences of each specific non-terminal that may be inserted. Thus, the user may limit insertions to just a small number of a handful of categories. We conducted a set of experiments to evaluate the effect of word and category insertions on the runtime performance of the LCFlex parser. As mentioned earlier, word and category insertion is a very costly form of flexibility, as it dramatically increases the search space of analyses that must be considered by the parser. While it has been demonstrated in (Rosé, 1997) that even limited numbers of general insertions are infeasible with a practical sized grammar, we demonstrate here that LCFlex's style of limited insertions does not demand too high of a price in speed.

Insertions in LCFlex are primarily controlled by setting a global parameter and then recompiling the grammar. When insertions are turned on, the global variable that controls this behavior is an association list of



RULE: ($< *SUBJECT - AUX > (< SUBJECT > < AUX >)$)

CHILDREN:
$< SUBJECT >$ ((DUMMY +))
$< AUX >$ ((ROOT DO) (NEGATIVE +))

RESULT:
$< *SUBJECT - AUX >$ ((SENTENCE-TYPE *STATE) (VERB-FORM2 BASE)
(AUX DO) (DUMMY +))

RULE: ($< *SUBJECT - VERB > (< *SUBJECT - AUX > < VERB >)$)

CHILDREN:
$< *SUBJECT - AUX >$ ((VERB-FORM2 BASE) (SENTENCE-TYPE *STATE)
(AUX DO) (DUMMY +))
$< VERB >$ ((VERB-FORM BASE) (ROOT HAVE))

RESULT:
$< *SUBJECT - VERB >$ ((VERB ((ROOT HAVE))) (ADVERB *DUMMY*)
(NEGATIVE +) (SENTENCE-TYPE *STATE)
(AUX DO) (DUMMY +))

Figure 1.7 **Insertion Example**

non-terminals paired with integers or the symbol `*any*`. This indicates that any combination of the specified maximum number of the specified non-terminals may be inserted into an analysis. For example, `((`$< np >$ `1) (`$< det >$ `*any*))` specifies that at most one noun phrase and any number of determiners may be inserted. `*or*` may be used to spec-

ify a disjunction of constraints. Thus, `(*or* ((`$< np >$ `1) (`$< det >$ `*any*)) ((`$< copula >$ `1)))` indicates that either at most one copula or any determiner and at most one noun phrase may be inserted.

One example where inserting is necessary in our ESST task is where speakers drop the subject as in, "Don't have any two hour slots." Because our ESST grammar does not allow the speaker to drop the subject in constructions such as this one, this sentence would ordinarily not be able to parse. Figure 1.7 displays the correct c-structure for this construction according to our ESST grammar. This sentence can only parse by inserting a $< subject >$ non-terminal before the $< aux >$ non-terminal under the $< *subject - aux >$ non-terminal. When a non-terminal is inserted, a "dummy" feature structure is inserted into the analysis in place of the one that would have normally been constructed by the parser for the corresponding input words. The dummy feature structure has the feature `dummy` with a value of `+`. This dummy feature ensures that inserting a dummy feature structure will not cause unification constraint equations to fail. Whenever a unification equation refers to an empty slot within a dummy feature structure, the value that is returned is `*dummy*`. Constraint equations involving a `*dummy*` value never fail. Figure 1.7 shows the result of two unifications involving dummy feature structures. In the first rule application, The dummy feature structure is unified with the result so that the resulting feature structure is also a dummy feature structure since it contains the `dummy` feature with value `+`. Additional constraint equations in the unification augmentation part of the rule, not shown, cause other features of the dummy feature structure to be assigned values. In the second rule application, the value of the `adverb` feature of the dummy feature structure is passed up to the result. Since the `adverb` slot of the dummy feature structure is empty, the value `*dummy*` is assigned. Thus, subsequent constraint equations testing the value of the `adverb` feature of the resulting feature structure will not fail.

Allowing the parser to insert a single non-terminal symbol only slightly decreases the parser's efficiency. To demonstrate this, we evaluated the effect of inserting one particular grammar category, $< subject >$, and compared it with allowing the insertion of any one category into the analysis. We picked the $< subject >$ category because we have evidence of a few sentences, such as the example above, in which the insertion of this particular category is in fact necessary for recovering the correct analysis. Average parse times as a function of sentence length for these tests appear in Figure 1.6. As can be seen, the insertion of the $< subject >$ category does result in a noticeable, but tolerable, effect

on runtime performance. However, the insertion of even one arbitrary category results in infeasible parse times.

