Robust Interactive Dialogue Interpretation

by

Carolyn Penstein Rosé

B.S., University of California at Irvine, 1992

M.S., Carnegie Mellon University, 1994

Submitted to the Graduate Faculty of
the Language Technologies Insititue in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

Carnegie Mellon University

1997

To My Dad

Acknowledgements

In the five years that I have studied and worked here at Carnegie Mellon, I have benefited from my interactions with many fine researchers and students both here and at the University of Pittsburgh. And though these five years have in some ways been the most difficult of my life, I am thankful for having had the opportunity to follow a path that has grown out of my childhood dreams. Having always been a nerdy, unpopular child, at age thirteen I dreamed of having a computer I could talk to. In 1984 my parents helped me buy my first computer, a Commodore 64. Some of my first programs were ones that I could have simple conversations with. And so began my interest in Computational Linguistics. As an undergrad at the University of California at Irvine, my interests and career goals matured as I studied Computer Science and worked with great researchers such a Richard Granger, from whom I developed a broad interest in Artificial Intelligence, and Alfred Bork, from whom I developed an interest in Educational Technology. By the time I graduated I decided that my life long career goal would be to develop the technology to make natural language understanding practical for intelligent tutoring systems. My hope is that this dissertation project serves as one milestone along the path towards reaching this goal.

I especially appreciate my committee for giving me the freedom to pursue my interests and goals although they didn't necessarily correspond with their own interests at every point. My conscientious advisor, Lori Levin, has been an inspiration to me since early in my time here at CMU. By now she deserves a medal of valor for slogging through the largest number of drafts of this dissertation and earlier documents. I have always appreciated her gentle and supportive way of keeping me on track. Barbara Di Eugenio served almost as a second advisor at times and worked closely with me in the development of the Enthusiast discourse processor used in the interaction stage of this project. It is often said of Jaime Carbonell that one meeting with him is worth ten with anyone else. Though my interactions with Jaime were few, I can honestly say they were worth the wait. Alon Lavie joined my committee late, but my debt to him is most definitely the largest of all of my committee. As I look back on my experiences as a student here, some of my best memories are of projects we worked on together and ideas we have developed together. There is no one I consider more of a partner in this research than Alon. My two outside committee members, Johanna Moore and Sandra Carberry, provided me with much needed encouragement and criticisms at strategic moments along the way. I especially appreciated Johanna's frank manner, so that I never had to guess what she thought of my work, and Sandra's way of always finding something good to point out just when I needed a little extra encouragement.

The list of people to whom I owe a debt of gratitude is too large to enumerate here. Some of those who deserve special note are Ronnie Smith, Lynn Lambert, Branimir Boguraev, David Traum, Ted Pedersen, Christie Watson, Karen Ward, Philip Resnik, Judith Klavans, Susan Haller, Pammela Jordan, Sue Holm, Kathy Baker, Ariel Cohen, Steve Beale, Annette Baumgaertner, Yan Qu, Donna Gates, Julie (Yehudit) Van Dyke, Sondra Ahlen, Jennifer Potter, Larry and Rachel Mathers, Abe Sandler, Joel Barson, and Allen Schwartz. My best friends, Shiela Woodard, Guy Ziv, Sunny Loh, and Elizabeth Wall, provided me with constant support. I couldn't have done it without you guys!

My dearest family and wonderful in-laws have always been right by my side cheering me on. This dissertation is dedicated to my dad, Richard Penstein, who has made the biggest contribution to my education, bigger than any other single person in the world. All of the skills which have proved to be the most valuable to me in my work here at CMU I learned by watching my dad. Dad never waited for someone else to teach him what he needed to know, he found a book and taught himself. As a child, whenever I asked him a question, instead of giving me an answer, he handed me three books. Dad never took the easy way out, he always did what he believed was right. When the most direct route towards his goals was not available to him, he found a creative alternative path. He taught me how to think for myself and how to believe in my ideas.

I'm sure my husband Eric must be the greatest, most supportive husband there is. I hate to think of what these five years would have been without him to share this experience with. Whenever I need him, he never thinks twice but to drop whatever he is doing to be there for me, as though I were the most important person in the world. His childlike humor keeps my spirits high. His gentle manner provides me with a safe place to hide when the world seems to be crumbling around my ears. Most importantly, he serves as a constant reminder that my career is not the basis of my worth as a person, nor is my career my highest priority.

The most important lesson I have learned throughout all of my studies is that my own abilities and resources can only take me so far. In all my life I owe the biggest debt of gratitude to God, who's leading in my life provides security and hope, and who's strength makes up for my weaknesses.

Table of Contents

List of Tables

List of Figures

Robust Interactive Dialogue Interpretation

Carolyn Penstein Rosé, Ph.D.

Carnegie Mellon University, 1997

The first essential task of a natural language interface is to map the user's utterance onto some meaning representation which can then be used for further processing. The three biggest challenges which continue to stand in the way of accomplishing even this basic task are extragrammaticality, ambiguity, and recognition errors. In this dissertation I address the issue of how to handle the problem of extragrammaticality efficiently, where extragrammaticality is defined as any deviation of an input string from the coverage of a given system's parsing grammar.

A useful analogy can be made between human/computer interaction through a natural language interface and language interaction between speakers of different languages with a small shared language base. Humans who share a very small language base are able to communicate when the need arises by simplifying their speech patterns and negotiating until they manage to transmit their ideas to one another (Hatch, 1983). As the speaker is speaking, the listener "throws his net" in order to catch those fragments of speech which are comprehensible to him which he then attempts to fit together semantically. His subsequent negotiation with the speaker builds upon this partial understanding.

The approach presented here is based on this same model. The ROSE[1] approach, RObustness with Structural Evolution, repairs extragrammatical input in two stages. The first stage, Repair Hypothesis Formation, is responsible for assembling a set of hypotheses about the meaning of the ungrammatical utterance. This stage is itself divided into two steps, Partial Parsing and Combination. The Partial Parsing step is similar to the concept of the listener "casting his net" for comprehensible fragments of speech. Lavie's GLR* parser (Lavie, 1995; Lavie and Tomita, 1993) is used to obtain an analysis of islands of the speaker's sentence in cases where it is not possible to obtain an analysis for the entire sentence. In the Combination step, the fragments from the partial parse are assembled into a set of alternative meaning representation hypotheses. A genetic programming approach is used to search for different ways to combine the fragments in order to avoid requiring any

[1] ROSE is pronounced Rosé, like the wine.

hand-crafted repair rules. In ROSE's second stage, Interaction with the user, the system generates a set of queries, negotiating with the speaker in order to narrow down to a single best meaning representation hypothesis.

The primary objective of the ROSE approach is to handle the problem of extragrammaticality in an effective and efficient way. The most straightforward way to evaluate different approaches to handling extragrammaticality is by comparing them based on improvement in terms of percentage of sentences handled correctly or improvement of overall accuracy on a particular corpus. However, it is misleading to compare instantiations of different approaches this way since in theory many of these approaches have the potential for yielding the same amount of improvement given sufficient resources in terms of space (both static and dynamic), time (both development time and run time), and interactional effort. The real question is which approach can use these resources most economically.

I argue that the ROSE approach of separating the Partial Parsing and Combination steps is more efficient than placing the full burden of robustness on a single parsing algorithm. An analogous trade-off in human/human communication would be the "casting and combining" model versus one in which the listener tries to construct a complete syntactic analysis for a sentence outside of his language competence. Though humans are known to make a mental note of grammatical features that they are not able to process correctly, most of them are regarded mainly as "noise" (Hatch, 1983).

Another goal of this work is to demonstrate that it is more efficient to separate Repair Hypothesis Formation from User Interaction rather than interleaving them. In other words, a set of alternative ways of fitting the whole set of fragments from the partial parse (Global Repair Hypotheses) is constructed before any queries are generated, rather than generating a query to verify each repair step (Local Repair Hypotheses). Besides being more efficient, this approach is arguably more effective. When humans collaborate for the purpose of understanding, they direct their questions towards information which is necessary for accomplishing their task (Clark and Schaefer, 1989; Clark and Wilkes-Gibbs, 1986). When questions are directed at clarifying the speaker's meaning, rather than furthering the shared task, speakers become agitated (Garfinkel, 1964). By delaying the interaction until hypotheses about the speaker's whole meaning are formed, it is possible to focus the interaction on the task level rather than on the language level.

Therefore, it will be shown that the ROSE approach robustly extracts the meaning from the user's extragrammatical utterance efficiently and without placing an undue burden on the user in terms of interactional effort. Finally, because the ROSE approach does not

rely on any hand crafted repair rules or additional knowledge sources, it is a completely general and portable solution.

Part I

Introduction

Chapter 1

Introduction

Although giving users the ability to interact with computers through spontaneous natural language would be ideal, it poses challenges which continue to fall outside of the reach of state-of-the-art technology. Among these challenges is the problem of extragrammaticality, where extragrammaticality is defined as any deviation of an input string from the coverage of a given system's parsing grammar. It is impossible to anticipate all of the ways in which users will express ideas meant to be covered by a language interface. Thus, extragrammaticality is unavoidable in today's systems and must be dealt with in an effective and efficient way. In this dissertation I address the issue of how to handle the problem of extragrammaticality efficiently in a large scale system.

A useful analogy can be made between human/computer interaction through a natural language interface and language interaction between speakers of different languages with a small shared language base. If two conversational participants share only a limited language base, it is likely that the set of forms accessible to one speaker is not the same set of forms accessible to the other speaker. Furthermore, it is not mutually known from the start which forms are shared and which are not. Therefore, the listener must be able to cope with the situation where only part of the speech produced by the speaker is comprehensible. Since no language interface developed to date can process all of even a single human language, the situation in human/computer interaction is similar. A naive user, in general, is not aware of the exact capabilities of the natural language interface through which he is interacting with the computer. So a robust language interface must be able to cope with the situation where only part of the input utterance is comprehensible.

Humans who share a very small language base with one another are able to communicate when the need arises by simplifying their speech patterns and negotiating until they manage to transmit their ideas to one another (Hatch, 1983). As the speaker is speaking, the listener "throws his net" in order to catch those fragments of speech that are comprehensible to him, which he then attempts to fit together semantically. His subsequent

negotiation with the speaker builds upon this partial understanding. Through this negotiation process, the conversational participants learn to communicate with one another. They adapt their speech patterns as they discover through experimentation which are most effective. Since humans possess the innate ability to participate in this collaborative process in order to achieve understanding, it is worthwhile to explore approaches to human/computer interaction that exploit this ability. In previous language interface research, humans have exhibited the ability to adapt similarly to the interface's limitations as they observe how the system responds to their utterances (Slator, Anderson, and Conley, 1986; Lehman, 1989).

The approach presented in this dissertation is based on this collaborative model. The ROSE[1] approach, RObustness with Structural Evolution, repairs extragrammatical input in two stages. The first stage, Repair Hypothesis Formation, is responsible for assembling a set of hypotheses about the meaning of the ungrammatical utterance. This stage is itself divided into two steps, Partial Parsing and Combination. The Partial Parsing step is similar to the concept of the listener "casting his net" for comprehensible fragments of speech. Lavie's GLR* parser (Lavie, 1995; Lavie and Tomita, 1993) is used to obtain an analysis of islands of the speaker's sentence in cases where it is not possible to obtain an analysis for the entire sentence. In the Combination step, the fragments from the partial parse are assembled into a set of alternative meaning representation hypotheses. A genetic programming approach is used to search for different ways to combine the fragments in order to avoid requiring any hand-crafted repair rules. In ROSE's second stage, Interaction with the User, the system generates a set of queries, negotiating with the speaker in order to narrow down to a single best meaning representation hypothesis.

1.1 Characterization of the Problem

The first essential task of a natural language interface is to map the user's utterance onto some meaning representation that can then be used for further processing. The three biggest challenges that continue to stand in the way of accomplishing even this basic task are extragrammaticality, ambiguity, and recognition errors[2].

- Extragrammaticality is anything that places a particular input string outside of the coverage of a particular system's parsing grammar.

[1] ROSE is pronounced Rosé, like the wine.

[2] Recognition errors include speech recognition errors in a speech system or words incorrectly identified because of spelling errors in a text-based system.

- Ambiguity is measured by the extent to which more than one analysis for a particular input string can be derived by applying a system's particular parsing grammar to it with the system's particular sentence-level interpretation algorithm.

- Recognition errors occur when the string of words that are presented to the sentence level interpretation algorithm deviate in any way from the actual words that were intended by the user, either because of speech recognition errors or because of typing mistakes on the user's part, depending upon the input method employed.

These three issues are tightly interrelated. For example, recognition errors are often the source of extragrammaticality, although they are by no means the only source. In turn, extragrammaticality has the effect of making the meaning of the user's utterance less certain and thus more ambiguous because of the additional flexibility required in order to process the sentence. Since speech recognizers often make use of local context information to constrain their expectations for the current word, extragrammaticality can also be the source of some recognition errors to the extent that extragrammaticality either violates these local context constraints or establishes false ones. Furthermore, in order to avoid some measure of ambiguity, grammar rules for uncommon constructions may be left out of parsing grammars, thus making these uncommon constructions extragrammatical as a result. In this dissertation I focus primarily on the issue of extragrammaticality. I do not attempt to address the issue of reducing recognition errors. And I only address ambiguity that is a side effect of my approach to handling extragrammaticality.

The primary objective of the ROSE approach is to handle the problem of extragrammaticality in an effective and efficient way. The most straightforward way to evaluate different approaches to handling extragrammaticality is by comparing them based on improvement in terms of percentage of sentences handled correctly or improvement of overall accuracy on a particular corpus. However, it is misleading to compare instantiations of different approaches this way since in theory many of these approaches have the potential for yielding the same amount of improvement given sufficient resources in terms of space (both static and dynamic), time (both development time and run time), and interactional effort. The real question is which approach can use these resources most economically.

The work described in this dissertation was conducted in the context of the Enthusiast system (Suhm et al., 1994; Levin et al., 1995), part of the large-scale JANUS multilingual speech-to-speech machine translation project (Lavie et al., 1996). This machine-translation system currently deals with the scheduling domain. The dialogues which provide input to the system are spontaneous conversations between two individuals who are attempting to schedule a meeting together. As in any interactive system, interactional effort

and run-time are the most important considerations in evaluating the appropriateness of an approach to robust understanding. An approach that either takes an inordinate amount of processing time or that overburdens the user with tedious questions just for the sake of understanding is not acceptable, since it distracts the user from the task. Development time is a secondary, but nevertheless important, consideration as well, since it is common for system specifications to change frequently. Thus, a domain independent, in other words totally portable, approach like ROSE is the best in this regard.

1.2 Overview of The ROSE Approach

As mentioned above, the ROSE approach is composed of two stages: Hypothesis Formation and Interaction with the User. The Hypothesis formation stage is itself divided into two steps: Partial Parsing and Combination. In this dissertation, I argue that the ROSE approach of separating the Partial Parsing and Combination steps is more efficient than placing the full burden of robustness on a single parsing algorithm. The two step approach is more efficient both because the two separate steps are more restricted than the single parsing algorithm would have to be and because having two separate steps makes it possible to bypass the second step when it is not needed. Lavie's GLR* parser (Lavie, 1995) is used for partial parsing in ROSE. This parser is capable of skipping over any portion of an input utterance that cannot be incorporated into a grammatical analysis in order to recover the analysis for the largest grammatical subset of the utterance. The parse for the largest segment plus analyses for the skipped portions together form the set of chunks that are input to the Combination step. Thus, ROSE distributes the flexibility required over two steps rather than placing the full burden of robustness on a single parsing algorithm.

In contrast, the Minimum Distance Parsing (MDP) approach which is described separately by Hipp, Lehman, and Ramshaw in (Hipp, 1992; Lehman, 1989; Ramshaw, 1994) is the canonical example of an approach that places the full burden of robustness on a single parsing algorithm. It is called Minimum Distance Parsing because it searches for the minimum number of insertions, deletions, and sometimes substitutions and transpositions that need to be performed in order to transform the extragrammatical input into something within the coverage of the parsing grammar. An analogous trade-off in human understanding would be the "casting and combining" model versus one in which the listener tries to construct a complete syntactic analysis for a sentence outside of his language competence. Though humans are known to make a mental note of grammatical features that they are not able to process correctly, most of them are regarded mainly as "noise" (Hatch, 1983).

Because the ROSE approach does not rely on any hand crafted repair rules or additional knowledge sources dedicated to repair, unlike other "casting and combining" approaches such as (Ehrlich and Hanrieder, 1996; Danieli and Gerbino, 1995), it is a completely general and portable solution. These other "casting and combining" approaches rely on hand-coded repair rules to guide the search for a correct interpretation of the user's extragrammatical sentence. Though these approaches are effective and efficient, they make it necessary to solve the repair problem again and again each time a system in a new domain is developed. With the ROSE approach, on the other hand, it can be applied to new domains and new languages by simply automatically training its statistical knowledge sources used to bias the search and its fitness function which measures the relative goodness of repair hypotheses.

Another goal of this work is to demonstrate that it is more efficient to separate Repair Hypothesis Formation from Interaction with the User, rather than interleaving them. In other words, a set of alternative ways of fitting the whole set of fragments from the partial parse (Global Repair Hypotheses) is constructed before any queries are generated rather than generating a query to verify each repair step (Local Repair Hypotheses). This will be demonstrated by contrasting the two stage ROSE approach with an approach to Incremental Repair with Local repair Hypotheses (IRLH) as described by Rosé and Waibel (1994). The primary problem with the IRLH approach is that each of the repair hypotheses are generated locally, with each decision building upon the result of the last successful hypothesis. So through trial and error, the repair module is forced to ask a large number of very tedious questions. Besides being more efficient than IRLH, the ROSE approach is arguably more effective. When humans collaborate for the purpose of understanding, they direct their questions towards information that is necessary for accomplishing their task (Clark and Schaefer, 1989; Clark and Wilkes-Gibbs, 1986). When questions are directed at clarifying the particulars about the speaker's meaning, rather than furthering the shared task, speakers become agitated (Garfinkel, 1964). By delaying the interaction until hypotheses about the speaker's whole meaning are formed, it is possible to focus the interaction on the task level rather than on the language level.

These claims about ROSE will be demonstrated in a rigorous evaluation.

- ROSE's "casting and combining" approach will be demonstrated to be orders of magnitude faster than the MDP approach by comparing run times of the two alternative approaches over a large corpus of sentences and holding all other factors[3] constant.

[3]such as parsing grammar complexity and vocabulary size.

- ROSE's two stage approach to interaction will be demonstrated to be more efficient than the interleaved approach by comparing average number of questions required to arrive at an acceptable result with ROSE and with IRLH.

- By making use of information provided by a plan-based discourse processor about alternative meaning representation hypotheses, ROSE will be demonstrated to be able to formulate 20% of its queries to the user in terms of the task rather than in terms of the user's literal meaning.

1.2.1 Hypothesis Formation

The first step of the Hypothesis Formation process is to obtain an analysis for "islands" of the speaker's utterance if it is not possible to obtain an analysis for the whole utterance. This is accomplished with Lavie's GLR* parser (Lavie, 1995; Lavie and Tomita, 1993), described in more detail in Appendix A. See Figure 1.1 for an example parse. Here the GLR* parser[4] attempts to handle the sentence "That wipes out my mornings." The expression "wipes out" does not match anything in the parsing grammar. The grammar also does not allow time expressions to be modified by possessive pronouns. So "my mornings" also does not parse. Although the grammar recognizes "out" as a way of expressing a rejection, as in "Tuesdays are out," it does not allow the time being rejected to follow the "out". Note that, although the parser was not able to obtain a complete parse for this sentence, it was able to extract four chunks.

The chunks are feature structures in which the parser encodes the meaning of portions of the user's sentence. This frame based meaning representation is called an interlingua because it is language independent. It is defined in a document called an interlingua specification (see Appendix B) which is the primary symbolic knowledge source used by the Combination step in the Hypothesis Formation stage. Each frame encodes a concept in the domain. The set of frames in the meaning representation are arranged into subsets which are assigned a particular type. Each frame is associated with a set of slots. The slots represent relationships between feature structures. Each slot is associated with a type, which determines the set of possible frames that can fill the slot.

The four chunks extracted by the parser each encode a different part of the meaning of the sentence "That wipes out my mornings." The first chunk represents the meaning of "that". The second one represents the meaning of "out". Since "out" is generally a way of rejecting a meeting time in this domain, the associated feature structure represents the

[4] A restricted version of GLR* is used here. In the original GLR* algorithm, the parser can skip over any portion of the input sentence in order to search for the largest subset which it can parse. In the restricted version, it can only construct analyses for contiguous portions.

concept of a response which is a rejection. Since "wipes" does not match anything in the grammar, this token is left without any representation among the fragments returned by the parser. The last two chunks represent the meaning of "my" and "mornings" respectively.

The second step of the Hypothesis Formation stage is to put these islands of meaning together. During this step, ROSE constructs a meaning representation for the whole utterance from the partial analysis using a genetic programming approach. This second step makes it possible to make a large portion of the remainder of the types of repairs that can be made with the minimum distance parsing approach, but that can not be made with the GLR* parser alone. What distinguishes the ROSE approach from other similar "casting and combining" approaches is that it does not rely on any hand crafted repair rules. The genetic programming paradigm provides a completely general search engine for experimenting with different ways of combining the chunks from the parser, which is necessary in the absence of specific repair rules. It uses the knowledge represented in the interlingua representation specification to distinguish legal ways of combining chunks from illegal ones.