6.2 INSERTING NON-TERMINALS IN AN LC FRAMEWORK

Insertions affect every aspect of how the parsing algorithm proceeds, including the conditions under which active edges are created, extended, and considered complete. Because its impact on the operation of the parser is so pervasive, inserting comes at a greater computational expense than skipping. Normally active edges corresponding to rules are created when an inactive edge of the category of the associated rule's first right hand side daughter are inserted into the chart. With unconstrained insertions, active edges would be created whenever an inactive edge of the category of any of the associated rule's right hand side daughters is inserted into the chart. If insertions are constrained, either in number or in terms of which non-terminals are allowed to be inserted, then rather than doing this for every right hand side category, the parser would only do this for the category of every right hand side daughter such that the constraints on insertions allow the preceding right hand side daughters to be inserted. Thus, when insertions are enabled, top-down predictions are computed with a modified version of the left-corner relation that is computed not only between the left hand side category of a rule and the category of the first right hand side daughter, but instead between the left hand side category and the category of any right hand side daughter that can be the "trigger" for creating an active edge.

Just as one or more non-terminal symbols may be inserted at the beginning of a rule match, one or more non-terminals may be inserted at the end. Thus, insertions also affect the conditions under which an edge may be considered complete, and thus become inactive. As long as inserting the remaining "needed" categories does not cause the constraints on insertions (if any) to be broken, an edge may become inactive. However, if the "needed" list is non-empty at this point, the corresponding active edge remains in the chart in case any of the remaining "needed" categories may be constructed and thus make it possible to create a similar inactive edge with a smaller insertion error.

Non-terminals may also be inserted in the middle of rule matches. Thus, inserting also affects the way in which active edges are extended. Normally, an active edge ending at vertex i with x as the next needed category will only be extended when an inactive edge of category x is inserted in the chart beginning at vertex i. With insertions enabled, an active edge may be extended when an inactive edge of category x is

inserted beginning at vertex i as long as x is any one of its "needed" categories and the constraints on insertion allow the insertion of the categories preceding x on the "needed" list.

7. SELECTIVE FLEXIBLE FEATURE UNIFICATION IN LCFLEX

Skipping and inserting lend flexibility to the manner in which LCFlex constructs c-structures for sentences from the context-free backbone portion of its LFG-like grammar formalism. Similarly, its selective flexible unification adds flexibility to the manner in which the grammar formalism's unification augmentations are evaluated. LCFlex's unification augmentations serve the dual purpose of enforcing grammatical constraints such as agreement and building a feature structure representation for the sentence. To the extent that feature unification is used in any grammar to eliminate undesired analyses allowed by the context-free portion of the grammar, flexible unification can be used to allow these less desirable analyses in the case where it is impossible to construct a parse otherwise.

7.1 WHY SELECTIVE FLEXIBLE FEATURE UNIFICATION

Because the unification augmentations of our LFG-like grammar formalism are used to eliminate a large number of incorrect analyses, unconstrained flexible unification would cause a massive explosion of ambiguity. We address this problem primarily by allowing users to specify which features LCFlex should treat as "soft features" and in which order they should be evaluated. Our evaluation demonstrates that setting aside a small number of features as "soft features" does not dramatically decrease the parser's overall efficiency.

LCFlex allows its users to specify which features to set aside as "soft features" by setting a global variable. When flexible unification is turned on, the global variable that controls this behavior stores a list of lists of soft features. Each list of features represents a separate unification stage. At parse time, when an edge becomes complete and the unification part of the rule is fired, it fires in stages according to the division of features in this list of lists. With each successive stage, the resulting feature structure becomes further specified with the result of unifications involving the associated features. The parser fires successive stages until it either finishes the complete set of stages or one stage fails. At that point, it returns the result from the last successful unification stage. If