Recovery from parser failure is a natural application of genetic programming (Koza, 1992; Koza, 1994). One can easily conceptualize the process of constructing a global meaning representation hypothesis as the execution of a computer program that assembles the set of chunks returned from the parser. This program would specify the operations required for building larger chunks out of smaller chunks and then even larger ones from those. Because the programs generated by the genetic search are hierarchical, they naturally represent the compositional nature of the repair process. See Figure 1.2 for an example repair hypothesis. `MY-COMB` is a simple function taking three parameters, namely, a parent chunk, a child chunk, and a slot. It attempts to insert the child chunk into some slot in the parent chunk. It selects a slot, if a suitable one can be found, and then instantiates the third parameter to this slot. In this case, the `WHEN` slot is selected. So the feature structure corresponding to "mornings" is inserted into the `WHEN` slot in the feature structure corresponding to "out". The result is a feature structure indicating that "Mornings are out." Though this is not an exact representation of the speaker's meaning, it is the best that can be done with the available feature structures[5]. Notice that since the expression "wipes out" is foreign to the parsing grammar, and no similar expression is associated with the same meaning in it, the MDP approach would not be able to do better than this because it can

[5]Note that it is lucky that part of the expression "wipes out" matches a rule in the grammar that just happens to have a similar meaning, since "out" can be used to reject a suggestion as in "Tuesday is out." If the expression had been something like "out of sight", which is positive, both ROSE and MDP would construct the opposite meaning from the intended meaning. Problems like this can only be dealt with through interaction with the user to confirm that repaired meanings reflect the speaker's true intention.

only insert and delete in an attempt to fit the current sentence to the rules in its parsing grammar. Additionally, since the time expression follows "out" rather than preceding it as the grammar expects, only MDP with transpositions in addition to insertions and deletions would be able to arrive at the same result. Note that the feature structures corresponding to "my" and "that" are not included in this hypothesis. The job of the Combination Mechanism is both to determine which fragments to include as well as how to combine the selected ones.

In the genetic programming approach, a population of programs are evolved which specify how to build complete meaning representations from the chunks returned from the parser. A complete meaning representation is one that is meant to represent the meaning of the speaker's whole utterance, rather than just part. Partial solutions are evolved through the genetic search specifying how to build parts of the full meaning representation. Because in the same population there can be programs that specify how to build different parts of the meaning representation, different parts of the full solution are evolved in parallel, making it possible to evolve a complete solution quickly.

Since the Combination Mechanism produces a set of alternative global meaning representation hypotheses, the result of the Combination Mechanism step is similar to an ambiguous parse. See Figure 1.3 for two alternative repair hypotheses from the Combination Mechanism for the example in Figure 1.1. The result of each of the hypotheses is an alternative representation for the sentence. The first hypothesis corresponds to the interpretation, "Mornings and that are out." This hypothesis is erroneous because it includes the "that" chunk, which in this case should be left out[6].

In the second hypothesis, the Combination Mechanism attempted to insert the rejection chunk into the time expression chunk, the opposite of the ideal order. No slot could be found in the time expression chunk in which to insert the rejection expression chunk. In this case, the slot remains uninstantiated and the largest chunk, in this case the time expression chunk, is returned. This hypotheses produces a feature structure that is indeed a portion of the correct structure, though not the complete structure.

[6]Though this would be a perfectly acceptable paraphrase into Pittsburghese, most speakers of Standard English would find this unacceptable.

Sentence: *That wipes out my mornings.*

Partial Analyses:

Chunk1: that

((ROOT THAT)
(TYPE PRONOUN)
(FRAME *THAT))

Chunk2: out

((TYPE NEGATIVE)
(DEGREE NORMAL)
(FRAME *RESPOND))

Chunk3: my

((ROOT I)
(TYPE PERSON-POSS)
(FRAME *I))

Chunk4: mornings

((TIME-OF-DAY MORNING)
(NUMBER PLURAL)
(FRAME *SIMPLE-TIME)
(SIMPLE-UNIT-NAME TOD))

Figure 1.1: Parse Example

Ideal Repair Hypothesis:

```
(MY-COMB                                                    ;insert arg2 into arg1 in slot
  ((FRAME *RESPOND) (DEGREE NORMAL) (TYPE NEGATIVE))        ;arg1
  ((TIME-OF-DAY MORNING) (NUMBER PLURAL)                    ;arg2
   (FRAME *SIMPLE-TIME) (SIMPLE-UNIT-NAME TOD))
  WHEN)                                                     ; slot
```

Ideal Structure:

```
((FRAME *RESPOND)
 (DEGREE NORMAL)
 (TYPE NEGATIVE)
 (WHEN ((FRAME *SIMPLE-TIME)
        (TIME-OF-DAY MORNING)
        (NUMBER PLURAL)
        (SIMPLE-UNIT-NAME TOD))))
```

Gloss: Mornings are out.

Figure 1.2: Combination Example

Some Alternative Repair Hypotheses:

Hypothesis1:

```
(MY-COMB
   (MY-COMB
      ((FRAME *RESPOND))
      ((FRAME *SIMPLE-TIME) (TIME-OF-DAY MORNING)
       (NUMBER PLURAL) (SIMPLE-UNIT-NAME TOD))
      WHEN)
   ((FRAME *THAT) (ROOT THAT) (TYPE PRONOUN))
   WHEN)
```

Result1: Mornings and that are out.

```
((FRAME *RESPOND)
 (DEGREE NORMAL)
 (TYPE NEGATIVE)
 (WHEN (*MULTIPLE* ((FRAME *SIMPLE-TIME) (TIME-OF-DAY MORNING)
                    (NUMBER PLURAL) (SIMPLE-UNIT-NAME TOD))
                   ((FRAME *THAT) (ROOT THAT) (TYPE PRONOUN)))))
```

Hypothesis2:

```
(MY-COMB
  ((TIME-OF-DAY MORNING) (NUMBER PLURAL)
   (FRAME *SIMPLE-TIME) (SIMPLE-UNIT-NAME TOD))
  ((FRAME *RESPOND) (DEGREE NORMAL) (TYPE NEGATIVE))
  ??)
```

Result2: Mornings.

```
((FRAME *SIMPLE-TIME) (TIME-OF-DAY MORNING)
 (NUMBER PLURAL) (SIMPLE-UNIT-NAME TOD))
```

Figure 1.3: **Repair Hypotheses**

1.2.2 Interaction With the User

The purpose of the second stage of the ROSE approach is to strategically use interaction with the user in order to narrow down from the set of hypotheses produced in the Hypothesis Formation stage to a single best meaning representation when there is a good hypothesis. Another purpose of interaction is to determine whether a rephrase is more likely to yield an improvement in translation quality over any of the hypotheses constructed in the Hypothesis Formation stage. In these cases, the hypotheses generated by the Hypothesis Formation stage are abandoned and the user is queried for a rephrase.

In any case, first a Comparator extracts the set of features that distinguish the alternative hypotheses. Features that distinguish the three hypotheses discussed earlier include one that indicates whether or not to include the feature structure for "that" and whether or not to include the feature structure for "out". The set of features that the Comparator extracts are then passed on to the Interaction Mechanism, which sorts the features according to a set of criteria including coherence with previous questions asked and potential search space reduction. This sorted list is then used to generate a series of queries to the user. Since the best repair hypothesis is not always the first one, and since some hypotheses are decidedly incorrect, this interaction stage is important for ensuring a high level of accuracy. However, as we shall see the results for the performance of the Hypothesis Formation stage alone indicate that it is possible to increase robustness with the ROSE approach even without interaction.

Since the ROSE Hypothesis Formation stage produces hypotheses that cover the whole utterance, it is possible to compute the differences between the alternatives in terms of how they affect the discourse context. This makes it possible to focus the interaction on either the sentence level meaning or on the task level. Two instantiations of ROSE are evaluated and compared in Chapters 9 and 10, the differences being only relevant to the Interaction with the User stage. Assume the feature selected distinguishes between hypotheses producing structures with "out" and ones without. In the first instantiation, which is the generic version of ROSE, the questions to the user are formulated in terms of the sentence level meaning of the user's utterance. For example, "Is something like MORNINGS ARE OUT part of what you meant?" In the second instantiation, Discourse ROSE, ROSE's Interaction with the User stage is augmented to make use of the discourse context. Once the features are evaluated with respect to their impact on the discourse state, questions can be reformulated in terms of the task. In this case, a rephrase of the preceding question would be "Are you indicating that the mornings are not a good time to meet?"

1.3 Efficiency

If a natural language interface is going to be used in any large scale interactive system, it must be able to run efficiently and to scale up well. The minimum distance parsing approach (Hipp, 1992; Lehman, 1989; Ramshaw, 1994) is a domain independent and portable approach. However, it has not been demonstrated to scale up well. Moreover, the intractability of MDP is demonstrated in this dissertation by evaluating it in the context of the Enthusiast translation system. ROSE is demonstrated to remain tractable in this large scale system. ROSE achieves its ability to run efficiently and to scale up by distributing its flexibility over two more constrained steps rather than introducing the full amount of flexibility at parse time.

As mentioned earlier, in the Partial Parsing stage, Lavie's GLR* parser (Lavie, 1995) is used. This parser is capable of skipping over any portion of an input utterance that cannot be incorporated into a grammatical analysis, and recovering the analysis for the largest grammatical subset of the utterance. Because it is more constrained than the MDP approach, it is correspondingly more efficient. Naturally, a parser that can only skip over words will be faster than a similar parser that can also insert and substitute words, although there will be some repairs that cannot be made with skipping alone which could be made with the MDP approach.

Furthermore, since islands of analysis are built in the parsing step, the additional flexibility that is introduced in the combination step is channeled to the part of the analysis that the system does not have enough knowledge to handle straightforwardly. The partial analyses that the system does have the knowledge to construct normally remain intact. This is unlike the too powerful minimum distance parsing approach in which the full amount of flexibility is blindly applied to every part of the analysis, whether the sentence is grammatical or not. Therefore, we expect this two stage process to be more efficient since it first determines what it knows and then attempts to fill in the rest, making use of what it knows from the first stage to constrain the search in the second stage. Experiments described in Chapter 10 confirm that the two stage approach is orders of magnitude more efficient.

1.4 User Interaction and Collaborative Effort

User interaction is a key facet of the ROSE approach. Because no current natural language understanding system can handle all human language, the user must be aware of the limitations of the interface in order to use it effectively. As described earlier, users have been demonstrated to adapt to the limitations of natural language interfaces as they observe

how the system responds to their utterances. However, this is only feasible if the system can both extract at least a partial representation from most utterances and can express to the user what it was able to understand. If for each utterance that is not within the coverage of the interface, the user is simply asked to repeat the utterance, it will not likely be clear what aspects of the utterance are outside of the system's coverage, and the user might be confused. If, on the other hand, the system is able to indicate its best guess at a meaning for an extragrammatical utterance, the user would have the opportunity by experience what types of utterances can successfully be communicated and what types should be avoided. Though the coverage of such a system is not complete, it allows users to express their ideas naturally, extracting what it can. The user can then learn how to interact with the system implicitly while using it rather than explicitly being taught.

These claims are consistent with findings about human/human interaction discussed by Clark and Wilkes-Gibbs (1986) and Clark and Schaefer (1989). In this work, a model of human communication is developed in which one speaker presents an utterance, and then the listener responds, indicating his understanding of the speaker's utterance. If the speaker determines, based on the listener's response, that correct understanding was not achieved, he can try again to convey the portion of the utterance that was not understood. In this way, the speaker and hearer collaborate in order to insure that the speaker's utterance is communicated successfully. Furthermore, Schober and Clark (1989) demonstrate in that overhearers who are prevented from participating in this collaborative process do not achieve the same level of understanding as those who are able to participate directly. Therefore, this collaborative process is an essential aspect of successful human/human communication.

The ROSE approach is contrasted with the IRLH approach discussed in Rosé and Waibel (1994) in which interaction is interleaved with hypothesis formation. In IRLH, the repair module searches for the complete meaning representation structure by generating and testing hypothesised local repair actions. Local repair actions include steps such as first determining which of the available chunks contain the top level frame of the target meaning representation structure and then testing each other chunk separately to determine what, if any, slot it could fill the top level frame. If a hypothesis is confirmed to be correct through interaction with the speaker, the repair module then makes the specified repair. Otherwise it generates and tests a new hypothesis until it finds one that is correct. The primary problem with this approach is that each of these hypotheses are generated locally, with each decision building upon the result of the last successful hypothesis. Thus, the repair module is forced to ask a large number of tedious questions. In contrast, the ROSE system first evolves a set

of hypotheses that generate meaning representations that cover the whole utterance. Since questions are generated to eliminate as many alternative complete hypotheses as possible, not every repair action is confirmed. In this way, the number of questions that the user must answer is reduced.

1.5 Minimization of Development Time and Portability

Any approach to recovery from parser failure that is not domain independent is impractical in this age where funding is scarce and changing system specifications are the norm, even on the level of which domain to work in. A domain independent approach makes it possible to do only once most of the work necessary in order to introduce the recovery process. Then, if the system specifications change, the work that has already been done will not need to be duplicated. The repair process I propose in this dissertation makes use of knowledge sources that are already part of a natural language understanding system, such as a parsing grammar and a meaning representation specification. Because ROSE makes use of these resources, but is otherwise independent of them, it can be used when these are still under development. They are not required to be complete. This makes it possible to introduce the capability of repair early in the system development process, though it should be kept in mind that the performance of the ROSE approach to repair is limited by the ability of the parser to produce meaningful chunks. And because it will only build structures which are consistent with the meaning representation specification, it will never be able to successfully repair sentences who's meaning cannot be represented in the meaning representation language.

The machine learning component is also essential for making the approach highly portable. It consists of a set of networks that compute connection strength between slots and fillers. These are used to bias the genetic search in the Combination Mechanism. This is the only knowledge needed by ROSE's Hypothesis Formation stage that is not part of the system to begin with, and it can be acquired by ROSE automatically. Furthermore, these networks can "learn by doing" and therefore continue learning while they are being used. An approach to repair that can "learn by doing" is ideal since it makes it possible for ROSE to improve its performance over time. Some types of "learning by doing" are not practical for a large system, however. In particular, learning approaches that add new knowledge in terms of rules for a parsing grammar have been observed to become impractical, particularly where multiple users, each with their own idiosyncratic language patterns, are concerned (Lehman, 1989). As new rules are added, the time and space requirements of the system increase. If, on the other hand, new knowledge is added by changing weights, as with a

statistical or neural net approach, the time and space requirements of the system are exactly the same after learning as they were before learning. This makes it possible to "learn by doing" with multiple users without the system becoming bogged down. All of the knowledge sources that ROSE requires other than those that would be part of the system to begin with are acquired automatically. Moreover, the "weights" can be modified during use of the system without bogging down the system with additional time and space requirements.

1.6 Dissertation Overview

The following is an outline of the remainder of this dissertation.

- **Chapter 2:** *Robust Parsing: Background and Relevant Literature.* In this chapter several different alternative approaches to robust interpretation are reviewed and compared in order to illustrate the space of possible alternative solutions to the problem of extragrammaticality.

- **Chapter 3:** *Genetic Programming: Background and Relevant Literature.* In this chapter, the genetic programming paradigm is described at a basic level. This chapter can be skipped by the reader who already has a passing familiarity with genetic programming.

- **Chapter 4:** *Incremental Repair with Local Hypotheses.* Chapters 4 and 5 describe the two approaches that I contrast with the ROSE approach in my evaluation. Chapter 4 contains a detailed description of Rosé and Waibel's IRLH approach, in which repair and interaction are interleaved.

- **Chapter 5:** *Minimum Distance Parsing.* In this chapter a description of the MDP approach is given, as well as a demonstration of how MDP becomes intractable for systems of realistic scope.

- **Chatper 6:** *The ROSE Approach: Overview.* This chapter contains a detailed overview of the ROSE approach.

- **Chapter 7:** *The Combination Mechanism.* Where Chapter 6 provides a broad overview of the ROSE approach, chapters 7 and 8 each elaborate on one stage in the two stage ROSE process. Chapter 7 focuses on the Combination step inside the Hypothesis Formation Stage.

- **Chapter 8:** *Interaction with the User.* This chapter contains a detailed description of the Comparator and Interaction Mechanism, which together compose ROSE's second stage, Interaction with the User.

- **Chapter 9:** *Discourse ROSE: Using Discourse Information in Repair.* In this chapter, I describe Discourse ROSE, the version of ROSE's Interaction stage that uses discourse information. First, the discourse processor is discussed. Then the application of the discourse processor to the interaction task is discussed.

- **Chapter 10:** *Evaluation.* In this chapter three separate evaluations are presented. In the first evaluation, I demonstrate that ROSE generates global repair hypotheses orders of magnitude faster than the MDP approach. In the second evaluation I demonstrate the reduction in interactional effort required by the ROSE approach as opposed to the IRLH approach. In the final evaluation, I demonstrate the improvement in interaction quality which discourse information provides.

- **Chapter 11:** *Conclusions and Future Directions.* In this chapter, this dissertation is drawn to a close with a discussion of specific weaknesses of the ROSE approach as well as directions for future research.

- **Appendix A:** *GLR* Parser: Background Information.* This appendix contains some background information on the GLR* parser, described in more detail in (Lavie, 1995).

- **Appendix B:** *Interlingua Representation for Scheduling Dialogues.* This appendix defines the meaning representation language that specifies the range of legal meaning representations that can be built with ROSE. First the syntax of the specification is described. Then portions of the meaning representation for the scheduling domain are presented as examples.

Part II

Background Information and Literature Review

Chapter 2

Robust Parsing: Background and Relevant Literature

Robust natural language interfaces must be able to cope with input that only partially matches the knowledge they have encoded in their grammars and other knowledge sources. Here this problem is referred to as the problem of extragrammaticality. Attempts to solve this problem on the parsing level are referred to as robust parsing. The goal is to achieve a high level of robustness efficiently.

There are many different approaches to robust parsing, but which way is best? The most straightforward way to evaluate different approaches to handling extragrammaticality is by comparing them based on improvement in terms of percentage of sentences handled correctly or improvement of overall accuracy on a particular corpus. Several approaches to handling the problem of extragrammaticality have been evaluated in this manner, and results along these lines are presented in this chapter for the various approaches compared and contrasted in it. Typically, more flexible approaches achieve a higher level of robustness than less flexible ones, but at a high computational cost. The goal of the ROSE approach is to achieve a high level of robustness at a lower computational cost than competing approaches.

It is misleading to compare instantiations of different approaches only in terms of their relative performance over a corpus. In theory many of these approaches have the potential for yielding the same level of performance given sufficient resources in terms of space (both static and dynamic), time (both development time and run time), and interactional effort. Results for particular approaches have traditionally been presented in papers for specific instantiations of these parameters. The real question is which approach can use these resources most economically. But this can only be accomplished by comparing different approaches holding these factors constant, or by varying them consistently for each approach. It is not possible to compare all of the approaches discussed in this chapter along these lines since this information is not available. In Chapter 10, however, a comparison

of ROSE to two competing approaches will be presented, holding all secondary factors constant.

2.1 Variation Among Approaches to Extragrammaticality

The alternative approaches to dealing with the problem of extragrammaticality vary along a number of different dimensions which affect how they compare with one another in terms of the three resources discussed above.

- **Interaction:** For example, some approaches involve no interaction with the user, while some involve interaction in a separate stage from when the actual repairs are made, and still others interleave the repair and interaction. Obviously approaches with no interaction have no interactional effort requirement, whereas the other two types may require more or less than one another. The question is, what advantage does the interaction provide? And which approach to interaction contributes more for the associated cost?

- **Complexity/Distribution of Labor:** Some approaches attempt to perform all of the repairs within a single stage, i.e., during parse time, while others separate it into two or more stages, such as the ROSE approach. The two stage approach is perhaps more complicated to develop. But the question is: which approach will run faster for the same improvement in interpretation accuracy?

- **Machine Learning:** Different machine learning techniques may be involved, either at parse time or ahead of time. These might cause the time and space requirements of the approach to increase over time as performance improves. Does the associated improvement in performance make the corresponding increase in necessary resources worth the cost?

- **Discourse/Domain Knowledge:** Finally, varying degrees of discourse or domain knowledge may be brought to bear. This increases the development time as well as the run-time space and time requirements. The question is whether the improvement in performance is worth the extra cost.