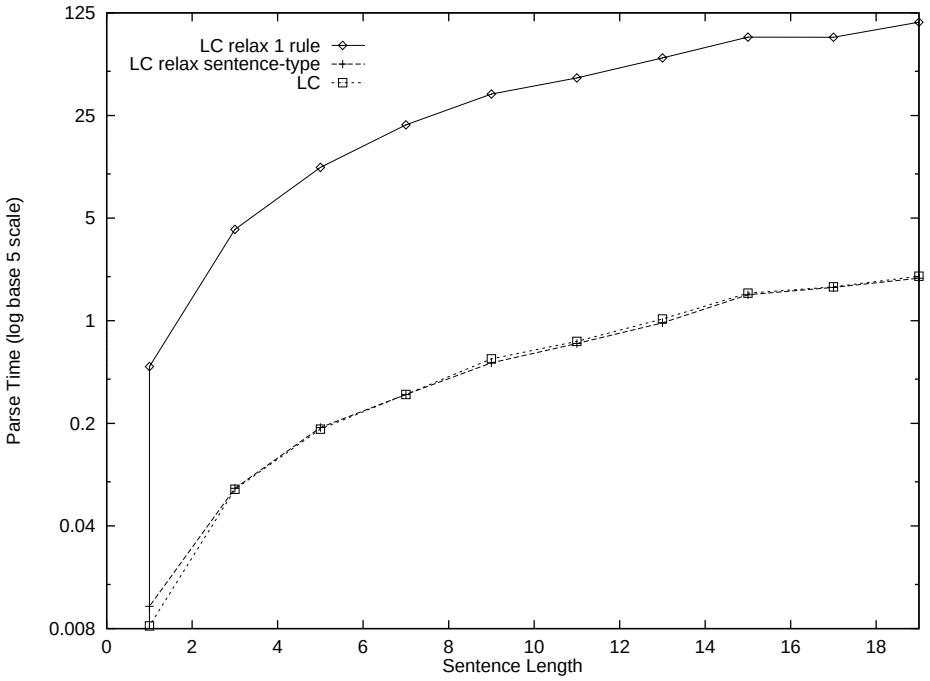


Figure 1.8 **Avg Parse Time vs. Sentence Length for** LCFLEX**, Various Flexible Unification Settings**

it fails in one of its unification stages, it also returns an indication of which stage failed.

An example where flexible feature unification is useful is the sentence "How the week after look?", as displayed in Figure 1.9. The problem with this sentence is that the auxiliary "does" has been dropped. Our ESST grammar allows "How" to be attached to a yes/no question in order to form a wh-question. Without the "does", the text "the week after look" still parses as a sentence, but it is analyzed as a statement rather than a question. Thus, the rule that would normally attach the "How" in order to form a wh-question fails when it tests the `sentence-type` feature. This sentence can be handled by relaxing the `sentence-type` feature. Relaxing a single feature such as `sentence-type` only slightly decreases the parser's efficiency, as demonstrated in Figure 1.8. In this evaluation, relaxing a single specific feature was compared with allowing a single rule application to fail during unification for any analysis. In the case where a rule application failed, a dummy feature structure was returned as the result from the unification. Notice how much more efficient relaxing one specific feature is than allowing a single rule application to

RULE: ($< SENTENCE >$ ($< WH-WORD >$ $< *SUBJ-BE-LOOK >$))

CHILDREN:
$< WH-WORD >$ ((SENTENCE-TYPE2 *QUERY-IF)
(SENTENCE-TYPE *QUERY-REF)
(WHAT ((FRAME *HOW))))

$< *SUBJ-BE-LOOK >$ ((VERB ((ROOT LOOK)))
(SENTENCE-TYPE *STATE)
(TYPE TEMPORAL)
(WHEN ((SPECIFIER (*MULTIPLE* DEFINITE
FOLLOWING))
(NAME WEEK)
(FRAME *SPECIAL-TIME))))

RESULT:
$< SENTENCE >$ ((FRAME *HOW) (NO-WHEN-0 +)
(A-SPEECH-ACT *SUGGEST)
(SENTENCE-TYPE *QUERY-REF)
(WHEN ((SPECIFIER (*MULTIPLE* DEFINITE
FOLLOWING))
(NAME WEEK)
(FRAME *SPECIAL-TIME))))

Figure 1.9 **Flexible Unification Example**

fail. This demonstrates the effectiveness of selective flexible feature unification in increasing the parser's flexibility without compromising its efficiency.

One problem with allowing features to "fail soft" is that it may cause an infinite recursion in the grammar. Thus, in addition to allowing the user to specify which features should be treated as "soft features", LCFlex allows the user to specify how many rules may "fail soft" within a single parse. This is accomplished by setting a separate global variable. By limiting the set of "soft features" and possibly the number of rules allowed to "fail soft" within a single analysis, flexible unification can be handled in an efficient manner.