On a more abstract level, approaches to robust parsing can be compared in terms of how flexible they are and how their flexibility is distributed. Flexibility here is taken to be the size of the class of extragrammaticalities that an approach can recover from. Flexibility can be implemented in a number of different ways. Connectionist approaches are flexible

because of their fuzzy type of computation. Some connectionist approaches might be better at recovering from certain types of extragrammaticalities than others because of the way they are trained or the type of architecture in which they are based. Flexibility can also be implemented in terms of edit operations performed by a parsing algorithm. The subset of edit operations allowed determines the class of extragrammaticalities that the parser is able to handle. One approach might have more flexibility than another approach in the sense that one might be able to recover from a larger class of extragrammaticalities. On the other hand, one approach might have the same degree of flexibility as another, but distribute it differently. A parser with a full set of edit operations might be able to recover from the same class of extragrammaticalities as a parser with a subset of the edit operations paired with a second algorithm that improves upon the result of the parsing step. Though the flexibility in the second step may not be implemented in terms of the same set of edit operations that make the parsing algorithm flexible, both the one step parsing approach and the two step approach may have the same amount of flexibility. This abstract notion of flexibility will play a significant role in the discussion contained in this chapter.

The great majority of research aimed at handling ill-formed input has focused on flexible symbolic parsing strategies. Recently, however, connectionist and statistical approaches have also shown promise. I will review several approaches to robust parsing in this chapter, keeping the above stated trade-offs in view.

2.2 Flexible Symbolic Parsers

Flexible symbolic parsers vary in terms of how much and what type of flexibility they incorporate. In the extreme case, in a Minimum Distance Parser (MDP), extragrammatical sentences are mapped onto strings within the coverage of the grammar through a series of insertions and deletions, and sometimes substitutions and/or transpositions [1].

- **Insertions:** The parser inserts or simulates inserting terminals or non-terminals into the input sentence.
- **Deletions:** The parser deletes or simulates deleting a portion of the input sentence.
- **Substitutions:** The parser substitutes or simulates substituting a portion of the input sentence for some other sequence of terminal or non-terminal symbols.

[1] MDP is described here as applying edit operations to the input sequence. Note that it can equivalently be thought of as applying edit operations on grammar rules, and is often implemented that way.

- **Transpositions:** The parser switches or simulates switching one or more terminal symbols in the input sentence with one or more other terminal symbols in the input sentence.

Other more restrictive parsers allow some subset of these edit operations. In this dissertation I argue that the MDP approach is intractable in a system of realistic scale and that a more restrictive parsing algorithm is necessary. These more restrictive parsers trade off coverage for speed. The idea is to introduce enough edit operations to gain an acceptable level of flexibility at an acceptable computational expense. The goal of ROSE's two stage approach is to increase the coverage possible at a reasonable computational cost by introducing a post-processing repair stage, which constructs a complete meaning representation out of the fragments of a partial parse.

2.2.1 Minimum Distance Parsers

In this section I will compare several approaches to Minimum Distance Parsing. Lehman's approach described in (Lehman, 1989; Lehman and Carbonell, 1989) as well as Smith and Hipp's approach discussed in (Hipp, 1992; Smith, 1992) rely on interaction with the user to verify that ill-formed input is handled correctly. Lehman also makes use of language patterns learned from previous interactions with the user to improve future performance. The Smith and Hipp approach as well as the Ramshaw approach described in (Ramshaw, 1994) both rely on contextual information in handling ill-formed input.

2.2.1.1 Lehman

In (Lehman, 1989; Lehman and Carbonell, 1989), Lehman describes an approach to handling naturally occurring text in a robust way. Her approach is based on the observation that although the types of linguistic patterns produced by different speakers vary widely, each speaker is fairly consistent in the types of expressions he/she uses over time. Although human speech will inevitably deviate from any core grammar written by hand, a single speaker will speak in such a way that his/her speech patterns will deviate in consistent ways. These observations were made in the context of Wizard of Oz experiments in which subjects used a simulated computer-based calendar system with a natural language interface.

The system she describes, CHAMP, starts out with a kernel grammar that can be augmented through repeated use by the same speaker. When the system receives an utterance from the user, it attempts to parse it with its potentially augmented grammar. If an exact parse is not possible, recovery strategies are attempted in what she calls a least-deviant first manner. This approach was first explored in Carbonell and Hayes's MULTIPAR parser

(Carbonell and Hayes, 1984). These recovery strategies include combinations of the following four operations on grammar rules: insertion, deletion, substitution and transposition. On the lexical level, these operations can be used to detect and correct spelling errors. On the syntactic level, these can be used to detect and correct grammatical errors. If a successful parse results from the application of a small number of these operations, the system asks the user to confirm that the system's interpretation is correct.

Once the parser has found the least-deviant parse, the next step is to extract some set of new grammatical components with which to augment the grammar. CHAMP's adaptation mechanism embodies two competing guidelines. First of all, the new components must be accessible to future parses so that other sentences like the current one can be recognized directly, with no deviation. And secondly, the new components must not add undue cost to processing sentences in which these new components play no role. In other words, the goal is to avoid both overgeneration and undergeneration. CHAMP employs three principles in order to follow the guidelines described above. First, only the constituent where the recovery occurs is relevant to learning. Second, because the grammar is hierarchical, changing one rule causes the learned pattern to apply not only in the exact context in which it first occurred, but also in every grammatically similar context. If a noun phrase rule is modified, the new pattern may occur wherever noun phrases may occur. In this way, an appropriate degree of generalization is enforced. Finally, when more than one method of adaptation is possible, the one that introduces the least amount of ambiguity into the grammar is chosen.

Lehman's results indicate that the model on which her implementation is based is effective. In an experiment in which six users participated in three to nine sessions with CHAMP, results indicate that quite a bit of adaptation takes place in the first few sessions, anywhere from three deviations for every seven sentences parsed, to thirteen adaptations for thirteen sentences parsed. By the ninth session, adaptation to sentences parsed ratios ranged from zero to two out of fourteen. Thus, the approach appears quite successful since eventually the system and the user adapt to one another. The main weakness of the approach is that if multiple users use the same system, the grammar quickly becomes intractable because the types of deviation exhibited by different users are not consistent with one another. She writes, "If our goal were to understand every user with a single system, the limit on extendibility would be the union of all idiosyncratic grammars, and the ambiguity inherent in such a language description would place the goal computationally out of reach" p. 4 in (Lehman, 1989). So her approach is appropriate only in contexts where

the system will be consistently used by a single user over time. In addition, the domain in which CHAMP was evaluated is very restricted.

2.2.1.2 Smith and Hipp

Hipp's parser (Hipp, 1992) which is used with Smith's discourse model (Smith, 1992) is a minimum distance parser similar to Lehman's. Smith and Hipp's combined work is conducted in the context of a human/computer dialog system where the user communicates with the system through speech. Similar to Lehman's CHAMP, Hipp's minimum distance parser attempts to find the parse with the minimum penalty from insertions and deletions on grammar rules. What is different about his parser is that it parses over an n-best lattice[2] (returned from the speech recognizer) rather than over a linear sequence of input words. This is advantageous because the recognizer returns multiple interpretations, none of which may be completely correct. The Smith and Hipp approach relies on user interaction to verify that the interpretations of ungrammatical input are correct. Their system ranks repair hypotheses based on (1) how much the parser needed to deviate from the grammar in order to find a parse and (2) how coherently each hypothesis attaches to Smith's discourse model. The Smith and Hipp approach makes use of dialogue expectations from the discourse model to resolve ambiguity, anaphora, and ellipsis.

Smith and Hipp report good results, claiming that by introducing verification dialogs they can raise the effective accuracy of the parser from 83% to 97%. But it should be noted that 52% of the utterances included in this evaluation were trivial utterances, typically consisting of only one word. Only 43% of non-trivial utterances were ungrammatical, even taking speech-recognition errors into account. 69% of the non-trivial utterances were parsed correctly without verification, 12% more than the percentage of non-trivial sentences which were grammatical to begin with. They do not mention what percentage of non-trivial ungrammatical utterances were parsed correctly with verification. Also, they consider a parse to be correct if the parser gets at least half of the meaning of the input utterance, but they do not specify what half of the meaning means. They also do not report the average number of questions which the system must ask the user in order to get the correct interpretation.

One major weakness in their approach is that Smith's discourse model operates in a purely top-down fashion, starting with a representation of the current discourse state from which a list of templates for possible next sentences is generated. In a larger, less constrained

[2]The Circuit Fix-It Shop's speech recognizer produces a lattice encoding the n-best hypotheses about what the speaker has said rather than simply returning its best guess.

system the possibilities for next utterances in many cases are open-ended enough to make this purely top-down approach unmanageable.

2.2.1.3 Ramshaw

Ramshaw (1994) uses context in a similar way to the Smith and Hipp approach. The first major difference between his approach and the Smith and Hipp approach is that Ramshaw's discourse model uses bottom-up plan inferencing that makes use of top-down constraints, whereas the Smith discourse processor is entirely top-down. Additionally, his approach generates repair hypotheses in a different manner. Ramshaw assumes that there is exactly one error in each sentence caused by exactly one word. It is unclear how his approach extends to the case where there may be more than one error in the sentence. Rather than use a minimum distance parser, he generates a list of variations on the input sentence, each with a different one of the words wildcarded. These wildcarded versions are then parsed into partial meaning representations, each of which contain unbound wildcard variables. These partial interpretations are then matched with the discourse context to see which will attach most coherently. The context serves both for selecting among hypotheses and for binding these unbound wildcard variables. While his discourse model seems more practical than Smith's, his approach to generating repair hypotheses seems tailored to his particular problem of fixing sentences that contain exactly one error.

2.2.1.4 Intractability of Minimum Distance Parsing

In practice, the Minimum Distance Parsing approach has only been used in very small domains. It is important to note the scale of the project providing the context for work of this nature. Flexible parsing algorithms introduce extra ambiguity to a greater or lesser extent, which may deem certain approaches impractical for systems of more realistic scale. Lehman's core grammar, described in (Lehman, 1989), has on the order of 300 rules, and all of the inputs to her system can be assumed to be commands to a calendar program. Hipp's Circuit Fix-It Shop system, described in (Hipp, 1992), has a vocabulary of only 125 words and a grammar size of only 500 rules . ROSE was developeded in a system on a much larger scale, a speech system with vocabulary size on the order of 3000 words, and grammar size also on the order of 1000 rules. The question is whether the Minimum Distance Parsing approach is still practical in a system of this scale or whether a more restrictive algorithm is necessary. In this dissertation I demonstrate that the MDP approach is intractable in a system of this scale and that the ROSE approach performs orders of magnitude more efficiently.

2.2.2 More Restrictive Approaches

More restrictive parsing algorithms trade off coverage for speed. An example of a more restrictive parsing algorithm is Lavie's GLR* skipping parser described in (Lavie, 1995). This parser is capable of skipping over any portion of an input utterance that cannot be incorporated into a grammatical analysis and recovering the analysis of the largest grammatical subset of the utterance. In other words, it performs deletions but not insertions, or any other edit operation. Partial analyses for skipped portions of the utterance are also returned by the parser. Thus, whereas MDP considers insertions and transpositions in addition to deletions, GLR* only considers deletions. The weakness of this and other partial parsing approaches (Abney, 1996; Van Noord, 1996; Srinivas et al., 1996; Federici, Montemagni, and Pirrelli, 1996) is that part of the original meaning of the utterance may be thrown away with the portion(s) of the utterance which are skipped, if only the analysis for the largest subset is returned, or part of the analysis will be missing if the parser only attempts to build a partial parse. Trading off coverage for speed, the idea is to introduce enough flexibility to gain an acceptable level of coverage at an acceptable computational expense. In this section I discuss several early approaches to limited flexibility parsers.

2.2.2.1 Hobbs

In (Hobbs et al., 1991), Hobbs et al. describe their approach to robust handling of naturally occurring text in the TACITUS system[3]. They start with a bottom-up chart parser. They assert that a bottom-up parser is more robust since it creates edges which would not be created with a top-down parser. These added edges allow for more flexibility but they cost the system in terms of efficiency. For this reason they add an "agenda mechanism" which allows them to order the edges in the chart according to how likely they are to be part of the correct parse, so they can be handled efficiently. The edges are ranked according to a preference score based on structural characteristics of the constituent associated with each edge. When they cannot derive a correct parse for a sentence, they attempt to span the chart with the fewest number of highest-scoring edges. The types of constituents they look for include main clauses, verb phrases, adverbial phrases, and noun phrases.

They report that they can extract most of the propositional content of a sentence from these pieces with the help of a pragmatics component. Also, a method which they employ with sentences longer than 60 words is to break them into segments at commas, conjunctions, relative pronouns, and instances of the word "that" in particular contexts.

[3]This is similar to the approach described by Jensen et al. in (Jensen et al., 1984)

Their parser steps through each segment searching for an analysis either only of it, or for an analysis that covers both it and previous segments. By proceeding this way, they claim the parser is more efficient because it does not have to look inside of the analysis for each segment for attachment sites each time it combines two or more segments.

Their approach was evaluated on its ability to process one-page newspaper articles, extracting information in order to fill in templates. In the MUC-3 evaluation of text-understanding systems (Hobbs et al., 1991), the TACITUS system ranked the highest in precision[4] at 65%. Their recall of 44%, the percentage of correct fillers ignoring incorrect ones, ranked somewhere in the middle.

2.2.2.2 McDonald

David McDonald describes a similar approach in the Sparser system in (McDonald, 1993b; McDonald, 1993a; McDonald, 1992; McDonald, 1990). Again, he makes use of a bottom-up chart parser, and he ranks the edges similarly for efficiency. One major difference between his approach and the Hobbs et al. approach is that he makes use of a semantic grammar instead of a syntactic one. He created this semantic grammar by starting with a syntactic grammar and then dividing the syntactic categories according to semantic distinctions. In this way, his grammar only derives syntactic structures, but it makes use of semantic constraints. This allows him to ensure that no parse will be derived for which a semantic interpretation cannot be derived. It also cuts down on attachment ambiguity because only semantically valid attachments will be made.

Sparser begins the parsing process by locating function words and punctuation which allows it to segment the text similar to the Hobbs et al. approach. Sparser attempts to deal with the unknown word problem by gleaning as much information as possible from capitalization and morphological endings. For example, a sequence of capitalized unknown words is likely to be a proper name. A word ending in *-ed* is perhaps a verb in the past tense. Sparser attempts to find the largest constituent in each segment. Its chart parsing algorithm attempts to combine sets of edges that match grammar rules. It employs a heuristic that allows it to skip intervening edges as it searches for rules that match the set of edges in the chart. In combining edges, it delays attachments to the left until no more attachments to the right can be made. In this way, it avoids the combinatorial explosion of constantly examining multiple attachment sights, which would have been necessary had this heuristic not been in place. This amounts to a preference for attaching low.

[4]Precision is the ratio of correct fillers out of the total number posited by the system.

Both Hobbs and McDonald employ grammar-specific heuristics, a sub-optimal strategy since it prevents the approach from being a general solution. It should also be noted that unlike any of the other approaches reviewed in this document, both Hobbs and McDonald have focused on robust handling of *text*[5]. It is true that many of the ideas at the heart of their work carry over to spoken language systems, such as the idea of combining partial parses. One must always ask, however, what assumptions behind their approaches do not apply in dialog situations, whether spoken or transcribed.

One major assumption which does not carry over is that the input is grammatical[6]. Hobbs and McDonald can both assume that if they cannot derive a complete parse for a sentence, they can always find analyses for most portions of the sentence since it is largely grammatical and is made up of prepositional phrases, noun phrases, verb phrases, and so forth, which the system should be able to cover. This must be taken into account if one were to apply their techniques to a dialog situation where this assumption is not valid, especially when recognition errors are taken into account.

2.2.2.3 Ward

Ward (Ward, 1989; Woszcyna et al., 1993) argues for a different type of semantic/skipping approach originally developed for the ATIS domain. He uses a concept-based pattern-matching parser called Phoenix, which is similar to Carbonell and Hayes's DYPAR parser (Carbonell and Hayes, 1984).

Rather than do a detailed syntactic analysis of the input sentence, a set of concept grammars is matched against each position of the input sentence in order to identify portions that express concepts relevant to the domain. These concept grammars are made up of patterns of semantic tags which match ways of expressing domain specific pieces of information. Each tag specifies a separate component of the meaning of the corresponding type of expression, such as from-city, to-city, departure-time, etc. Unlike McDonald's approach where only syntactically grammatical structures may be derived, Ward style grammars do not enforce any syntactic restrictions. These context-free grammars with semantic non-terminals are compiled into ATNs that parse the text, inserting semantic tags to label portions of text. No feature structures are built, only context-free parse trees. Since the grammars are matched against every position of the input, a lattice of possible parses is produced. The parse that accounts for the largest number of words is returned. Parses can also be ranked statistically or based on how well they match expectations generated by a

[5]Text here refers to published text such as newspaper articles, not simply input that is not speech.

[6]Hobbs and McDonald process text from the AP Newswire. Although this text is not 100% grammatical, it is arguably more formal and grammatical than spontaneous speech.

dialog model. Ward and Young make extensive use of predictions derived statistically from their dialog model (Young and Ward, 1993; Young, 1990; Young, 1993).

With a parser such as this one, rather than the known word vs. unknown word distinction, there is simply a notion of whether a portion of text matches one of the predefined patterns or not. Everything that doesn't match is skipped. So unknown words that occur outside any of the concepts that fill slots in the associated frame pose no problem for the system. Additionally, a vocabulary list can be specified indicating that words outside of this list which are encountered anywhere in the utterance should be completely disregarded by the parser. This accomplishes something similar to McDonald's skipping heuristic in the case of skipping over an intervening unknown word in order to combine two edges. Apart from this, however, Phoenix has no mechanism for dealing with insertions or deletions within patterns it is trying to match.

The Ward approach is suboptimal if a feature structure representation is desired as output. Ward's output representation retains the exact ordering of the input, whereas feature structures are in principle orderless and contain features that can be tested. Because there are no features in Ward's formalism, and all of the relevant grammatical information is encoded in the semantic tags, a larger grammar must be written in order to make the same number of grammatical distinctions as in an otherwise equivalent feature structure based formalism like LFG.

Recently the Phoenix parser has been used in the scheduling domain (Mayfield et al., 1995b; Mayfield et al., 1995a). In (Lavie, 1995), Lavie reports that Phoenix was able to achieve an acceptable translation for 75% of 54 transcribed Spanish utterances. On a similar evaluation of 99 English utterances, the Phoenix was able to achieve 85.5% acceptable translations.

2.2.2.4 Tomita and Lavie

Lavie, in (Lavie and Tomita, 1993; Lavie, 1995), discusses extensions to Tomita's GLR parsing algorithm (Tomita, 1986a). This parser could be used with any syntactic or semantic grammar written for the standard GLR parser. So this approach can be easily adopted by systems which are already making use of Tomita's LR parser.

The idea behind Lavie's approach is to identify and parse the maximal subset of the input string that is found to be grammatical according to the parsing grammar. The mechanism by which it accomplishes this is discussed in Appendix A. Because Lavie's skipping parser makes use of the same LR tables as the standard GLR parser, and because it retains the ability to make use of local ambiguity packing, it maintains some degree of

the efficiency inherent in the LR approach. In fact, the computational complexity can be demonstrated to be the same as the original LR parsing algorithm (Lavie, 1995), although in practice the addition of the skipping mechanism slows the parser down considerably.

In (Lavie, 1995), Lavie reports good results on spontaneously generated sentences from the scheduling domain compared with the GLR parser and with the Phoenix parser. He reports that on a corpus of 798 transcribed Spanish utterances, only 2.1% of these utterances were unparsable by the GLR* parser, whereas 32.0% were unparsable with the standard GLR algorithm. GLR* derives good parses for 78.7% of these sentences overall on the same corpus. This compares very well with Jain's PARSEC parser described below. (Jain reports obtaining good parses for 78% of the cases in his corpus described in (Jain, 1991). It should be noted that Lavie's results are reported on a much larger and more difficult corpus.) Lavie also compares his results with those of the Phoenix parser on a translation task. His results demonstrate that the two achieve comparable levels of performance. On a set of 54 transcribed Spanish utterances, Lavie's GLR* was able to get 78.7% acceptable translations while Phoenix was able to get 75.0%. On a similar evaluation on 99 English utterances, Lavie's GLR* was able to get 83.9% acceptable translations where Phoenix was able to get 85.5%. The conclusion is that the GLR* performance is very similar to that of the Phoenix parser.

Lavie's GLR* parser returns the parse that covers that greatest number of words in the input utterance. The current version of the GLR* parser uses heuristic methods as well as statistical modeling based on Carroll and Briscoe's statistical LR parser, (Carroll and Briscoe, 1993), for selecting the correct parse. Lavie's approach is primarily designed to handle deletions. Substitutions are handled in a limited way by keeping track of words that are commonly confused by the speech recognizer, and no special provision is made for insertions. Nevertheless, the evaluation discussed above shows that his parser exhibits good performance on spontaneous language data.

2.2.2.5 Limitations on Limited Flexibility Parsers

All of these limited flexibility parsers are far more efficient than the minimum distance parsing approach. Nevertheless, even with their limited flexibility they are slower than parsers without this flexibility, though they have the same $O(n^3)$ complexity. The fact that these parsers slow down considerably although the computational complexity is equivalent is an important point since minimum distance parsers which allow insertions, deletions, and substitutions also claim $O(n^3)$ complexity. Intuitively, if skipping slows the

parsing algorithm down considerably, inserting and substituting would slow it down even more.

Though limited flexibility parsers offer a computational advantage over MDP parsers, their limitations in flexibility place limitations on their ability to handle extragrammaticality. In particular, none of these limited flexibility parsers can perform insertions. Therefore, they cannot produce a full analysis for any utterance that is missing an essential constituent for constructing a full parse of the sentence. They also can not handle the case where constituents are ordered in an unexpected way. Though the evaluations of these parsers indicate that a limited flexibility parser can go a long way towards reaching a desirable level of robustness, they are not in themselves a complete solution to the problem of extragrammaticality.

The ROSE approach is a two stage approach in which a post-processing repair stage is introduced. The goal is to increase the coverage possible at a reasonable computational cost. The repair stage constructs a complete meaning representation out of the fragments of a partial parse. Since the input to the second stage is a collection of partial parses, the additional flexibility that is introduced at this second stage can be channeled just to the part of the analysis that the parser does not have enough knowledge to handle straightforwardly. This is unlike the MDP approach, where the full amount of flexibility is unnecessarily applied to every part of the analysis. Therefore, as I will demonstrate in this dissertation, this two stage process is more efficient since the first stage is highly constrained by the grammar, and the results of this first stage are then used to constrain the search in the second stage. In cases where the limited flexibility parser is sufficient, the repair stage can be entirely bypassed, which is another advantage of the two stage approach over the MDP approach.

2.3 Connectionist Approaches

Above I discussed Lehman's symbolic approach to learning and adaptation (Lehman, 1989; Lehman and Carbonell, 1989). Subsymbolic approaches include statistical and connectionist methods. Though the ROSE approach to robust interpretation of language is primarily a symbolic one, it makes use of information provided by subsymbolic components. For that reason I review two of the leading subsymbolic approaches to language interpretation as well. Though robustness can be achieved through both symbolic and subsymbolic means, there are definite trade-offs between the two approaches. The flexibility of the subsymbolic approach is built into the architecture, whereas flexibility must be added heuristically to the symbolic approach. But the symbolic approach gives the system designer

more control over types of flexibility the system will exhibit. For a more in depth discussion of these trade-offs, see Klavans and Resnik's collection (Klavans and Resnik, 1997).

2.3.1 Jain

PARSEC, described by Jain and Waibel in (Jain, 1991; Jain and Waibel, 1990), is a good example of a connectionist approach to robust parsing in a practical natural language processing system. It has been used in the JANUS speech-to-speech translation system, (Woszcyna et al., 1993), just as the GLR* parser (Lavie, 1995) discussed above. Jain explains that parsers such as PARSEC have three advantages for robust handling of natural language. First of all, they can learn from examples and generalize well compared to hand-crafted grammars. Secondly, they can tolerate several different types of noisy input. And finally, it is easy to teach them how to use multi-modal input. PARSEC is highly structured compared to other connectionist parsers. It is built from six modules, each dedicated to a particular sub-task, and each including a recurrent backpropagation network with some glue-code that maps words and feature structures to and from the bit-mapped representations that form the interfaces of each network. The design principle behind PARSEC's architecture is to let the connectionist component solve the problems that it is good at, such as sensing phrase boundaries, and to handle the rest with a symbolic approach. In an evaluation of PARSEC's performance, Jain reports a 78% success rate, but on a much smaller and simpler corpus than that used in Lavie's evaluation described above.

Polzin (Polzin, 1994; Woszcyna et al., 1993) has extended PARSEC to process structures of arbitrary depth by making it auto recursive[7]. This makes PARSEC much more powerful since it gives it the ability to produce structures for complex, multi-clausal sentences. It also makes it possible for PARSEC to learn faster. Since it can now make more abstraction steps, each step is smaller, and thus easier to learn. Unfortunately, this makes the whole parsing process much slower.

2.3.2 Gorin

Gorin et al. present a hybrid connectionist/statistical approach based on mutual information between the set of words in input utterances and meaningful system responses in (Sanker and Gorin, 1993; Miller and Gorin, 1992; Gertner and Gorin, 1993). The Gorin approach is similar to Lehman's in that it is also interactive. It learns while it is performing its task based on the feedback it gets from the users. Unlike Lehman's approach, however,

[7] Auto recursive means that the output can be cycled back into the input in order to get the neural net to do roughly the equivalent of recursive processing.

Gorin does not assume his system will be used by only one speaker. His approach does not run into the same problems with the grammar becoming too large to be tractable since his grammar is, to a large extent, implicit in the mutual information statistics the system keeps. The amount of storage required to keep this information increases very little over time, only as new words are added to the system. Gorin (Gertner and Gorin, 1993) reports an experiment in which 13 users interacted with his system over a period of two weeks, resulting in over 1000 dialogs, without reporting any difficulties with storage or speed as the system acquired new knowledge.

Gorin describes a number of different experiments that he has conducted with his mutual information networks in different configurations, ranging from a system that maps user utterances onto departments in a department store, for use as a filter to direct callers to the right department, to a simple version of an airplane reservation system. In each case, he makes use of the same basic network architecture, inspired by Magerman and Marcus's work in parsing with mutual information statistics (Magerman and Marcus, 1990). I describe this network architecture in detail here since I make use of it in my own work.

Gorin's basic two-layer network architecture is as follows: There is one input node corresponding to each word that has been encountered. Each node on the output layer corresponds to a semantic action. The connection weight between an input node n and an output node m is the current mutual information between word n and semantic action m plus a bias equal to the frequency of the associated output node. Mutual information is roughly a measure of how strongly associated two concepts are. It is defined by the following formula:

$$log[P(c_n|v_m) \div P(c_n)]$$

where c_n is the nth element of the input vector, which is the ordered set of input nodes, and v_m is the mth element of the output vector, which is the ordered set of output nodes.

This network architecture can be expanded to three layers by adding an intermediate layer in which each node represents a word pair or triple. Because what roughly corresponds to "parsing" in Gorin's system is merely calculating the mutual information between an utterance and a semantic action based on current relative frequencies, it can accept totally unconstrained input.

Gorin's approach relies heavily on feedback from users. The user first begins a dialog with the system, making a request for a particular action. The system then responds, indicating which action it believes the user is requesting by calculating the mutual information between the utterance and each semantic action and selecting the one with the highest score. The user then responds to the system, indicating the appropriateness of

the system's interpretation and possibly making clarifying comments. This interaction continues until the user and system come to some agreement on what the user is asking for, or if it does not converge within some arbitrary number of utterances, the system gives up. If and when a dialog converges, relative frequency statistics can be recalculated. Gorin reports that the non-convergence rate decays exponentially as the maximum dialog length increases.

The dialog between user and system is guided by a confidence model which allows the system to calculate how confident it is that the semantic action it has chosen as the most likely one is in fact the intended one. A confidence vector is first calculated by taking the average of output activation vectors, one corresponding to each user input[8]. The confidence value is calculated as follows: if the difference between the largest and second largest value in the confidence vector is greater than 5, the confidence rating is very confident. Otherwise, if the difference between the largest and smallest values in the confidence vector is less than or equal to 2, the confidence rating is moderately confident. Otherwise, it is rated low confidence.

Gorin describes how he used this confidence rating to guide interactions in a system where users asked about attributes of states, for example, the state flower of California. If the system is very confident about both the state and the attribute, it would go ahead and give the corresponding answer, namely, the Golden Poppy. If the system is moderately confident about the state but low in its confidence of the attribute, it would ask for clarification about the attribute. Likewise, if it was moderate in its confidence about the attribute but low in its confidence about the state, it would ask for clarification about the state. If it was low in its confidence for both the state and the attribute, it would guide the user, first asking for the state and then the attribute. In other cases, it would confirm that its interpretation was correct, for example, "Did you want to know the state flower of California?" The confidence rating allows the system to guide its interaction with the user in such a way that it shows as much understanding as possible so as not to frustrate the user. This simple heuristic works for his system since the user can always be assumed to be asking for the value of an attribute as it relates to a particular state.

There are several trade-offs between approaches such as Gorin's and approaches such a Lehman's and Hipp's, and neither seems to be satisfactory for a system such as the one in which ROSE was developed. For example, as an input utterance becomes more and more garbled, Gorin's approach remains a viable solution. But more garbled sentences are more difficult for Hipp and Lehman's approaches. Neither of them present data in

[8]Gorin assumes the purpose of each of the user's sentences within the same dialogue are an attempt to communicate the same idea.

their theses about how the performance of their systems degrades as input becomes more garbled, but intuitively, if the parser must make more insertions and deletions in order to find a parse for the utterance, there will be much more ambiguity to deal with. To deal with this difficultly, they both make heavy use of contextual information. Assuming this information is reliable, this may remain a viable solution, but this remains to be seen. On the other hand, as the task becomes more complex, Gorin's approach becomes less tractable in its pure form. For example, the set of possible meanings to be handled even in a domain such as scheduling is infinite[9], so it would be impossible to construct a network that maps sets of words onto their respective meanings. An ideal solution would combine the benefits of both approaches while avoiding the respective pitfalls. ROSE is an attempt to do just that.

2.4 Learning and Adaptation

Lehman, Gorin, and Jain all take different approaches to learning. Lehman and Gorin both take a reinforcement learning approach where learning continues as the system is in use. Feedback consists of an acknowledgement of whether the task was carried out correctly. On the other hand, backprop nets such as those employed in PARSEC are trained off-line by example, and feedback consists of error signals computed from sets of exact input/output pairs. The advantage of the reinforcement approach is that it continues to learn as the system performs its function. This reduces the need to collect large quantities of training data ahead of time.

A major advantage of the subsymbolic approach is that the system does not become unmanageable as it continues to learn. On the other hand, in the symbolic approach, if the grammar becomes large from learning new rules, it becomes intractable.

2.5 Interactive Repair

Lehman, Hipp, Gorin, and Haas and Hendrix (Haas and Hendrix, 1983) all make extensive use of interaction with the user in their adaptation mechanisms. The idea of interactive language analysis along with examples of early approaches more along the lines

[9]Of course one could argue that the meaning representation could be very simple and finite if you consider only two possible general meanings, i.e. busy or free, along with specifications for meetings within a particular time period. Where it would be certainly possible to limit the representation in this way, it would not cover a large portion of utterances which occur in spontaneous scheduling dialogues about reasons for being busy or free or other information about contingencies in schedules. Also, one must consider that in a single sentence more than one meeting specification could be given, for example, a list of meeting times could be given, and some of these times will be ranges, like "between 10 and 12 on Thursday".

of disambiguation than repair are discussed in (Carbonell and Tomita, 1987). This work was conducted in the machine translation community for the purpose of avoiding the need for pre-editing and post-editing. Here I identify two main issues in this type of user interaction. The first issue is what type of questions to ask. The second is when to ask and when not to ask.

Both Lehman and Hipp direct their questions at verification of complete utterances (Global Repair Hypotheses) as in "Did you mean to say X?" This is because their repair mechanism works by trying to construct a complete parse of the input sentence. And when they have done this, they have the complete meaning all at once rather than uncovering it incrementally. Gorin, on the other hand, asks questions about specific portions of the user's utterance (Local Repair Hypotheses). This is because his repair mechanism works by extracting different parts of the meaning of the utterance from different networks. This allows the questions which it asks to be more pointed, specifically focused on what the system does not know. In Gorin's application, there are few enough variables that this approach is reasonable. The approach discussed in (Carbonell and Tomita, 1987) is a multiple-choice type of approach which makes sense when the decision can be narrowed down to a small set of possibilities and the system is text-based. Haas and Hendrix have the most strikingly different approach in which they verify the implications for the knowledge base in terms of where the new knowledge from the utterance fits in, rather than verifying the meaning of the utterance. The recovery mechanism itself discussed in (Haas and Hendrix, 1983) is very simple. New words are matched with a predicate that is expected given the semantic context. Two important issues not specifically addressed by any of these authors are which approach makes it possible to ask the fewest number of questions, and which approach is the least frustrating for the user. These questions will be revisited later when ROSE's Interaction with the User phase is discussed and evaluated.

The issue of when to verify a possible repair is a separate issue from what types of information to verify. The question of when to verify boils down to how confident the system is that its interpretation is correct. There are three possibilities that a system designer can take. The first is to verify everything. This is the approach Lehman takes in her dissertation work. This has the advantage that the system will always eventually get its interpretation correct, but it can be annoying for the user. Another possibility is to never verify anything, just to take as the correct one the repair hypothesis that scores best according to some measure. This has the advantage of putting much less of a demand on the user. But it has been demonstrated that some user interaction is advantageous. For example, Hipp reports an increase of 15% in parser accuracy with selective verification

in (Hipp, 1992). The third approach, the one taken by both Hipp and Gorin, is to verify selectively. In order to accomplish this, there must be an accurate measure of how confident the system is in its answer so that it will rightly use interaction just in those cases where it is not sufficiently confident. It also presupposes that the system is confident in the same cases when it is right, which cannot be assumed to be the case.

2.6 The ROSE Approach: A Preview

ROSE[10]: RObustness with Structural Evolution, repairs extragrammatical input in two stages. The first stage, Repair Hypothesis Formation, is responsible for assembling a set of hypotheses about the meaning of the ungrammatical utterance. This stage is itself divided into two steps, Partial Parsing and Combination. In the Partial Parsing stage, Lavie's GLR* parser (Lavie, 1995; Lavie and Tomita, 1993) is used to obtain an analysis of islands of the speaker's sentence in cases where it is not possible to obtain an analysis for the entire sentence. In the Combination step, the fragments from a partial parse are assembled into a set of meaning representation hypotheses. In ROSE's second stage, Interaction with the User, the system generates a set of queries and then uses the answers to these queries to narrow down the set of choices to a single best meaning representation hypothesis.

As discussed above, the most important criteria for evaluating an approach to robust interpretation of language is which approach uses the available resources most economically. Important resources to consider are space (both static and dynamic), time (both development time and run time), and interactional effort. In this dissertation I compare ROSE to two alternative approaches, namely the Minimum Distance Parsing (MDP) approach and Incremental Repair with Local Hypotheses (IRLH). Though space is also an important resource, I focus my evaluation on a comparison of time and interactional effort.

MDP, IRLH, and ROSE are all domain independent and portable, and therefore require no additional system development time above what would normally be required for any symbolic natural language interface. Therefore, they are equivalent in terms of development time. However, rather than placing the full burden of robustness on a single parsing algorithm as in the MDP approach, in this dissertation I argue that it is more efficient in terms of run time for Partial Parsing and Combination to be separate steps in the Hypothesis Formation stage as in ROSE. Another goal of this work is to demonstrate that it is more efficient, in terms of interactional effort, to separate Repair Hypothesis Formation from User Interaction as in ROSE rather than interleaving them as in IRLH. In other words, a set of alternative ways of fitting the whole set of fragments from the partial

[10]ROSE is pronounced Rosé, like the wine.

parse (Global Repair Hypotheses) is constructed before any queries are generated, rather than generating a query to verify each repair step (Local Repair Hypotheses).

Therefore, it will be demonstrated that the ROSE approach is not only domain independent and portable, but that it also robustly extracts the meaning from the user's extragrammatical utterance efficiently and without placing an undue burden on the user in terms of interactional effort.

Chapter 3

Genetic Programming: Background and Relevant Literature

Humans who share a very small language base are able to communicate with one another by simplifying their speech patterns and negotiating until they manage to transmit their ideas to one another. As the speaker is speaking, the listener "throws his net" in order to catch those fragments of speech that are comprehensible to him, and which he then attempts to fit together semantically. The search process for combining these fragments in ROSE is accomplished using a genetic programming technique. In this chapter I provide some background on this search paradigm. For a more in-depth discussion of genetic programming see (Koza, 1992; Koza, 1994). Genetic programming is a method for "evolving" a program to accomplish a particular task. It is an extension of the traditional genetic algorithm originally introduced in (Holland, 1975) and developed further in (Michalewicz, 1994).

3.1 Background on the Genetic Algorithm

The genetic algorithm is a machine learning approach inspired by the Darwinian model of evolution (Darwin, 1859). Character strings that represent chromosomes or genetic features encode data which is used by some program. The "fitness" of a character string is computed by measuring how well the program performs given that character string as data. A population of character strings is evaluated for their fitness value. A set of strings that are relatively more fit than the rest of the population are chosen to "mate" with each other and produce a new set of strings for the next generation. When these strings "mate", an operation called *crossover* is used. Some position is chosen as a crossover point. Two new strings are then produced. The first one contains the part before the crossover point from the first parent concatenated with the part after the crossover point from the second parent. The other one contains the part before the crossover point from the second parent

concatenated with the part after the crossover point from the first parent. The idea then is that what is good about one string will hopefully be combined with what is good about the second string. In this way, partial solutions to a problem can be built up over time. In addition to *crossover*, another operation that affects the next generation is *mutation* where a single character in a small subset of strings is randomly changed. This is to partially compensate for what "genetic diversity" is lost as the population becomes more and more focused on those strings that seem to be the most fit, until a string is evolved that reaches some threshold of fitness or some maximum number of generations are exceeded.

Holland (1975) describes what is called the Schema Theorem which explains why genetic algorithms work. I will briefly sketch his argument here. The idea is that on each generation, the current population of strings are evaluated for their fitness value. The question is what is it that makes the good strings good and the bad strings bad? Each string can be associated with a set of what are called schemas which describe a class of strings. For example, say there is a string which looks like the following: 101, and we want to ask ourselves what it is that makes this string as good as it is. This string can be associated with the following 7 schemas: 1**, *0*, **1, 10*, 1*1, *01, 101. Each of these schemas can be thought of as a potential argument for why the string is good. For example, the second schema is equivalent to the argument that the string is good because the middle character is 0. The third schema is equivalent to an argument that the string is good because its last character is 1. Ideally, the genetic algorithm would find out which schemas are responsible for the relatively high fitness of the good strings and combine them into one best string. This problem is similar to what is called the Two Arm Bandit problem where there is a slot machine with two arms, with one which has a better pay-off rate than the other one. Obviously, if it was possible, a gambler would rather continually pull the arm with the better pay-off. So the question is, how can the gambler find out which arm has the better pay-off in such a way that he maximizes his prize? He has to trade off how certain he wants to be that he has the right arm with how much money he wants to waste on the bad arm. Each schema represented in a population of strings can be thought of as an arm in a Multi-Arm Bandit problem. The genetic algorithm is the gambler who tries to bet on schemas that seem likely to be ones with the best pay-off, i.e. fitness. In the Schema Theorem, Holland demonstrates that the genetic algorithm with crossover and mutation causes the good schemas to increase from generation to generation at an exponential rate which, according to his Multi-Armed Bandit theorem, is the optimal rate. A very clear explanation of this can be found in Chapter 3 of (Koza, 1992).

3.2 Background on Genetic Programming

Genetic programming (Koza, 1992; Koza, 1994) is an extension of the traditional genetic algorithm where the structures that are undergoing adaptation are computer programs, which dynamically vary both in size and shape. This search paradigm is a technique for finding the most fit computer program for a particular task. This approach can be used to evolve computer programs in any computer language, though in this dissertation it will be used to evolve only Lisp programs.

Since the space of possible computer programs is infinite, to simplify the problem and focus the search, genetic programmers generally impose limits on the depth of the programs which they evolve. A genetic programming algorithm starts out with a set of functions that can be composed in order to build a program and a set of terminals (i.e. variables and constants) to provide the leaf nodes of the program, which can be thought of as a tree. Additionally, a set of fitness cases are provided over which a population of programs can be evaluated for their fitness. These are generally bindings for the variables that are passed in or cases to run the program over. In the 0th generation, a set of programs are generated randomly. Usually the programs in this initial population have a low fitness. But some programs will have a better fitness than others. Those which are relatively more fit are selected as in the traditional genetic algorithm for reproduction. A modified version of the *crossover* and *mutation* operators are used. The *crossover* operation swaps a subprogram from one parent program with a subprogram of the other parent program. The mutation operator swaps a function with another randomly chosen function or a terminal with another randomly selected terminal.

Nothing as concrete as the Schema Theorem has been demonstrated for genetic programming, but in practice this technique has been shown to behave similarly to the genetic algorithm where the equivalent to schemas is subprograms which form the building blocks for the full solution to the problem. A discussion of this can be found in Chapter 6 of (Koza, 1992).

3.3 Application of the Genetic Algorithm and Genetic Programming to Computational Linguistics

The genetic algorithm and genetic programming techniques have not been applied very widely to computational linguistics. Here I will review the applications which I am aware of.

Losee (1995) describes an approach to using the genetic algorithm to evolve parsing grammars to be used for document retrieval. Berwick (1991) and Clark (1991) both describe applications of the genetic algorithm to evolving parameter settings for parameterized GB grammars.

A review of applications of genetic programming to computational linguistics can be found in (Koza, 1994). Two examples are given. The first example is Siegel's (1994) work in which genetic programming was used to induce decision trees for word sense disambiguation. The other example was Nordin's (1994) work in which genetic programming was used to evolve machine code programs for classifying spelled out Swedish words as nouns or pronouns.

I am not aware of any application of genetic programming or the genetic algorithm to recovery from parser failure or any analogous problem.

3.4 How to Apply the Genetic Programming Technique

The genetic programming technique can be applied to a wide range of problems. In general, it can be applied to any problem that can be solved with a computer program. There are five steps involved in applying the genetic programming paradigm to a particular problem:

- Determine a set of terminals.
- Determine a set of functions.
- Determine a fitness measure.
- Determine the parameters and variables to control the run.
- Determine the method for deciding when to stop the evolution process.