7.2 IMPLEMENTING SELECTIVE FLEXIBLE FEATURE UNIFICATION

Selective flexible unification is accomplished in LCFlex by modifying the way the grammar's unification augmentations are compiled into lisp code. When the grammar is compiled, and each grammar rule's unification augmentation is compiled into a lisp function, each constraint

equation is placed into one of several stages. Those constraint equations not involving any of the soft features are placed in the first stage. Those involving features from the first list but not involving any features from the subsequent lists are placed in the second stage, and so on. Each list of constraint equations is compiled separately into lisp code. This lisp code is then placed in an overarching `case` statement such that which set of constraint equations is applied is dependent upon the stage. This `case` statement is then placed in a lisp function associated with the grammar rule from which the original list of constraint equations was taken. At parse time, the stage is first set to 0 and the function is evaluated. Thus, all of the constraint equations not involving any soft features are evaluated. If that unification fails, no result is returned. The unification is said to have failed. If it does not fail, however, a result will be returned regardless of what happens with unification at subsequent stages. The stage is then set to 1 and the previous result is passed back into the lisp function. As the constraint equations associated with stage 2 are applied, they have the affect of elaborating the result from the previous stage that was passed in. If this unification fails, the previous result is returned with an associated flexible unification error code of 1 that gets inserted into the resulting edge's `unif` slot. If it does not fail, the stage is set to 2 and process cycles again. This continues until either unification fails at some stage or the function has been evaluated for every stage. The most complete result is returned.

8. TWO LEVEL PARSING IN LCFLEX

Two level parsing further enhances LCFlex's efficiency by allowing flexibility to be applied to a sentence on an "as needed" basis. The simple idea behind it is to parse a sentence first at a constrained level of flexibility, and then to combine the best partial analyses constructed in the first pass at a higher level of flexibility. The largest partial analyses are extracted from the chart using a greedy "shortest path" algorithm. In the second stage, inactive edges corresponding to the best partial analyses constructed in the first pass are inserted in the second pass the same way that edges corresponding to lexical entries were inserted in the first pass. Thus, the effective "sentence length" in the second stage is shorter than that in the first stage. Because the additional flexibility is only applied to the places in the analysis where it is really needed (i.e., that would not parse at the lower level of flexibility), the overall parse time and number of edges created is smaller with two level parsing than with one level parsing.

We conducted a preliminary test of LCFlex's two level parsing mode. In the first stage, we enabled inter-analysis skipping and intra-analysis skipping with a maximum skip limit of 1. In the second stage we increased the maximum skip limit to 6. The run time performance and translation quality for this two level parsing experiment were already reported above in Section 5.3.

9. MANAGING AMBIGUITY

Increasing the parser's flexibility also increases the amount of ambiguity produced by it. Consequently, an increase in flexibility in general slows down the parser and makes selecting the best analysis a more challenging task. Thus, while managing ambiguity is an orthogonal issue to robustness, it is intimately related. Any practical robust parser must manage ambiguity effectively. Both in GLR* and LCFlex we approach the efficiency issue with local ambiguity packing and pruning, and the selection issue with statistical disambiguation. Specifics of GLR*'s ambiguity packing, pruning, and statistical disambiguation are described in depth elsewhere (Lavie, 1995). Thus, we focus here on how these techniques are incorporated into LCFlex.

9.1 AMBIGUITY PACKING AND PRUNING

Because LCFlex is a chart parser, it keeps track of the parse trees constructed in the course of the parsing process via inactive edges stored in its chart. Whenever a grammar rule is successfully applied, a corresponding inactive edge is created. That inactive edge has the rule's left hand side category as its category, its first child's starting position as its starting position, and its last child's ending position as its ending position. In order to make parsing as efficient as possible, LCFlex attempts to pack together all edges of the same category that have the same starting and ending positions. Thus, it attempts to pack together all analyses of the same span of text that result in the same category.

Packing edges together reduces the number of rule applications the parser must attempt, thus reducing both the parser's time and space requirements. However, the resulting packed edges have much more complex feature structures than those of non-packed edges. And rule applications involve evaluating unification equations over the feature structures of the child edges. Thus, while the parser will have fewer rule applications with ambiguity packing turned on, each rule application involving packed edges will take longer to complete than a corresponding rule application on non-packed child edges. Increasing the parser's flexibility along any dimension exacerbates this problem since it increases

the number of analyses of the same category that span the same segment of text. Thus, while ambiguity packing alone results in a significant savings in time in traditional non-robust parsers, it is not sufficient in robust parsers even with limited flexibility.

This problem is addressed in LCFlex, as it was in GLR*, by pruning local analyses that will almost definitely result in global analyses of lesser quality. While pruning is attractive because it directly addresses the efficiency issue, it is risky in unification augmented grammar formalism's such as LCFlex's because it is not clear on a local level which analyses will be eliminated through unification in subsequent rule applications. Thus, it is dangerous to base decisions about which analyses to prune only on the statistical score of the context free portion of the analysis. At the same time, it is not reliable to prune based on the magnitude of the error level (i.e., how many words skipped, how many non-terminals inserted, how many flexible unifications) of local analyses since an analysis with a lower error locally may only be able to be included in analyses that have a higher global error level than other local analyses with a locally higher error level.

LCFlex prunes based on a heuristic adapted from GLR*. Analyses are pruned that have errors that strictly subsume errors in other analyses of the same category over the same span of text. In GLR*, analyses were pruned that covered a strict subset of words that were covered by another analysis of the same category spanning the same text. Since LCFlex allows flexibility along more dimensions than GLR*, it needs a correspondingly more complex pruning heuristic that takes into account not only skipping but also insertions and unification relaxations. However, the same principle of eliminating analyses with errors that strictly subsume errors of otherwise similar analyses can be extended to this more complex case.

Pruning in LCFlex takes place in three stages. In the first stage, analyses are divided into sets of analyses that have the same insertion and flexible unification errors. Within these sets, pruning takes place based on skipping as in GLR*. Then the remaining analyses are combined into a single set which is then divided into subsets again, this time grouping those analyses that have the same skipping and inserting errors. This time, analyses within each subset are pruned if they contain flexible unification errors that subsume those of other analyses within the same subset. Finally, the remaining analyses are divided into subsets once again, this time according to skipping and flexible unification errors. Analyses with insertion errors that subsume those of other analyses within the same subset are then pruned.