These five steps are discussed in some detail below. A more in depth discussion of these five steps can be found in Chapter 7 of (Koza, 1992).

3.4.1 Terminals and Functions

All of the programs evolved with the genetic programming technique are composed of the set of terminals and functions that are initially provided. Terminals can be variables, constants, or structures of any kind. Terminals that are variables become parameters for the programs evolved. The functions can be either primitive functions in the programming language (e.g.. Lisp), or they can be custom-made functions for the particular task for

which the program is being evolved. For instance, if you are trying to evolve a program to move a block from one location to another location, you can start with a set of primitive Lisp functions, since it is certainly possible to write a Lisp program to do the desired data structure manipulation. But it would be more efficient and straightforward to start with functions that are directly relevant for the task, such as a function that selects a block, a function that can pick up a selected block, and a function that can put a block down. The most straightforward approach is to select the functions and terminals in such a way that the following principles are adhered to:

- It is possible to construct a program with the terminals and functions provided which accomplishes the desired task.
- Each function can take any of the terminals or the result of any of the functions for any of its arguments.

3.4.2 The Fitness Function

The fitness measure is the most critical part of any application of genetic programming since the fitness evaluations are what guide the search process. It is also the part which distinguishes genetic programming from other machine learning techniques such as neural networks. Neural networks make use of a technique called backpropagation, which compares the result of the current network with the desired result and adjusts the weights to make the distance between the actual result and the desired result smaller. Genetic Programming is more indirect. There is not in general an ideal result for the program given a specific set of inputs to compare the actual result with. And even if there were, there is no algorithm for modifying a computer program in order to get one that generates output that is a specific adjustment on the output of the current program. Rather than compare the resulting program to an ideal in this way, the fitness function measures how well the evolved program performs at a particular task. The fitness function assigns a better score to those programs that perform better. This information would be useless in training a neural network.

For example, suppose that the goal of the learning process is to simulate a state machine. This could be accomplished either with a neural network or a genetic programming technique. The result in both cases would be a function that would take as input a state and an action in that state and produce as output the resulting state. The training cases could be triples composed of a state, an action, and a resulting state.

In the case of the neural network, for each training example on each epoch, the state and action would be encoded as the input to the network and propagated forward.

Then the resulting state would be decoded from the output. The resulting state would be compared with the desired resulting state. Then the weights would be adjusted in such a way as to make the resulting state encoding closer to that of the desired state. This distance information would not be useful to the genetic programming approach in the same way since there is not a general algorithm for automatically adjusting a program to make the output come out closer to an ideal output.

Instead, with the genetic programming approach, a measure of how well the program performed is used. This could be the same as the error function used for the neural net. But it could also be one that counts how many actions are required in order to get from one state to a desired target state in the state machine, and prefers a shorter path. Note that this would make it possible to evolve a state machine for a particular task without first knowing what transitions the state machine should learn. There could also be other evaluations, like a preference for state machines that make it possible to get from a start state to a target state with a particular set of functions, while avoiding some particular state. The possibilities are endless. Once a set of criteria is selected, it can be used to rank alternative hypothesized programs. When the best of these programs are used to generate the next generation of programs, the resulting set of programs will be more like the best programs than the ones that were less good. In the same way that information which was useful to the neural net approach is not useful to the genetic programming approach, information which is useful to the genetic programming approach would not be useful to the neural net approach. For example, knowing how well the state machine produced by the current network performs does not tell you anything about how to adjust the weights in order to make the performance better.

The advantage of this indirect learning is that it makes it possible to learn a function in cases where you do not know what the ideal function should look like, but once you have a function, you can tell how good it is.

3.4.3 Parameters and Stopping Criteria

Parameters for the run include things like the method of selecting programs to mate with each other, or the crossover or mutation rate. They also include things like the size of the population and the maximum number of generations. For any given problem, these parameters are generally determined experimentally. Sometimes a stopping criteria besides the maximum number of generations can be provided, which indicates how well the desired target program must perform. If a program that performs well enough is evolved before the maximum number of generations have been exceeded, either by some cut off

in fitness value or by some separate criteria, the genetic programming algorithm can be terminated early.

3.5 Genetic Programming in ROSE

Ideal Repair Hypothesis:

```
(MY-COMB                                                    ;insert arg2 into arg1 in slot
  ((FRAME *RESPOND) (DEGREE NORMAL) (TYPE NEGATIVE))        ;arg1
  ((TIME-OF-DAY MORNING) (NUMBER PLURAL)                    ;arg2
   (FRAME *SIMPLE-TIME) (SIMPLE-UNIT-NAME TOD))
  WHEN)                                                     ; slot
```

Ideal Structure:

```
((FRAME *RESPOND)
 (DEGREE NORMAL)
 (TYPE NEGATIVE)
 (WHEN ((FRAME *SIMPLE-TIME)
        (TIME-OF-DAY MORNING)
        (NUMBER PLURAL)
        (SIMPLE-UNIT-NAME TOD))))
```

Gloss: Mornings are out.

Figure 3.1: Combination Example

In ROSE, the search process for combining the set of fragments from the partial parse is accomplished using a genetic programming technique. A population of programs, such as the example in Figure 3.1 introduced in Chapter 1, is evolved which represents different ways of fitting the chunks together into a single meaning representation structure. The chunks themselves are the terminal symbols. A combination operator that attempts to insert one chunk into a slot inside of another chunk is the sole function in the function set. This combination operator makes reference to the interlingua specification in order to compute the total set of ways the one chunk can be inserted into a slot in the other chunk. When it is executed the first time, it selects one of these slots and instantiates its slot parameter. This ensures that if the same program is evaluated more than once, it will

always produce the same result[1]. Through this genetic search process, ROSE attempts to fit together semantically the comprehensible fragments "caught in its net" during the Partial Parsing step. This Combination step is described in greater depth in Chapter 7.

Note that the application of genetic programming in ROSE is strikingly different from most common applications of genetic programming, which require populations on the order of thousands of individuals and months of run time before a sufficient number of generations have been processed in order to converge upon an acceptable solution. The programs that generate repair hypotheses are much less complicated than the sorts of programs that solve the types of problems which are common applications of genetic programming. However, the genetic programming paradigm provides the right sort of opportunistic search environment required, because the system does not know a priori which repair strategy will prove most fruitful for any given set of chunks. The genetic programming algorithm samples a wide space of possibilities shallowly, and then pursues those strategies that appear to be most successful. ROSE's use of statistical information to bias the search allows it to converge upon a reasonable solution more quickly than would otherwise have been possible with genetic search.

[1]Otherwise, in cases where there are multiple possible slots in the parent chunk where the child chunk could be inserted, there would be no way to make sure the same slot would always be selected each time the program is evaluated.

Part III

In Depth Discussion of Alternative Approaches

Chapter 4

Incremental Repair with Local Hypotheses

The goal of my dissertation project is to demonstrate that the two stage ROSE approach can achieve better results more efficiently than either Incremental Repair with Local Hypotheses (IRLH) or the Minimum Distance Parsing (MDP) approach. In this chapter I describe an implementation of IRLH, which is described in greater depth in (Rosé and Waibel, 1994). IRLH is a "casting and combining" model similar to ROSE. What distinguishes both IRLH and ROSE from other two stage repair approaches (Hobbs et al., 1991; Ehrlich and Hanrieder, 1996; Danieli and Gerbino, 1995) is that neither of them rely on any hand crafted repair rules. Instead they have the ability to search for an acceptable combination of partial analyses by making reference to the meaning representation specification which describes the meaning representation language. This ability makes both IRLH and ROSE completely portable.

What distinguishes IRLH from ROSE is that it interleaves Hypothesis Formation with Interaction with the User. In IRLH, queries are generated to verify each repair step (Local Repair Hypotheses) rather than first constructing a set of alternative ways of fitting together the whole set of fragments (Global Repair Hypotheses). Because Local Hypotheses focus on specific aspects of the speaker's meaning rather than on the meaning of the speaker's whole utterance, it must focus its interaction with the user on clarifying the speaker's meaning rather than on furthering the task. This focus will be demonstrated to make it necessary to burden the user with more queries.

This implementation of IRLH is a hybrid statistical/symbolic approach. It takes as input a partial parse from the GLR* parser and returns a repaired meaning representation as output. It negotiates with the speaker about what the complete meaning of the utterance is by first generating hypotheses about how to fit the fragments of the partial parse together and then verifying each repair before it is made. In this way it is possible to insure that

every repair made is correct. By drawing upon both statistical and symbolic information, it is able to constrain its repair hypotheses to those which are both likely and meaningful.

4.1 Repair Process Overview

The repair processes both in ROSE and in IRLH are analogous in some ways to fitting pieces of a puzzle into a mold which contains receptacles for particular shapes. In this analogy, a meaning representation specification, described in depth in Appendix B, acts as the mold with receptacles of different shapes, making it possible to compute all of the ways partial analyses can fit together in order to create a structure which is legal in this frame based meaning representation. Readers not familiar with this meaning representation specification are encouraged to read Appendix B.

Since the number of possible meaning representation structures is so large[1] that a brute force search method is computationally intractable, mutual information statistics are used to guide the search. These mutual information statistics primarily encode regularities in the types of fillers which tend to occur in particular slots. They also compute which types of feature structures tend to be generated most often by particular non-terminal symbols in the parsing grammar at parse time.

The IRLH approach searches for the complete meaning representation structure by generating and testing hypothesised repair actions. When these hypotheses are confirmed to be correct through interaction with the speaker, the repair module then makes the specified repair. The primary problem with this approach is that each of these hypotheses are generated locally, with each decision building upon the result of the last successful hypothesis. So through trial and error, the repair module is forced to burden the speaker with a large number of tedious questions. Besides the problem of the large number of questions which this approach makes necessary, questions which focus on specific aspects of the speaker's meaning have been demonstrated to be annoying to people (Garfinkel, 1964).

4.2 Integral Knowledge Sources

As mentioned above, neither the IRLH repair module nor the ROSE one generate repair hypotheses randomly. Instead they draw upon both symbolic and statistical sources of information in order to ensure that the resulting repaired structure is meaningful, and that proposed repairs are likely to be correct.

[1] Since the meaning representation is recursive, the search space is potentially infinite.

The main symbolic knowledge source used is the interlingua specification. This specification is made up of rules which specify how to generate the full set of legal meaning representation structures, called interlingua structures since the meaning representation used in Enthusiast was designed to be language independent. Though this meaning representation specification represents knowledge which must be encoded by hand, it is knowledge which can be used by all aspects of the system, not only the repair module as would be the case with repair rules.

In addition to this prominent symbolic knowledge source, the repair module makes statistical predictions based on mutual information statistics stored in a set of networks (Gertner and Gorin, 1993; Miller and Gorin, 1992; Sanker and Gorin, 1993). Mutual information is intuitively a measure of how strongly associated two concepts are. It is defined by the following formula:

$$log[P(c_n|v_m)/P(c_n)]$$

where c_n is the nth element of the input vector and v_m is the mth element of the output vector.

The mutual information networks are used to bias the formation of three types of hypotheses in IRLH. This statistical information is used for ranking alternative hypotheses. Though it is not guaranteed to rank correct hypotheses over incorrect ones in all cases, it gives a useful indication of which hypotheses are more likely to be good than others.

The first type of hypothesis which the mutual information statistics rank are ones about which top level semantic frame is most likely given the whole set of chunks. A network is used that has input nodes corresponding to each non-terminal symbol in the parsing grammar and output nodes corresponding to each possible top level semantic frame. Since the grammar is a semantic one, the non-terminal symbols can make predictions about the larger meaning representations which the parser may have been attempting to build. To use the network to make these predictions, input nodes corresponding to each non-terminal symbol found among the parses for each chunk are set to 1 while the other nodes are set to 0. When activation is propagated forward, the output nodes will receive the activation equivalent to the mutual information between the set of input nodes assigned a value of 1 and the corresponding output node. If the input nodes which are set to 1 make strong predictions about the top level semantic frame, the output nodes corresponding to the associated top level frame will be activated. So, for example, if a complete parse is not possible, and no top-level semantic frame is built at parse time, but chunks are built corresponding to fragments that tend to be more associated with one top level semantic frame over another, the frame or frames they are associated with will be ranked higher. Since phrases about

doctor's appointments or classes more often occur with rejections than any other type of sentence in the scheduling domain, frames like `*busy` and `*bad` will be ranked higher than other frames. But if only phrases that could occur in any type of sentence are constructed, such as time expressions or personal pronouns, no strong prediction will be made.

Similarly, mutual information statistics can be used to predict likely sentence-types for a sentence if it was not determined at parse time. Similar to the previous case, the input nodes are non-terminal symbols from the parsing grammar. In this case, however, the output nodes are sentence-types. In cases where the fragments built have portions more associated with one sentence type than another, a strong ranking on sentence-types will be generated. For example, in the scheduling domain, sentences with "you" in them tend to be interrogative where sentences with "I" tend to be declarative. When the mutual information statistics are trained, regularities like this are learned automatically.

Finally, mutual information statistics can be used for ranking repair triples, where each triple is made up of (1) a slot associated with the top level semantic frame (2) one of the chunks, and (3) a semantic frame associated with the chunk. This third piece of information is completely determined by the choice in (2) if the chunk matches a type in the meaning representation specification. Otherwise it is whatever type is most likely given the parse of the chunk and likely fillers for the slot selected in (1). The process by which this occurs is described below. It involves several mutual information networks. One ranks slots according to how likely they are to be filled. For example, the `who` and `when` slots which are very common among top level frames, are very often filled, where `purpose` slots rarely are. A similar network to the one which was used to select a top level frame can be used for assigning a type to a fragment which does not match anything in the interlingua representation. Another network ranks the set of possible fillers for each slot according to how frequently they occurred relative to one another as fillers of the associated slot in the training data.

These mutual information statistics are trained over a corpus of sentences which parse correctly, making it possible to compute the correspondences between non-terminal symbols in the parsing grammar in partial interlingua structures built by unification at parse time in addition to the other types of correspondences which could be computed just from the interlingua structures themselves without the parse.

4.3 Eight Alternative Repair Strategies

Because IRLH only asks questions about local repair hypotheses, it requires built in strategies for determining which questions to ask first, etc. It has eight different strategies

Strategy	**Question 1** *Top Level Semantic Frame*	**Question 2** *Build Constituents*	**Question 3** *Search Process*
1	T	T	B
2	T	T	T
3	T	B	B
4	T	B	T
5	B	T	B
6	B	T	T
7	B	B	B
8	B	B	T

Table 4.1: **The Three Questions**

which are generated by all possible ways of selecting either top-down or bottom-up as the answer to three questions. The answers to these three questions specify what makes each strategy different from each other strategy. They are summarized in Table 4.1 and described in detail below.

The first question is, "How will the top level semantic frame be selected?". The top-down approach is to keep the largest chunk returned by the parser as the top level structure, thereby accepting the top level frame in this chunk as representing the gist of the meaning of the sentence. Strategies 1 through 4 use the top-down approach. The bottom-up approach is to assume that the partial analysis returned by the parser is merely a portion of the meaning of the sentence which should fit into a slot inside of some other top level semantic frame. This is the case in the example in Figure 4.1. Strategies 5 through 8 use the bottom-up approach. If bottom-up is selected, a new top level semantic frame is chosen by taking the set of all parser non-terminal symbols in the tree structure for the partial analysis and from each skipped segment and computing the mutual information between that set and each meaning representation specification type. This gives it a ranked set of possible types for the top level frame. The meaning representation specification rule for the selected type would then become the template for fitting in the information extracted from the partial analysis as well as from the skipped portions of the utterance. See Figure 4.2. If a new top-level frame was selected, then a new sentence type must also be selected. Similar to selecting a top level frame, it computes the mutual information between the same set of parser non-terminal symbols and the set of sentence types.

The second question is, "How will constituents be built?". The top-down approach is to assume that a meaningful constituent to insert into the current meaning representation

structure for the sentence can be found by simply looking at the total set of chunks and portions of those chunks. See Figure 4.3 for a demonstration of how chunks are built with the top-down approach. Under **Available Chunks:** we see the original set of chunks pulled out of the parser's chart. Under **Constituents:** we see the full set of chunks, including both the original chunks plus any partial chunks constructed. Strategies 1, 2, 5, and 6 use the top-down approach. The bottom-up approach is to assume that a meaningful chunk can be constructed by combining chunks into larger chunks which incorporate their meaning. The process of generating predictions about how to combine chunks into larger chunks is similar to selecting a top-level frame from the utterance except that only the parser non-terminal symbols for the segments in question are used to make the computation. Strategies 3, 4, 7, and 8 use the bottom-up approach.

The third question is, "How will the search process be driven?". The bottom-up approach is to generate predictions of where to insert chunks by looking at the chunks themselves and determining where in the meaning representation structure they might fit in. Strategies 1, 3, 5, and 7 use the bottom-up approach. See Figure 4.4.

The top-down approach is to look at the meaning representation structure, determine what slot is likely to be filled in, and look for a chunk which might fill that slot. Strategies 2, 4, 6, and 8 use the top-down approach. See Figure 4.5.

4.4 Choosing a Strategy with the Meta-Strategy

The difference between these strategies is primarily in the ordering of hypotheses. But there is also some difference in the breadth of the search space. The bottom-up approach to selecting a chunk to insert in the top level structure will only generate hypotheses about chunks that the parser built. And if there is some doubt about what the type of a chunk is, only a finite number of possibilities will be tested, and none of these may match something which can be inserted into one of the available slots. The top-down approach generates its predictions based on what is likely to fit into available slots in the current meaning representation structure. It first tries to find a likely filler which matches a chunk which has a definite type, but in the absence of this eventuality, it will assume that a chunk with no specific type is whatever type it determines can fit into a slot. And if the user confirms that this slot should be filled with this type, it will learn the mapping between the symbols in that chunk and that type. Learning new words is more likely to occur with the top-down approach than with the bottom-up approach.

The meta-strategy answers these questions, selecting the strategy to employ at a given time. Once a strategy is selected, it continues until it either makes a repair or cannot

generate anymore questions. Also, once the first question is answered, it is never asked again since once the top level frame is confirmed, it can be depended upon to be correct.

The meta-strategy attempts to answer the first question at the beginning of the search process. If the whole input utterance parses or the parse quality indicated by the parser is good and the top level frame selected as most likely by the mutual information nets matches the one chosen by the parser, it assumes it should take the top-down approach. If the parse quality is bad, it assumes it should select a new top level frame, but it does not remove the current top level frame from its list of possible top level frames. In all other cases, it confirms with the user whether the top level frame selected by the parser is the correct one and if it is not, then it proceeds through its list of hypotheses until it locates the correct top level frame.

Currently, the meta strategy always answers the second question the same way. Preliminary results indicated that in the great majority of cases, the repair module was more effective when it took the top down approach. It is most often the case that the chunks which are needed can be located within the structures of the chunks returned by the parser without combining them. And even when it is the case that chunks should be combined in order to form a chunk which fits into the current meaning representation structure, the same effect can be generated by mapping the top level structure of the would be combined chunk onto an available chunk with an uncertain type and then inserting the would be constituent chunks into this hypothesized chunk later. Preliminary tests indicated that the option of combining chunks only yielded an increase in accuracy in about 1% of the 129 cases tested. Nevertheless, it would be ideal for the meta strategy to sense when it is likely to be useful to take this approach, no matter how infrequent.

The third question is answered by taking the bottom-up approach early, considering only chunks with a definite type and then using a top down approach for the duration of the repair process for the current meaning representation structure.

The final task of the meta strategy is for it to decide when to stop asking questions. Currently it does this when there are no open slots or it has asked some arbitrary maximum number of questions. It was discovered in the evaluations of this approach that the repair module asks primarily useful questions (yielding an increase in accuracy) early (within the first 5 or 10 questions) and then proceeds to ask a lot of irrelevant questions. But no optimal maximum number of questions has been determined. If the number of questions is too small, it will not be able to learn some new input patterns and sometimes fails to recover information it would have been able to recover had it been allowed to ask a few

more questions. But if the number is too large, it is unnecessarily annoying for the user, particularly in cases where the important information was recovered early in the process.

Speaker's Utterance: *Tuesday afternoon the ninth would be okay for me though.*

Speech Hypothesis From the Recognizer: Tuesday afternoon the ninth be okay for me that.

Partial Analysis:

```
((sentence-type *fragment)
 (when ((frame *simple-time)
        (time-of-day afternoon)
        (day-of-week Tuesday)
        (day 9)))
```

Paraphrase of partial analysis: Tuesday afternoon the ninth

Skipped Portions:

1. ((value be))
2. ((frame *free) (who ((frame *i))) (good-bad +))
3. ((frame *that))

Figure 4.1: **Sample Partial Parse**

Question: How will the top level semantic frame be selected?

Answer: Try Bottom-Up.

Hypothesis: (top-level-frame ((frame-name *free)))

Question: Is your sentence mainly about someone being free?

User Response: *Yes.*

New Current Meaning Representation Structure:

((frame *free))

Skipped Portions:

1. ((value be))
2. ((frame *free) (who ((frame *i))) (good-bad +))
3. ((frame *that))
4. ((frame *simple-time) (time-of-day afternoon) (day-of-week Tuesday) (day 9))

Figure 4.2: **The First Question**

Question: How will constituents be built?

Answer: Try Top-Down.

Available Chunks:

1. ((value be))
2. ((frame *free) (who ((frame *i))) (good-bad +))
3. ((frame *that))
4. ((frame *simple-time) (time-of-day afternoon) (day-of-week Tuesday) (day 9))

Constituents:

1. ((frame *simple-time) (time-of-day afternoon) (day-of-week Tuesday) (day 9))
2. ((frame *free) (who ((frame *i))) (good-bad +))
3. ((frame *i))
4. ((frame *that))
5. ((value be))

Figure 4.3: **The Second Question**

Question: How will the search process be driven?

Answer: Try Bottom-Up.

Current Constituent:

```
((frame *simple-time)
 (time-of-day afternoon)
 (day-of-week Tuesday)
 (day 9)))
```

Hypothesis:

```
(frame-slot ((frame-name *free)
             (when ((frame *simple-time)
                    (time-of-day afternoon)
                    (day-of-week Tuesday)
                    (day 9)))))
```

Question: Is Tuesday afternoon the ninth the time of being free in your sentence?

User Response: *Yes.*

New Current Meaning Representation Structure:

```
((sentence-type *state)
 (frame *free)
 (when ((frame *simple-time)
        (time-of-day afternoon)
        (day-of-week Tuesday)
        (day 9))))
```

Figure 4.4: **The Third Question - Part 1**

Question: How will the search process be driven?

Answer: Try Top-Down.

Current Slot: who

Hypothesis: (frame-slot ((frame-name *free) (who ((frame *i)))))

Question: Is it "I" who is being free in your sentence?

User Response: *Yes.*

New Current Meaning Representation Structure:

```
((sentence-type *state)
 (frame *free)
 (who ((frame *i)))
 (when ((frame *simple-time)
        (time-of-day afternoon)
        (day-of-week Tuesday)
        (day 9))))
```

Figure 4.5: **The Third Question - Part 2**

4.5 Interaction

The sample interaction in Figure 4.6 was collected during system testing when a user unfamiliar with the system was given the task of communicating to the system that he would not be available for a meeting the following week. The parser was unable to parse most of his utterance. The parse before repair is displayed in Figure 4.7. The portion of his utterance which the parser returned an analysis for completely misses most of the content of the user's utterance. Through a series of yes/no questions, the repair module was able to extract the essential meaning of the user's utterance from the partial parse. The purpose of the first three questions in the sample interaction is for the repair module to determine what the top level semantic frame of the meaning representation of the sentence should be. With the next two questions the repair module determines the sentence type of the sentence. The last four questions are for determining what the slot fillers for the representation should be. The resulting meaning representation hypothesis can be found in Figure 4.7.

Because IRLH makes hypotheses about local repairs, the questions generated focus on local information in the meaning representation of the sentence. For instance, rather than confirm global meaning representations as in , "Did you mean to say X?", it confirms local information as in, "Is two o'clock the time of being busy in your sentence?" which confirms that the representation for "two o'clock" should be inserted into the *when* slot in the `*busy` frame. Because the repair module generates local repair hypotheses, with each decision building upon the result of the last successful hypothesis, through trial and error it is forced to ask a large number of tedious questions. If it had access to more global information, it could have framed its question in a more natural way such as "Is it at two o'clock that you are going to be busy?" This would have been far more natural sounding and thus easier for the speaker to understand.

SYSTEM: Please enter a sentence (or quit):
USER: I am afraid I will be attending the funeral of my precious canary next week.

SYSTEM: Is your sentence mainly about an interjection?
USER: No.

SYSTEM: Is your sentence mainly about being available?
USER: No.

SYSTEM: Is your sentence mainly about someone being busy?
USER: Yes.

SYSTEM: Is your sentence a short but complete sentence?
USER: No.

SYSTEM: Is your sentence a statement?
USER: Yes.

SYSTEM: Is the next week in focus in your expression of someone being busy?
USER: Yes.

SYSTEM: Is it I who is being busy in your sentence?
USER: Yes.

SYSTEM: Is a class the reason for being busy in your sentence?
USER: No.

SYSTEM: Is being available the reason for being busy in your sentence?
USER: No.

Figure 4.6: **Example Interaction**

Parse before repair:

```
((sentence-type *fixed-expression)
 (type ((adverb unfortunately)
        (frame *adverb)))
 (frame *interject))
```

Paraphrase:

"Unfortunately"

Parse after repair:

```
((sentence-type *state)
 (frame *busy)
 (who ((frame *i)))
 (topic((specifier (*multiple* definite next))
        (frame *special-time)
        (name week))))
```

Paraphrase:

"The next week I don't have time."

Figure 4.7: **Meaning representation structures for example before and after repair**

4.6 Quantitative Evaluation

In this section I will discuss the results obtained with IRLH. In the following section, I will then discuss the primary shortcomings of this approach. In subsequent chapters I will demonstrate that these shortcomings are overcome in the ROSE repair approach.

The repair module was tested on two corpora with 129 sentences each. One corpus contains 129 transcribed sentences from spontaneous scheduling dialogues as described above. The other corpus contains the output from the speech recognizer from reading the transcribed sentences. So it contains all of the difficulties of the transcribed corpus in addition to speech-recognition errors. The performance of the repair module was compared with a baseline process on an additional corpus of 113 sentences. These results indicate that the performance of the repair module improves as the number of questions increases, that its performance generalizes to different data sets, and that the meta strategy consistently achieves better performance than any of the single strategies as well as the baseline comparison process.

The repair module was evaluated automatically with no human intervention whatsoever. Prior to the evaluation, a human coder hand coded ideal interlingua representations for each of the 129 sentences. The human-computer interaction component of the repair module was simulated by having the computer match each proposed repair against the ideal interlingua structure for the corresponding sentence to test whether it would make the current "in-progress" version of the interlingua representation internal to the repair module more like the ideal one. Each of these tests counted as one question. If the match indicated that the proposed repair was a good one, a "yes" answer was assumed, otherwise a "no" answer was assumed. If the answer was "yes", the repair module made the hypothesized repair. After each question, the possibly updated internal interlingua representation was matched with the ideal one in order to calculate a point value. The matching process was carried out recursively, first comparing the top level frame, and if it was the same, for each slot, comparing the fillers in the ideal structure with the corresponding ones in the internal structure. For each matching frame or atom, one point was assigned. The total possible was computed by counting the total number of frames and atoms in the ideal representation. From this, a percentage correct could be calculated at each stage of the repair process in order to track the improvement of the quality of the internal representation per question asked.

Figure 4.8 displays the relative performance of the eight strategies compared to the meta strategy on speech data. Figure 4.9 displays the relative performance of the strategies

on the transcribed data. In both of these figures, strategies are paired according to their answers to the second and third questions. This is because the answer to the first question only affects the early portion of the interaction. Apart from that, strategies which share the same answers to the second and third questions are identical. Note that these diagrams demonstrate that the meta strategy consistently achieves a better performance than any of the single strategies and that its performance improves as the number of questions allowed increases.

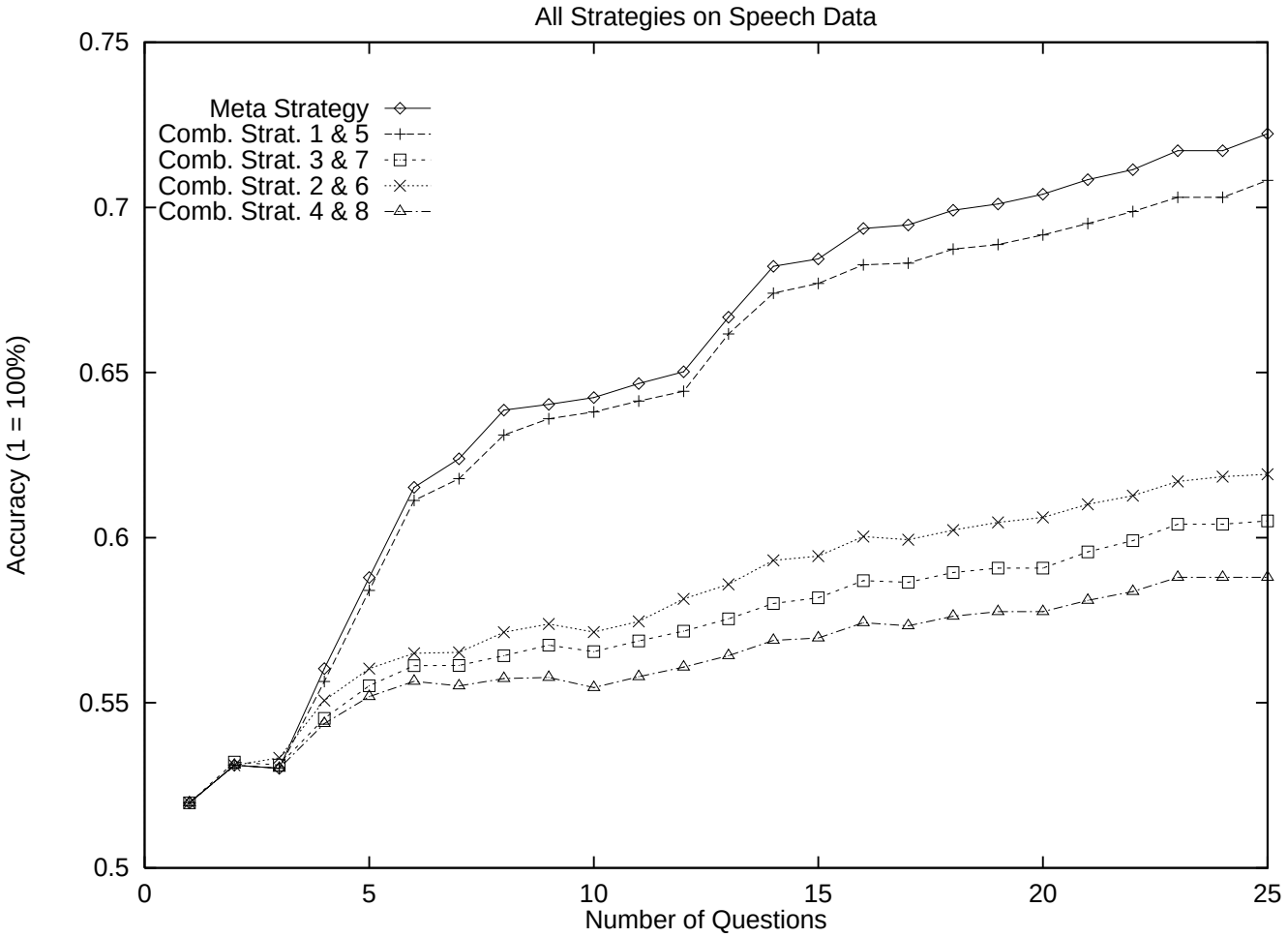

Figure 4.8: **Results from All Strategies on Speech Data**

Given a maximum of 10 questions to ask the user, this repair module can raise the accuracy of the parser from 52% to 64% on speech data and from 68% to 78% on transcribed data. Given a maximum of 25 questions, it can raise the accuracy to 72% on speech-data and 86% on transcribed data.

The baseline for comparison was a process consisting of building the correct interlingua structure from a Huffman Coded version of the interlingua specification where each

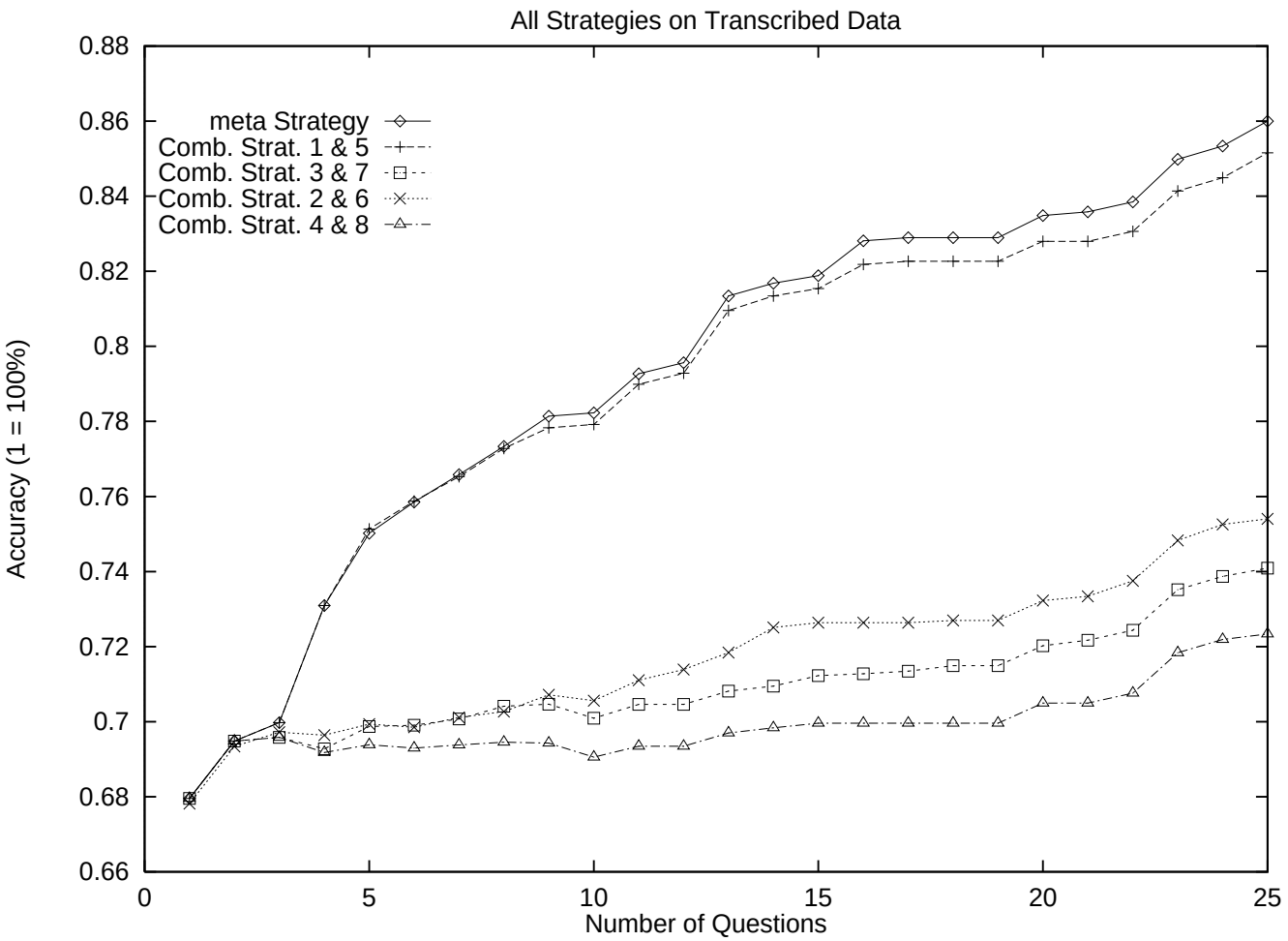

Figure 4.9: **Results from All Strategies on Transcribed Data**

node in the Huffman tree counted as one question. A Huffman tree was constructed for each type in the interlingua specification. The comparison process began by trying to select the top level semantic frame by traversing the Huffman tree for top level frames. It then continued the process recursively constructing fillers for each of the slots using the Huffman tree for the type corresponding to the slot. This process corresponds to an extreme version of the repair process not using any information from the parser whatsoever. The results of comparing the meta-strategy to the comparison process on the additional data set can be found in Figure 4.10. Notice that the meta-strategy is consistently better than the comparison process.

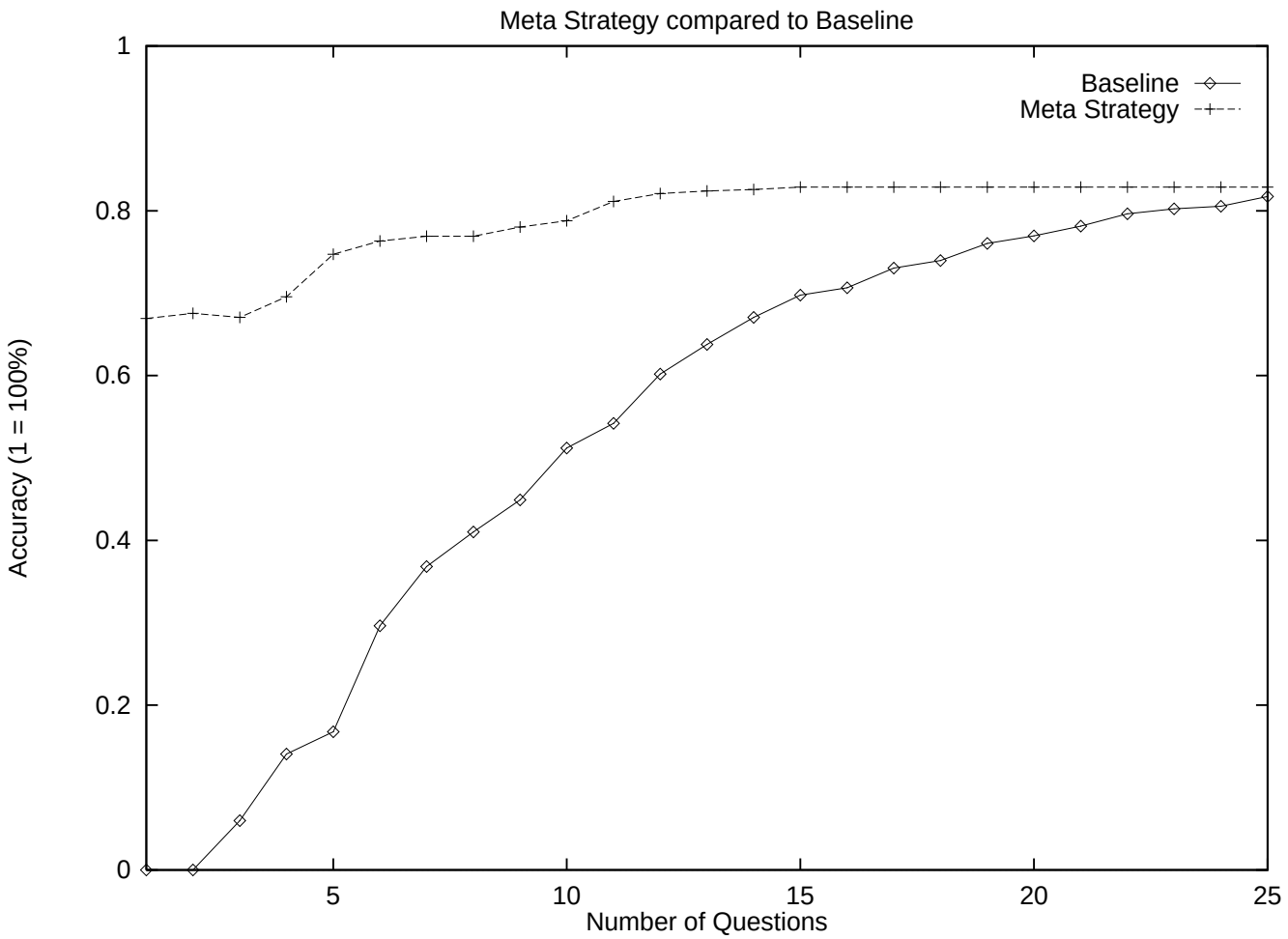

Figure 4.10: **Results from Meta Strategy compared to Baseline**

4.7 Shortcomings of the Incremental Repair Approach

In this section I qualitatively evaluate the IRLH, pointing out its primary shortcomings.

The ability of IRLH to make repairs of various sorts is determined by two parameters which guide the search process. The first is the maximum number of questions it was allowed to ask, and the second is what is called the bandwidth. Bandwidth is used in two different ways in IRLH. It specifies how long the list of likely types is assigned to a chunk with an uncertain type and how many potential slots are considered when a top-down approach is selected for driving the search process. This parameter is set for practical reasons so that the repair module spends the majority of its time considering hypotheses which are truly likely. If the maximum number of questions were infinite, it would not be important to have such a parameter. In practice, with a small finite number of questions, it is important.

It forces the repair module to sample a larger area within its search space more shallowly instead of looking at a very small area in depth.

The problem with setting a bandwidth is that it makes certain potential repairs impossible and affects the repair module's ability to learn. If the correct hypothesis in a particular case is extremely unlikely according to the statistical model, it will not most likely be ranked within the bandwidth, so it will never be considered. The maximum number of questions is similar. In some cases, where the correct hypothesis is extremely unlikely, the repair module will not be able to hone in on it within its maximum number of questions. In the case that the repair module is faced with the task of mapping new words onto a less frequently encountered concept, if it never manages to get to the correct hypotheses within its maximum number of questions, it will not be able to learn that mapping.