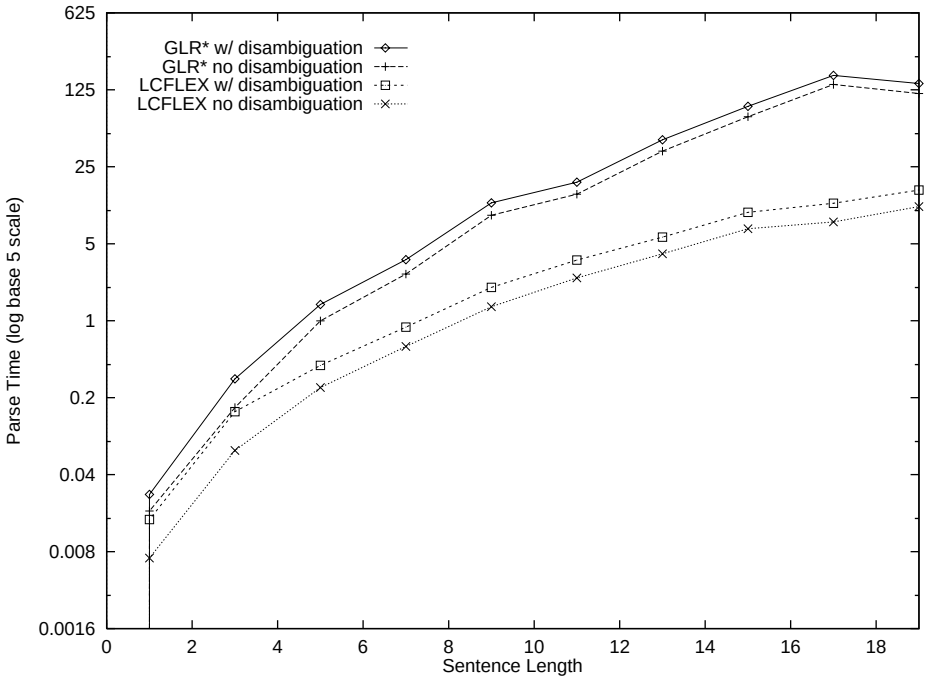


Figure 1.10 **Parse Times with and without Statistical Disambiguation**

9.2 STATISTICAL DISAMBIGUATION

As in GLR*, statistical disambiguation is used in LCFlex to select a single analysis from a packed list of analyses. Traditionally, probabilistic context free grammars assign probabilities to grammar rules. Recently, researchers have argued (Carroll and Briscoe, 1993; Lavie, 1995) that it is advantageous to assign probabilities to transitions between parser actions instead because it lends context-sensitivity to the score assigned to each rule application. So far that principle has only been implemented in the LR framework where probabilities can be associated with parser actions within individual states. Thus, the probability assigned to each parser action depends upon the state the parser was in when the action was performed. A similar context sensitivity is achieved in LCFlex by assigning probabilities to bigrams of rule applications. Thus, the probability of a rule application is dependent upon the previous rule application.

Sequences of rule applications are extracted from analyses similarly to how action sequences are extracted from analyses in GLR* (Lavie, 1995). A postfix procedure traverses the parse tree such that the extracted sequence of rule applications corresponds to the order in which

the rules were applied to build the parse tree. Rule transition bigrams are trained in LCFlex the same way action probabilities are trained in GLR*. A training corpus is used that contains sentences paired with correct feature structures. Each sentence is parsed. Then the list of produced analyses are searched for the one that produces the correct feature-structure. The sequence of rule applications is then extracted from this analysis. The counts for each of the rule application bigrams in the sequence are then incremented. After the entire corpus has been processed, the counts are smoothed using the Good Turing method. Finally, these counts are converted into probabilities.

While statistical disambiguation has been demonstrated to be an effective approach to selecting from multiple alternative analyses, it comes at a significant cost in terms of run time performance. Figure 1.10 displays parse times with and without statistical disambiguation for GLR* with a beam of 30 and LCFlex with a maximum skip limit of 3.

10. CONCLUSIONS AND FUTURE DIRECTIONS

This paper described our investigations into how to effectively balance robustness and computational efficiency in a general unified parsing framework for unification-based large-scale grammars. Our investigations were based on empirical evaluations conducted with two general robust parsers: GLR* - a robust GLR parser, and LCFlex - a left-corner chart parser. We focused on a spontaneous spoken language analysis task, where disfluent and extra-grammatical input is extremely common. Our investigation shows that flexible skipping, controlled via a beam search, is the most effective mechanism for achieving robustness at an acceptable level of computational cost. LCFlex can tolerate relatively high levels of skipping (and its resulting high levels of ambiguity), while still maintaining online parsing times. While the other types of flexibility investigated - category insertion and flexible unification - can help recover a correct parse in certain cases, their computation cost is very high. Our experiments show that these types of flexibility should be applied sparingly, with only a selective set of insertion categories and features. The specific appropriate sets of insertion categories and features for relaxation are grammar and domain dependent. Our flexible robust parsing framework, however, is general enough to work with any specified sets. One area of future research is the development of automatic methods for identifying appropriate insertion categories and features for relaxation, using machine learning techniques.

Another interesting direction for future research is multi-level parsing. The current LCFlex parser supports two-level parsing that allows it to construct a partial analysis at a more constrained level of flexibility and then combine the fragments of the partial parse by running the parser again at a far less constrained level of flexibility. The experiments we reported here demonstrated that the two-level mode can achieve an almost comparable level of robustness at a significantly lower computational cost. However, we suspect that the specific types and levels of flexibility that should be applied at each parsing stage are domain and grammar dependent. While our framework is general enough to support any division of labor between the parsing levels, we would like to explore methods for finding the optimal settings for a given grammar and domain automatically. We also would like to investigate whether adding additional parsing levels can result in further gains in efficiency.

As emphasized in this paper, effective parse scoring and disambiguation is essential in robust parsers such as GLR* and LCFlex. The framework for parse scoring and disambiguation in our current parsers does not model the various types of flexibility in a general uniform way. We would thus like to develop a parse scoring and disambiguation model that explicitly accounts for the various flexibility operations as well as for preferences for particular structures on both the constituent and feature levels. The relative fitness scores can then be applied to constrain and prune the parser search, as well as for parse selection and disambiguation. The relative preference (or cost) for applying certain types of flexibility should also be accounted for via the parse scoring model. A more unified parse evaluation framework would also allow us to better apply machine learning techniques, in order to automatically derive the set of optimal parameters for any specific domain or application.