With an infinite bandwidth and infinite number of questions, the repair module would have the ability to make any repair. This is intuitive because even with no information from the parse, by cycling through the possible top level frames and sentence types, it would eventually arrive at the correct combination. From there it knows from the interlingua specification what the possible fillers for each of the slots are, and by cycling through all of those possibilities, it would eventually arrive at the correct interpretation.

It is not practical to allow the repair module to operate this way, however. First of all, it is much more practical to make use of information from the parse where it is available, see Figure 4.10. In some cases, this means relying upon the parser to produce reliable results. If the repair module confirmed every piece of information in the feature-structure returned by the parser, it would ask far too many useless questions. On the other hand, if it took an all or nothing approach, it might throw out potentially useful portions of the analysis from the parser. Currently, it only checks to see if the top level semantic frame returned by the parser is correct. If it is, it keeps the whole partial analysis returned by the parser, otherwise it starts from scratch, using portions of the parser's partial analysis wherever possible. If it keeps the whole partial analysis, however, it may retain portions of it which are not correct. With a more reliable confidence model, this compromise would not be necessary.

As mentioned above, the primary problem with the IRLH approach is that each of its hypotheses are generated locally, with each decision building upon the result of the last successful hypothesis. So through trial and error, the repair module is forced to ask a large number of tedious questions. In this section I discussed the role played by the two parameters, maximum number of questions and bandwidth, in perpetuating this problem. In ROSE's two stage repair process, the bandwidth is replaced with the application of the

genetic programming paradigm which guides the search. The maximum number of questions parameter is replaced by a mechanism which calculates how many questions are necessary in order to distinguish between the set of top most likely global repair hypotheses for each specific case. It is not possible to calculate this for the IRLH approach since the repair and interaction stages are interleaved.

Chapter 5
Minimum Distance Parsing

As mentioned in the previous chapter, the goal of this dissertation research is to demonstrate that the two stage ROSE approach can achieve better results more efficiently than with either IRLH or with MDP. In the previous chapter I described an implementation of IRLH. In this chapter I focus on the MDP approach.

In this dissertation, I argue that the ROSE approach of separating the Partial Parsing and Combination steps is more efficient than placing the full burden of robustness on a single parsing algorithm, as in MDP. An analogous trade-off in human/human communication would be the "casting and combining" model versus one in which the listener attempts to construct a complete syntactic analysis for a sentence outside of his language competence. Undoubtedly, this is an unreasonably difficult task. Though humans are known to make a mental note of grammatical features which they are not able to process correctly, most of it is said to be regarded mainly as "noise" (Hatch, 1983).

5.1 High Level Description

Efforts towards robustly handling extragrammatical input have largely been in the direction of building flexible parsers, commonly Minimum Distance Parsers (Lehman, 1989; Hipp, 1992) which fit an extragrammatical sentence to the parsing grammar through a series of insertions, deletions, and transpositions. Since any string can be mapped onto any other string through a series of insertions, deletions, and transpositions, this approach makes it possible to obtain an analysis for any input sentence. The idea is that out of the various possible strings in a language, the one which is closest to the input string is more likely to be the closest analysis possible to the correct one, though in practice this is not always the case. So on the surface, the Minimum Distance Parsing approach appears to be a reasonable approach.

In practice, however, the Minimum Distance Parsing approach has only been used in very small domains. It is important to note the scale of the project which provides the context for work of this nature since flexible parsing algorithms introduce extra ambiguity to a greater or lesser extent which deems certain approaches intractable for systems of more realistic scale. For example, Lehman's core grammar, described in (Lehman, 1989), has on the order of 300 rules, and all of the inputs to her system can be assumed to be commands to a calendar program. Hipp's Circuit Fix-It Shop system, described in (Hipp, 1992), has a vocabulary of only 125 words and a grammar size of only 500 rules. In contrast, ROSE was developed in the context of a far less constrained system, a speech system with vocabulary size on the order of 3000 words, and grammar size also on the order of 1000 rules. The question is whether the less constrained Minimum Distance Parsing approach is still practical in a system of this scale or whether a more restrictive algorithm is necessary. In this chapter I argue that a more restrictive algorithm is necessary by presenting timing measurements for the two algorithms each operating with two different grammars.

5.2 Setting Up a Comparison with a More Restrictive Parsing Algorithm

An example of a more restrictive parsing algorithm is Lavie's GLR* skipping parser described in (Lavie, 1995). This parser has the ability to recover a parse for an utterance in which some subset of that utterance can be covered by the parsing grammar. It does this by skipping some subset of the words in the ungrammatical sentence in order to obtain an analysis for chunks of the input sentence, if not the complete sentence, and to return the analysis for the largest chunk which it can parse, or, alternatively, a set of chunks covering the entire input sentence. The weakness of this method is that part of the original meaning of the utterance may be thrown away with the portion(s) of the utterance which are skipped if only the analysis for the largest chunk is returned. In subsequent chapters I describe how ROSE overcomes this weakness without losing the computational advantage which is obtained by using a more restrictive parsing algorithm.

In order to do a fair comparison between the two parsing approaches, a version of the Minimum Distance Parsing algorithm was implemented as an extension to Lavie's GLR* parser called LR MDP. This way the MDP algorithm can operate seemlessly within the same system that the GLR* parser is currently operating. All factors besides the parsing algorithm can be held constant for comparison purposes. In order to demonstrate the dramatic affect increasing the complexity of the parser's flexibility has on the speed

of the parsing algorithm, the comparison between the two algorithms is conducted both with the same grammar used in (Hipp, 1992) as well as with the larger scheduling domain grammar. The grammar from (Hipp, 1992) was converted rule for rule from Hipp's STD formalism to GLR*'s unification based formalism.

The idea behind the Minimum Distance Parsing approach is to return an analysis for the sentence closest to the input sentence which can be parsed with the grammar. This is made possible by simulating insertion and deletion operations on the input sentence. The GLR* parsing algorithm is more restrictive than the minimum distance parsing approach in that it allows deletions but not insertions. Below I describe how the original GLR* parsing algorithm was modified in order to extend its power to that of the Minimum Distance Parsing approach.

5.3 Extensions for Minimum Distance Parsing in the GLR* Framework

Most Minimum Distance Parsers are based on chart parsing (Early, 1970; Kay, 1980). The implementation described here was developed in the LR parsing framework. Later this approach will be demonstrated to be at least as efficient as the analogous extension to the chart parsing approach.

The primary difference between the standard chart parsing approach and the LR parsing approach is that rather than making use of the grammar directly, an LR parser makes use of a finite-state push down automaton which is a compiled version of the grammar. The grammar rules are implicit in the state machine. Each state represents a set of possible completions for the part of the input string which has been processed so far. Associated with each state is a set of actions which are performed on a Graph Structured Stack (GSS) which keeps track of partial parses as they are built, similar to the chart in the chart parsing approach. On the GSS, *symbol* nodes and *state* nodes alternate, representing partial parses and corresponding states in the compiled grammar state table. Each state node on the top of the GSS represents an active state where the next input token can be pushed. Skipping words in the input string is accomplished by pushing input tokens onto previous state nodes on the GSS. A more in-depth discussion of the GLR* parser can be found in Appendix A. It is recommended that readers familiarize themselves with the material found in that appendix before attempting to understand the material found in the remainder of this section.

In the original Minimum Distance Parsing approach, edges are inserted into a chart, corresponding to matches or partial matches of grammar rules. This is not possible

straightforwardly in the LR parsing paradigm since the rules themselves are not explicitly made available to the parsing algorithm. So rather than making use of the grammar directly, the LR approach makes use of the compiled state machine described above.

In the GSS, state nodes and symbol nodes alternate. When a symbol node is pushed onto a state node, the new state is calculated by making reference to the compiled grammar which is a finite-state PDA. A new state node corresponding to the new state is then pushed on top of the symbol node. Each symbol node represents an element in a grammar rule, and therefore a portion of the input sentence. If the PDA contained empty transitions, making it possible to move to a new state without first pushing the requisite symbol node, this would make it possible to simulate insertions in this framework. Insertions, in this context, refers to inserting non-terminal symbols into the input stream, allowing rules to fire which otherwise would not be able to because of a missing portion of the input.

Inserting into the input stream is equivalent to skipping portions of grammar rules when they are matched against the input. Where a portion of a grammar rule might have been skipped in the original Minimum Distance Parsing approach, one or more states are skipped over in the compiled grammar state machine in the LR version. Since the state machine lists which state you can get to by pushing a symbol node of a particular category in a particular state, you can simulate skipping a non-terminal in a rule by jumping to the new state without first matching a portion of the input to create a symbol node of that category. In order to make the stack look as if a symbol node of that category had actually been built, a dummy symbol node of the corresponding category is pushed onto the GSS with an empty feature structure. On top of this is then pushed a state node corresponding to the new state. In this way, reductions can be performed on the GSS the same way for partial rule matches as with complete rule matches since the stack looks the same in both cases. A penalty equivalent to the minimum number of words needed to generate a symbol of the corresponding category is assigned to every dummy symbol node.

For an example, take a look at the grammar in Figure 5.1, the compiled grammar table in Figure 5.2, and the example derivation in Figure 5.3. Notice that the sentence in the example, "Sees the cat." is not a grammatical sentence according to the grammar. However, if an *NP* could be inserted at the beginning of the sentence, an analysis could be found. The grammar table indicates that when an *NP* symbol node is pushed in state 0, the parser's initial state, the parser then moves into state 3. So, in order to insert an *NP* at the beginning of the sentence, a dummy *NP* symbol node is inserted on top of the state node corresponding to the initial state. A state node corresponding to state 3 is then pushed,

and the derivation can proceed from there as if an NP, such as "The dog" had actually been present in the input stream.

(1) S --> NP VP
(2) NP --> det n
(3) VP --> v NP

Figure 5.1: Simple Example Grammar

	Reduce	Shift				Goto		
State		det	n	v	$	NP	VP	S
0		sh1				3		6
1			sh2					
2	r2							
3				sh4			5	
4		sh1				7		
5	r1							
6					acc			
7	r3							

Figure 5.2: Compiled Grammar Table

The *Goto* portion of the compiled grammar table normally indicates which state the parser moves into after inserting a symbol node of a particular category which results from performing a reduction. The same portion of the table is now used to indicate which dummy nodes can be inserted into each state and which states are then reachable as a result. According to the grammar table in Figure 5.2, from state 0 it is possible to insert a dummy *NP* node and then move into state 3 or a dummy *S* node and then move into state 6. From state 3 it is possible to insert a dummy *VP* node and then move into state 5. Finally, from state 4 it is possible to insert an *NP* node and then move into state 7. The GSS makes it possible to pursue multiple options in parallel. The set of state nodes at the top of the GSS after each word is processed is a beam of alternative analyses.

Of course, more than one state can be skipped at a time, just as in some cases more than one non-terminal in a rule may be skipped in the straightforward MDP approach. With a non-trivial grammar, the total set of possible sequences of non-terminals which can be skipped in any particular state is very large. If dummy symbol nodes corresponding to all of the ways of doing this were inserted into the GSS, the beam would branch far beyond

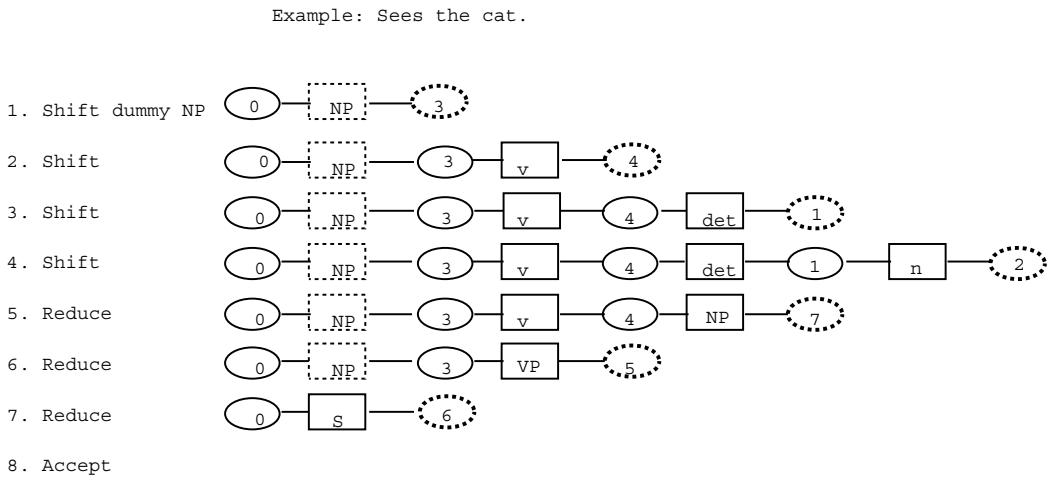

Figure 5.3: **Stack at Each Derivation Step**

a manageable level. In order to limit this trouble without sacrificing any power, exactly those dummy symbol nodes which are useful for continuing the parse are inserted. This set is computed by first computing the set of states reachable by inserting one or more dummy symbol nodes from the current set of active states. If there is more than one way to reach a particular state, only the cheapest way of getting there is kept, where cheapest is computed in terms of the penalty assigned. Of these states, only a subset of them will be able to have the next input token pushed on them, as determined by looking in the *Shift* portion of the grammar table. For each of these, dummy symbol nodes and corresponding state nodes are inserted into the GSS in order to make those states available for pushing the new token onto. To speed up the computation at run time in the actual implementation, a transition table is precompiled which lists the cheapest transitions from each state in the compiled grammar state machine to all of the states reachable from that state by inserting one or more dummy symbol nodes.

Note that because it is impossible for the parser to know whether the unparsed portion of the input will be grammatical, the MDP parser explores every option for inserting dummy nodes at every stage in the parse even when the input sentence is totally grammatical. It explores the option of inserting dummy nodes in order to make available at the top of the GSS every state reachable from states already at the top by inserting some sequence of dummy nodes. As mentioned above, for efficiency, dummy nodes are only actually inserted to reach states which the next input token can be pushed onto.

A problem occurs when the dummy non-terminal which needs to be inserted in order to make a parse possible occurs at the end of a rule. This is impossible with the grammar in Figure 5.1. However, suppose reduction 2, `NP --> det n` were replaced with

`NP --> det n PP`. Suppose also that a determiner and noun have been processed, and the next token on the input stream is a verb. Even if a dummy PP node is inserted to make it possible for this rule to fire, the next input token would not be able to be pushed onto the parse stack until either a dummy NP node is pushed onto the initial state, making it superfluous have inserted the dummy PP node, or reduction 2 is applied.

In cases like this, the state which the next input token needs to be pushed onto is not the state which would be reached by inserting one or more dummy non-terminals only, but the state that would be reached by both inserting one of more dummy non-terminals and then doing a reduction, inserting more dummy non-terminals, and so on. In the same way that it is not practical to push dummy node vertices corresponding to every transition which can be made from every state on the GSS, it is not practical, or even advantageous, to perform every reduction associated with each state which will be potentially inserted in order to calculate the total set of reachable states. For example, this would lead to endless loops in the case of recursive rules. But more importantly, it would lead to results which could be accomplished in a more efficient way in some cases. The point of a Minimum Distance Parser is to compute an analysis with the fewest number of insertions and deletions. So if there is more than one way to accomplish a particular result, it is not necessary to accomplish it in every possible way. Only the cheapest way will be part of the final result. So if the more expensive ways can be avoided, it is worthwhile to do this.

For example, take a look at the partial derivation in Figure 5.4. The first three words of the sentence "The dog sees the cat." have been processed, leaving the parser in state 4. From state 4, it is possible to insert a dummy *NP* node, moving the parser into state 7. And from there, it is possible to perform a reduction by applying rule 3, `VP --> v NP`. The question is, is it worthwhile to perform this reduction?

To answer this question it is necessary to determine how expensive it is to perform the reduction versus how expensive it is not to perform the reduction. If the reduction is performed, the total penalty assigned to the resulting *VP* symbol node will be the penalty from inserting the dummy *NP* symbol node, in other words 2. On the other hand, if this reduction is not performed, the only alternative for getting to state 5 from the state 3 node is to insert a dummy *VP* node there. But in this case, there would be both a penalty from inserting a dummy *VP* node, in other words 5, and the penalty for skipping over the *v* symbol node which had already been pushed, which is 1. The total penalty to get to state 5 from not doing the reduction, 6, comes out bigger than the penalty after doing the reduction. In this case, doing the reduction is worthwhile. Since doing the reduction increases the number of alternative analyses on the frontier of the GSS, the reduction is

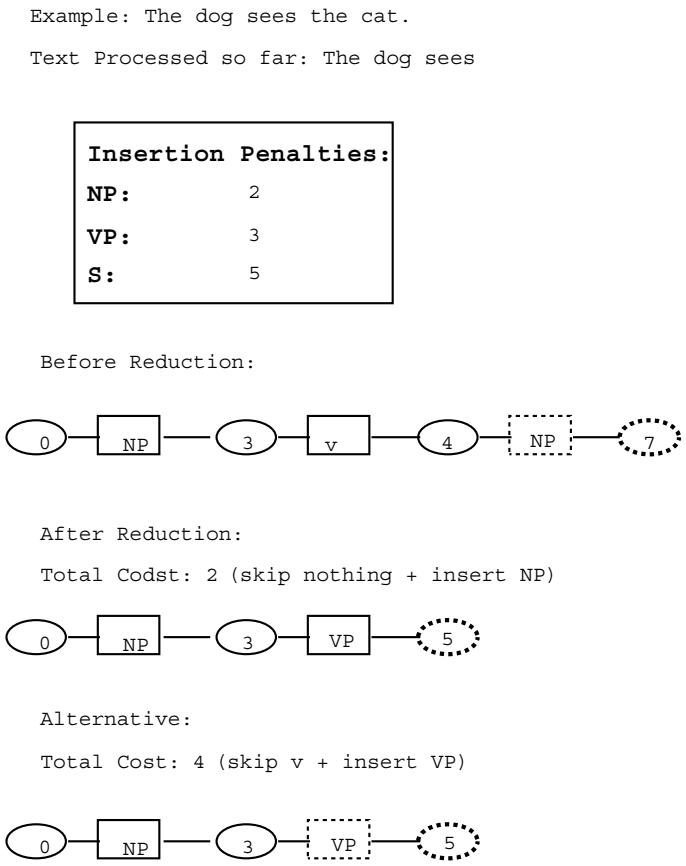

Figure 5.4: **Promising Reduction Illustration**

only considered worthwhile in the case that it comes out strictly cheaper than not doing it[1].

Trade offs like this are made before each new input token is pushed in order to compute the total set of "promising reductions". These are the reductions from states reachable from skipping non-terminals, such as the one just described, which make it possible to get a state with a smaller penalty than by getting to the same resulting state by inserting one or more dummy symbol nodes.

5.4 Completeness of the MDP Implementation

One question which remains is whether every parse made possible by the original Minimum Distance Parsing algorithm is also possible with this new LR version. Clearly every minimum distance parse is possible if no shortcuts are taken, i.e., if every dummy node

[1]Inserting a dummy symbol node also increases the number of alternative analyses on the frontier, but this will only be done in the case that the next symbol can be pushed onto it, whereas the reduction would be performed regardless.

corresponding to a non-terminal which can be skipped is inserted and if every reduction possible from any state reachable by inserting dummy nodes is performed. So the real question is whether the shortcuts described above eliminate any parse which might be desired. The key concept is that in the LR parsing algorithm, the only factor which determines how a parse can proceed is the set of states which are left on the the frontier of the GSS after the current word input token has been processed.

States are reached either by shifting new node vertices or by doing reductions. The dummy nodes which are inserted before pushing the current input token are exactly those which lead to states that are reachable from the states on the frontier of the GSS where you can push the current input token. If there are any other dummy nodes which could have potentially been inserted, they would be ones leading to states where the current input item could not be pushed. In other words, it would be useless to insert them. Therefore, this shortcut does not prevent any parse from completing.

In the case of promising reductions, the question remains if not performing a reduction deemed not promising would prevent any desirable parse. But the trade-offs in determining whether a reduction is promising or not are in terms of alternative ways of reaching the same state. Therefore, whichever alternative is selected, the same set of subsequent actions are possible. The one which is selected is the one which is computed to be cheapest. So, the only reductions which are eliminated are those which achieve the same ending state as some other course of action, but in a less efficient way. Since the purpose of the minimum distance parser is to return the analysis with the fewest number of insertions and deletions, this is desirable. In this way, it is possible to simulate the complete Minimum Distance Parsing algorithm by extending the GLR* parser as described above. Of course, there could be cases where a cheaper reduction might fail because of unification where a more expensive one would not. In this case, this shortcut would hinder the parser's performance. This is always a danger with pruning techniques in unification based parsers. Even in this case, however, the parser can still recover by inserting a dummy non-terminal later, making it possible to construct an analysis of the rest of the sentence.

5.5 Efficiency of the MDP Implementation

The final question which remains is whether the LR version of the minimum distance algorithm performs as efficiently as the original Minimum Distance Parsing algorithm. In general, LR parsers are regarded as more efficient than the bottom-up chart parser algorithm because of the way they use the left context in order to limit the set of rules which are applied at any given point. I argue that the only case in which this LR MDP parser

can be less efficient than another implementation of the MDP algorithm is if the other MDP version takes shortcuts which eliminate possible minimum distance parses from being considered.