11. ACKNOWLEDGEMENTS

Special thanks are due to all of those patient individuals who used LCFlex during its beta testing phase and sent us many helpful bug reports. David Schneider was the first to license a copy of LCFlex for a Written English as a Second Language tutoring system. His research on detecting syntactic errors provided the inspiration for LCFlex's limited insertions and selective flexible unification. In particular we'd like to thank the students in our Robust Parsing seminar class for providing us with stimulating discussions and getting involved in extending LCFlex's capabilities. Finally, we'd like to thank the many other people who have encouraged us to continue to develop this work, especially

the members of CIRCLE[2] who's vision for dialogue-based tutoring systems inspires us to continue to work towards the goal of robust and efficient natural language understanding. Our deepest gratitude belongs to our families, in particular Eric, Rachel, and Bob, without who's loving support this work would not be possible.

This research was supported in part by the Office of Naval Research, Cognitive and Neural Sciences Division (Grants N00014-91-J-1694 and N00014-93-I-0812) and the National Science Foundation (Grant IRI-94-57637).

[2]CIRCLE is the NSF funded center for intelligent tutoring housed at Carnegie Mellon University and the University of Pittsburgh.

References

Abney, S. (1996). Partial parsing via finite-state cascades. In *Proceedings of the Eighth European Summer School In Logic, Language and Information, Prague, Czech Republic*.

Aho, A. V. and Johnson, S. (1974). LR Parsing. *Computing Surveys*, 6(2):99–124.

Ait-Mokhtar, S. and Chanod, J. (1997). Incremental finite-state parsing. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*.

Alshawi (ed.), H. (1992). *The Core Language Engine*. MIT Press, Cambridge, MA.

Bod, R. (1998). Spoken dialogue interpretation with the DOP model. In *Proceedings of COLING/ACL-98*.

Buo, F. D. (1996). Feaspar - a feature structure parser learning to parse spoken language. In *Proceedings of COLING-96*.

Carroll, J. and Briscoe, T. (1993). Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1).

Carroll, J. A. (1993). *Practical Unification-Based Parsing of Natural Language*. PhD thesis, University of Cambridge, Computer Laboratory.

Gazdar, G., E., K., Pullum, G., and Sag, I. (1985). *Generalized Phrase Structured Grammar*. Blackwell, Oxford, UK.

Goodman, J. (1996). Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*.

Henderson, J. and Lane, P. (1998). A connectionist architecture for learning to parse. In *Proceedings of COLING/ACL-98*.

Hipp, D. R. (1992). *Design and Development of Spoken Natural-Language Dialog Parsing Systems*. PhD thesis, Dept. of Computer Science, Duke University.

Hobbs, J. R., Appelt, D. E., Bear, J., and Tyson, M. (1991). Robust processing of real-world natural-language texts. Technical report, SRI International.

Jain, A. N. (1991). *PARSEC: A Connectionist Learning Architecture for Parsing Speech*. PhD thesis, School of Computer Science, Carnegie Mellon University.

Jain, A. N. and Waibel, A. H. (1990). Incremental parsing by modular recurrent connectionist networks. In Tourertzky, D. S., editor, *Advances in Neural Information Processing 2*. Morgan Kaufman Publishers.

Kaplan, R. and Bresnan, J. (1982). Lexical-Functional Grammar: A Formal System for Grammatical Representation. In *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press.

Lang, B. (1974). Deterministic Techniques for Efficient Non-deterministic Parsers. In *Proceedings of 2nd Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 255–269, Saarbruken, Germany. Springer Verlag.

Lavie, A. (1995). *A Grammar Based Robust Parser For Spontaneous Speech*. PhD thesis, School of Computer Science, Carnegie Mellon University.

Lehman, J. F. (1989). *Adaptive Parsing: Self-Extending Natural Language Interfaces*. PhD thesis, School of Computer Science, Carnegie Mellon University.

Magerman, D. M. and Marcus, M. P. (1990). Parsing a natural language using mutual information statistics. In *Proceedings of AAAI*.

Mayfield, L., Gavaldà, M., Seo, Y.-H., Suhm, B., Ward, W., and Waibel, A. (1995a). Parsing real input in janus: A concept-based approach to spoken language translation. In *Proceedings of the Theoretical and Methodological Issues in Machine Translation*.

Mayfield, L., Gavaldà, M., Ward, W., and Waibel, A. (1995b). Concept-based speech translation. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'95)*.

Mayfield, L., Gavaldà, M., Ward, W., and Waible, A. (1995c). Concept-base speech translation. In *Proceedings of the IEEE 1995 International Conference on Acoustics, Speech, and Signal Processing*.

McDonald, D. (1990). Robust partial-parsing through incremental, multi-level processing: Rationales and biases. In *Proceedings of the AAAI Spring Symposium on Text-Based Intelligent Systems: Current Research in Text Analysis, Information Extraction, and Retrieval*. Paul S.

Jacobs, ed., A technical report from the GE Research and Development Center, Schnectedy NY, no 90CRD198.

McDonald, D. (1992). An efficient chart-based algorithm for partial-parsing of unrestricted texts. In *Proceedings of the 3rd Conference on Applied Natural Language Processing.*

McDonald, D. (1993a). Efficiently parsing large corpora. In *submitted to the ACL Workshop on Very Large Corpora: Academic and Industrial Perspectives.*

McDonald, D. (1993b). The interplay of syntactic and semantic node labels in partial parsing. In *Proceedings of the Third International Workshop on Parsing Technologies.*

Miller, S., Stallard, D., Bobrow, R., and Schwartz, R. (1996). A fully statistical approach to natural language interfaces. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics.*

Neumann, G., Backofen, R., Baur, J., Becker, M., and Braun, C. (1997). An information extraction core system for real world german text processing. In *Proceedings of the Fifth Conference on Applied Natural Language Processing.*

Pietra, S., Epstein, M., Roukos, S., and Ward, T. (1997). Fertility models for statistical natural language understanding. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics.*

Pollard, C. and Sag, I. (1987). *Information Based Syntax and Semantics: Vol. 1 - Fundamentals.* University of Chicago Press, Chicago, IL.

Rosé, C. P. (1997). *Robust Interactive Dialogue Interpretation.* PhD thesis, School of Computer Science, Carnegie Mellon University.

Rosé, C. P. and Lavie, A. (1997). An efficient distribution of labor in a two stage robust interpretation process. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing.*

Rosé, C. P. and Waibel, A. (1997). Recovering from parser failures: A hybrid statistical/symbolic approach. In Klavans, J. and Resnik, P., editors, *The Balancing Act: Combining Symbolic and Statistical Approaches to Language Processing.* The MITPress.

Rosenkrantz, D. J. and Lewis, P. M. (1970). Deterministic left corner parsing. In *Procedings of the IEEE Conferenc eof the 11th Annual Symposium on Switching and Automata Theory.*

Sanker, A. and Gorin, A. (1993). Adaptive language acquisition in a multi-sensory device. *IEEE Transactions on Systems, Man, and Cybernetics.*

Schneider, D. and McCoy, K. F. (1998). Recognizing syntactic errors in the writing of second language learners. In *Proceedings of COLING/ACL 98*.

Tomita, M. (1986). *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*. Kluwer Academic Publishers.

Tomita, M. (1987). An Efficient Augmented Context-free Parsing Algorithm. *Computational Linguistics*, 13(1-2):31–46.

Tomita, M. (1990). The Generalized LR Parser/Compiler - Version 8.4. In *Proceedings of International Conference on Computational Linguistics (COLING'90)*, pages 59–63, Helsinki, Finland.

Van Noord, G. (1997). An efficient implementation of the head-corner parser. *Computational Linguistics*, 23(3).

Ward, W. (1989). Understanding spontaneous speech. In *Proceedings of the DARPA Speech and Natural Language Workshop*.

Worm, K. (1998). A model of robust processing of spontaneous speech by integrating viable fragments. In *Proceedings of COLING-ACL 98*.

Woszcyna, M., Aoki-Waibel, N., Buo, F. D., Coccaro, N., Horiguchi, K., Kemp, T., Lavie, A., McNair, A., Polzin, T., Rogina, I., Rosé, C. P., Schultz, T., Suhm, B., Tomita, M., and Waibel, A. (1994). JANUS 93: Towards spontaneous speech translation. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*.

Woszcyna, M., Coccaro, N., Eisele, A., Lavie, A., McNair, A., Polzin, T., Rogina, I., Rosé, C. P., Sloboda, T., Tomita, M., Tsutsumi, J., Waibel, N., Waibel, A., and Ward, W. (1993). Recent advances in JANUS: a speech translation system. In *Proceedings of the ARPA Human Languages Technology Workshop*.