If it were possible to take short-cuts which the LR MDP parser doesn't take, one possibility would be to be more selective in choosing "promising reductions". Since states can be reached either by inserting node vertices or by reductions, if a reduction is deemed not promising, the state that would have been reached by performing the reduction can still be reached by inserting a dummy node vertex. It would be inserted in the same place in the graph structured stack where the non-terminal resulting from the reduction would have been inserted. But the current method of selecting "promising reductions" chooses the ones which make it possible to reach a state more cheaply than by inserting a dummy non-terminal. So being more restrictive would undermine the goal of MDP which is to find the parse with the *minimum* cost. The only other way to handle this trade-off more efficiently would be to not reach the full set of reachable states that the current input token could be pushed onto. By eliminating the possibility of pushing the current input token onto those states, the set of possible subsequent actions leading to complete analyses of the input are reduced, again making it impossible to guarantee that the final result will be the minimum distance parse.

The same applies in the case of inserting dummy node vertices in order to push the current input symbol. Here it is also the case that fewer resulting states would be left on the frontier of the GSS, and therefore, fewer possible courses of future actions would be available for completing the parse. Because it is always possible to insert or delete on subsequent steps, it will always be possible to find an analysis for the complete sentence in either case. But eliminating certain courses of future action based on local information makes it impossible to guarantee that the parse which it will end up with will be the real minimum distance parse.

Therefore, the LR MDP algorithm described here can be argued to produce the minimum distance parse for any sentence in the most efficient way possible without sacrificing the ability to find the real minimum distance parse. In the next section I will demonstrate that regardless of the fact that this version of MDP arguably achieves its objective as efficiently as possible with this approach, it is orders of magnitude less efficient than the more restrictive GLR* parser.

5.6 Complexity Comparison of GLR, GLR*, and LR MDP

Because both GLR* and LR MDP are built on top of the original GLR parsing algorithm, the complexity of these two derivative algorithms is based on the complexity of the original GLR algorithm. Complexity of a parsing algorithm can be computed with respect to two different dimensions. The first dimension is the length of the input sequence, and the second dimension is the size of the grammar. In this section, the GLR, GLR*, and LR MDP algorithms are compared with respect to these two factors. It will be shown that although the runtime performance of the three algorithms varies widely in a large system, the complexity for all three algorithms is the same.

5.6.1 Input Length Complexity

As discussed in (Kipps, 1991), complexity of the original GLR algorithm with respect to input length is (n^{p+1}) where n is the length of the input and p is the length of the longest grammar rule. Lavie argues that this worst case complexity also holds for GLR* (Lavie, 1995). Here I argue that LR MDP also has this same worst case complexity with respect to sentence length.

In the GLR architecture, six discrete steps must be performed in order to process each input item. Here is a discussion of those six steps along with their associated complexity with respect to sentence length for the original GLR algorithm. This description is based on the one presented in (Lavie, 1995). Below is a discussion of how this analysis applies to GLR* and LR MDP.

1. **Read:** First, input word i must be read from the input. This takes at most $O(1)$ time.

2. **Distribute-Reduce:** Next, reduce operations, determined by making reference to the Reduce portion of the compiled grammar table, are distributed to the active state nodes. For each active state node, the parsing table must be accessed, and then the reduce actions are inserted into the respective state node. Since for each active state node this can be done in constant time, the complexity of this step is proportional to the number of active state nodes at step i. The number of active state nodes is bounded by a constant since there can be no more active state nodes than there are states in the compiled grammar table[2]. Therefore, this step also takes $O(1)$ time.

[2]This is because active state nodes representing the same state are packed into a single active state node during the Merge step.

3. **Reduce:** During the Reduce step, the reduction actions inserted into the active state nodes during the previous step are performed. The total number of reductions performed is the set of initial reductions plus the set of subsequent recursive reductions. Since the number of nodes feeding into a state node (the fan-in) is $O(i)$, an initial reduction of length p for a single state node is performed $O(i^p)$ times. Kipps (Kipps, 1991) argues that the number of recursive reductions is also $O(i^p)$. Therefore the complexity of this step of the parsing process is $O(i^p)$ (Kipps, 1991).

4. **Distribute-Shift:** In this step, shift actions are distributed to active state nodes, similarly to how reduce actions are distributed in step 2. Therefore, this step is performed in $O(1)$ time.

5. **Shift:** Since a single shift operation requires only constant time, performing the shift operations takes no more time than distributing them. So this step is also $O(1)$.

6. **Merge:** During the merge step, the new nodes are scanned through once to see if any of them can be merged. All of the active state nodes corresponding to the same state are merged into a single state node. Since the total number of new nodes is bounded by the number of states in the grammar table, this step also takes constant time, i.e., $O(1)$.

Since the complexity of the Reduce step dominates, the complexity of the ith stage of the parsing process is equal to the complexity of the Reduce step for that stage, i.e., $O(i^p)$. Since these six steps must be performed for each of the input items, the complexity with respect to sentence length for parsing an n item sentence is $O(n^{p+1})$.

Lavie argues that the complexity with respect to sentence length for GLR* is also $O(n^{p+1})$ (Lavie, 1995). Obviously, the complexity for reading the input is unchanged for the GLR* algorithm. Similarly, the complexity for Distribute-Reduce and Reduce are unchanged since these depend upon the number of active state nodes and the fan-in. The number of active state nodes must still be bounded by the total number of states in the compiled grammar table. And the fan-in remains the same as with GLR because of ambiguity packing. The last three stages are different for GLR*, however, since shift operations are performed for every node in the GSS rather than only active state nodes. Since the number of state nodes for each stage is bounded by a constant, the number of nodes in the whole GSS is bounded by the number of items which have been processed. Thus, it is bounded by $O(i)$. Likewise, the number of shift operations performed will be bounded by $O(i)$, and the number of items which are scanned during the Merge step will also be

bounded by $O(i)$. Since the complexity of none of these steps dominates the Reduce step, the complexity of the parsing processes as a whole remains $O(n^{p+1})$ for GLR*.

For LR MDP, an additional step is inserted in between reading the next input item and distributing the reduce actions. In this step, a table is accessed which contains the cheapest transition between each state in the compiled grammar table and each other state which is reachable by inserting non-terminals. The set of active state nodes is scanned, and the set of cheapest reachable state transitions is collected. If there is a transition to the same state by more than one of the active state nodes, only the cheapest one is retained. And only transitions to states not represented among the active state nodes are retained. Finally, only those transitions to states where the next input item can be inserted are retained. Once this final set of transitions has been determined, dummy nodes corresponding to these transitions are then inserted into the GSS. The complexity of this operation is bounded by the number of state nodes in the compiled grammar table. Therefore, complexity with respect to input length for this extra step is $O(1)$.

A similar step is inserted after the Merge step for performing promising reductions. Again, the set of active state nodes is scanned, and the set of cheapest reachable state transitions is collected. If there is a transition to the same state by more than one of the active state nodes, only the cheapest one is retained. And only transitions to states not represented among the active state nodes are retained. This time, the set of states reachable by way of these transitions is examined for reductions which make it possible to reach new states more cheaply than by inserting alone. Since the number of new reachable states is still bounded by the number of states in the grammar table, and since the number of promising reductions is bounded by the total number of possible reductions, the complexity of this step is $O(i^p)$, just like the Reduce step.

After this step, an additional Merge step is performed, again taking $O(1)$ time.

Therefore, the total complexity for LR MDP, like for GLR and GLR*, remains $O(n^{p+1})$.

For several reasons, this complexity comparison is not very enlightening in terms of the actual differences in runtime performance of the three algorithms. Lavie demonstrates empirically that regardless of the fact that GLR and GLR* have the same complexity with respect to sentence length, in practice GLR's performance is closer to linear where GLR*'s matches that of a higher polynomial as the maximum number of words skipped is increased, though the performance is shown to be far more reasonable when a beam-limit is used to control the growth of the frontier of the GSS. In the same way, though LR MDP also has the same complexity, in practice it takes orders of magnitude more time to process the same

sentences as the GLR* algorithm, even when using a beam limit to control growth of the frontier of the GSS. This will be demonstrated in the next section.

The worst case complexity is not very enlightening for a few reasons. The first reason is that the maximum size of the frontier of the GSS is taken to be a constant since it is bounded by the number of states in the compiled grammar table. But it should be noted that the number of states in a realistic sized grammar is very large, on the order of 1000 states. So knowing that the number of nodes on the frontier of the GSS is bounded by this number is not very comforting. The closer the grammar is to an LR grammar, the closer the size of the frontier of the GSS will remain to 1 with the original GLR algorithm. But when skipping is introduced, each string represents $2^n - 1$ strings. So the size of the frontier of the GSS will grow exponentially with unrestricted skipping. Once insertions are also introduced, the frontier can grow even faster since in that case all of the states which would be present with GLR* will remain as well as all of the states reachable from those states by inserting one or more non-terminals. The only active state nodes which remain on the frontier when the next input token is processed are those states which the next input token can be pushed onto. So in practice, the worst case bound on number of states on the frontier of the GSS is never reached. Nevertheless, there is a big difference in practice between GLR, GLR*, and LR MDP.

It can be shown that the maximum fan-in for GLR* and GLR are the same because of ambiguity packing. In fact, it can be shown for the worst case grammar presented in (Kipps, 1991), the GSS for GLR* after processing each input item is the same as that for GLR, but only because the additional analyses produced by GLR* are all packed into nodes already produced by GLR. Thus, since GLR* allows for a far larger number of analyses than the GLR algorithm, GLR* spends more time in ambiguity packing than GLR. No complexity analysis for ambiguity packing is presented in (Lavie, 1995).

Another major factor which affecting run-tim performance of LR MDP which does not come out in the complexity analysis is that the LR MDP algorithm must spend extra time processing each node on the GSS, determining which of the reachable states to insert into the GSS given the next input item. Even if it determines that none of these reachable states are worth inserting, it spends time for each reachable state, bounded by the total number of states in the compiled grammar table, coming to this conclusion.

Therefore, regardless of the fact that GLR, GLR*, and LR MDP have identical complexity analyses with respect to sentence length, it is not surprising that in practice their run-time performance varies so widely.

5.6.2 Grammar Complexity

A detailed analysis of GLR's grammar complexity can be found in (Johnson, 1991). There it is explained that the amount of work it takes to process one input item is proportional to the number of active state nodes in the GSS while processing that item. By describing a class of grammars G_m where the number of such state nodes is demonstrated to be an exponential function of the grammar size, the GLR algorithm's grammar complexity is shown to be exponential for this class of grammars.

The class of grammars G_m is one which generates strings of the form a^n. These grammars produces such extreme behavior since the input items do not possess any predictive power, quite unlike natural language grammars. Johnson argues that the number of states in the grammar table for this class of grammars is proportional to $\Omega(c^{\sqrt{|G_m|}})$ (Johnson, 1991). He argues further that after m input items have been processed, the number of active state nodes will also be proportional to $\Omega(c^{\sqrt{|G_m|}})$. In fact, the full set of states in the compiled grammar table will be represented. Therefore, the total number of operations required to parse a string of length n is $\Omega(nc^{\sqrt{|G_m|}})$. Since rather than only shifting onto the active state nodes the GLR* parser shifts onto every node in the GSS, GLR*'s grammar complexity is $\Omega(n^2c^{\sqrt{|G_m|}})$. I argue that LR MDP's grammar complexity is the same. The only difference between LR MDP and GLR* is that more active state nodes are made available artificially by inserting dummy nodes. But the maximum number of active state nodes is still bounded by the number of states in the compiled grammar table. In other words, it is no different for any of these three algorithms. So the only thing which is different is the added time for inserting the dummy nodes. Since as stated above, the complexity of this process is bounded by the number of states in the grammar table, the complexity of this step with respect to grammar size is linear. Since it is dominated by the complexity of the other steps, the complexity remains the same.

Though the complexity for this class of grammars is shown to be the same for MDP, it should be noted that in practice the algorithm must perform many more steps. For example, with GLR and GLR*, it takes m steps before the maximum number of states in the grammar table are represented as active state nodes. On the other hand, since almost every state is reachable by some sequence of inserting dummy nodes, and since in this class of grammars, the current input item can be pushed onto an active state node for *any* state, almost the full set of states will be represented as active state nodes after the *first* input item has been processed. And though no more dummy nodes will ever be inserted into the GSS after this point, since a full set of states is henceforth reachable, a test will be made for

every state for each input item to see if that state is already reachable. All of this useless testing is bypassed in the GLR and GLR* algorithms.

It should be noted that this class of grammar's G_m, though it provides a convenient way of computing a complexity bound, is nothing like a natural language grammar. In natural languages, words are shown to be effective in constraining expectations for subsequent words. This property is what makes language modeling effective for speech recognition, for example. This explains why GLR's performance as sentence length grows even with the complex scheduling grammar is shown to be close to linear. In the next section it will be demonstrated that in highly constrained languages like for the Circuit Fix-It shop grammar, there is not much of a difference in performance between GLR* and LR MDP where for the more complex Scheduling grammar the performance of the two algorithms is shown to be orders of magnitude different.

5.7 Empirical Comparison of GLR* and LR MDP

	1-2 Words	3-4 Words	5-6 Words	7-9 Words	10-13 Words
GLR*	.01	.02	.06	.11	.27
MDP	.01	.07	.12	.4	.54

Figure 5.5: Comparison (Secs.) with Hipp grammar

The results from comparing GLR* with LR MDP with Hipp's grammar can be found in Figure 5.5. The results are presented in terms of average number of seconds taken. Here parsing times for the two algorithms were compared on a corpus of 65 sentences in Hipp's Circuit Fix-It Shop domain. The GLR* algorithm is faster than the MDP algorithm, but the difference is not extreme.

	4-5 Words	6-8 Words	9-10 Words
GLR*	6.18	9.7	16.48
MDP	1744.36	6108.83	7015.49

Figure 5.6: Comparison (Secs.) with Sched. grammar

On the other hand, the difference becomes far more pronounced when moving to the more complex 1000 rule Scheduling grammar, as demonstrated in Figure 5.6. The two algorithms are evaluated on a corpus of 20 sentences of varying lengths. Here we see that the MDP algorithm is on average 600 times slower than the GLR* parser on average length sentences, occasionally as much as over 2000 times slower than the GLR* algorithm. Not only is the MDP approach far slower than the GLR* algorithm, it is clearly impractical.

Parsing times of an hour and a half or more for a single sentence of average length are clearly not acceptable for an interactive system.

The difference in results between the two grammars also makes a different point. If results were presented for only the more complex of the two grammars, one could argue that the difference in performance was due to a poor implementation of the MDP algorithm. Considering that the performance of the two algorithms in the simpler domain is so similar, however, it more likely that the large difference in performance with the more complex grammar is due to the complex nature of the algorithm and not just a poor implementation. Though the complexity of the two algorithms was shown to be identical in the previous section, it is clear from these results that the additional flexibility included in LR MDP over GLR* has far reaching computational ramifications. Therefore, the great computational expense of the MDP approach cannot be compensated for simply by implementing the whole system in a more efficient programming language, such as C rather than Lisp, or by using a different parsing formalism, since the comparisons between the two algorithms here were done holding these factors constant.

This experiment makes it clear that the MDP approach does not scale up to a system such as JANUS/Enthusiast which provides the context for the research presented in this dissertation. This experiment does not address the question of how the GLR* algorithm would compare to a modified MDP algorithm with other possible shortcuts added for efficiency or how these shortcuts would affect overall performance of the algorithm in terms of coverage.

5.8 Qualitative Comparison of MDP and GLR*

In this section I qualitatively compare the capabilities of the GLR* parser to LR MDP. I discuss the specific cases which the MDP parser handles which the GLR* parser does not. It is these cases which I will demonstrate that can be handled with ROSE's Combination step. In subsequent chapters I will argue that the two stage repair process is theoretically more powerful than LR MDP while proving to be orders of magnitude more efficient. From this I will conclude that the two stage repair process is superior since it can achieve a higher level of robustness at a far lower computational cost than this implementation of MDP. This is significant since although it would be possible to implement a more powerful version of MDP, it would clearly be even less efficient since it would be less constrained. Therefore, in this case, the two stage repair approach would yield an even bigger computational advantage.

The primary difference between LR MDP and the GLR* parser is in how partial feature structures can be combined as the sentence is being parsed. On the lowest level, both parsers build the same basic partial feature structures as rules fire on the lexical items present in the sentence. As other rules match the non-terminals corresponding to these partial feature structures, they fire such that the partial feature structures can be combined into larger structures. Since the GLR* parser cannot insert missing portions of the input sentence, some rules which might have otherwise made it possible to combine two or more of these partial feature structures into a larger structure will not be able to fire in the GLR* parser. On the other hand, in the MDP parser, this is not a problem. If no constituent is present for a non-terminal in a rule, the MDP parser can insert the non-terminal. However, no new feature structure is built for the inserted non-terminal. Instead, the non-terminal is only present to make it possible for the rule to fire so that the partial feature structures which represent real portions of the input sentence can be combined into a larger structure as specified by the unification portions of the grammar rules.

Therefore, the disadvantage to using the GLR* parser alone is that it may not be able to incorporate all of the partial feature structures into one larger feature structure to the extent that the MDP parser can. Missing portions of the sentence might make it impossible for rules which combine the partial feature structures to fire.

Theoretically, the most powerful version of MDP should be able to find an analysis for any sentence. The idea behind it is to return an analysis for the sentence closest to the input sentence which can be parsed with the grammar. This is made possible by simulating insertion, deletion, and transposition operations on the input sentence. In order to make it viable to test the MDP approach in a system as large as Enthusiast, however, MDP was made more constrained. The first way in which this is so is that while the full MDP algorithm allows insertions, deletions, and transpositions, the more constrained version of MDP allows only insertions and deletions. While this still allows the MDP parser to repair any sentence, in some cases the result will not be as complete as it would have been with the most powerful version of MDP or with the two stage repair process.

Consider the case where the extragrammaticality is caused by a constituent found in the wrong place such as “I am, on the ninth, quite busy”. Assume that a constituent can be built from “on the ninth” and from “quite busy”. Assume further that the grammar contains a rule which makes it possible to combine the “quite busy” with “on the ninth”, thus building a larger constituent, but not in the opposite order. The more powerful algorithm would be able to reverse the order of these constituents, finding a parse which covers the entire sentence. The more restrictive algorithm would be able to find a parse by skipping

over the temporal constituent, finding a parse for "I am quite busy", but missing the information about when the speaker is busy. The GLR* parser would return two constituents, one corresponding to "I am quite busy" and the other corresponding to "on the ninth".

Both the GLR* parser and the constrained MDP implementation have the weakness that feature structures cannot be built for portions of the sentence which are missing. In the case of LR MDP, this is because while skipping extraneous portions of the input sentence is handled on the word level, it simulates insertions of missing portions at the rule level by inserting non-terminals. Since any of a number of different feature structures could be associated with the same non-terminal symbol, this does not make it possible to build feature structures for the non-terminals which are inserted.

Consider the sentence, "I am busy on ninth." A grammatical sentence would have a determiner before "ninth". If missing necessary words were handled on the word level, the missing determiner could just be inserted before "ninth", making it possible to build a complete parse for the sentence. But suppose that in the grammar, there is no non-terminal corresponding to this determiner, but instead the lexical entry "the" is found in the rule which covers phrases like "the ninth", i.e., `NP --> the <ord-num>`. Because LR MDP can only insert non-terminals, the rule which covers phrases like "the ninth" will not be able to fire without the determiner being present in the sentence. The rule which covers prepositional phrases will still be able to fire for "on ninth", however, because it can skip over the noun phrase part of the prepositional phrase rule, building a prepositional phrase out of "on". But it will not be able to build a prepositional phrase which includes "ninth" as the more powerful version would have been able to do.

Of course it is possible to implement the full version of MDP which handles both insertions and deletions on the word level, thus making it possible to build partial feature structures also for the missing portions of the sentence. But with a vocabulary size of 3000 words or more, this is clearly not practical in general. It is imperative to keep in mind the flexibility versus efficiency question. The more restrictive version of MDP used for comparison purposes in this dissertation research is more efficient than the full version since its search space is significantly smaller. And yet, while it is more efficient than full MDP, it is orders of magnitude less efficient than the GLR* parser.

In conclusion, the primary way in which the GLR* parser is shown here to be more restrictive than the MDP parser is that it may not be able to incorporate all of the partial feature structures into one larger feature structure to the extent that the MDP parser can. This is because missing portions of the sentence might make it impossible for rules which combine the partial feature structures to fire. However, I will demonstrate in

subsequent chapters that ROSE's Combination step overcomes this limitation by allowing the partial feature structures to combine in any way allowed by the meaning representation specification. Additionally, ROSE makes it possible to combine partial feature structures in ways not allowed by this implementation of MDP, making it a more powerful and yet orders of magnitude more efficient repair approach.

Part IV

The ROSE Approach

Chapter 6

The ROSE Approach: Overview

The primary objective of the ROSE approach is to handle the problem of extragrammaticality in an effective and efficient way. The ROSE approach was developed in the context of the Enthusiast system (Suhm et al., 1994; Levin et al., 1995), as part of the large-scale JANUS multi-lingual speech-to-speech machine translation system (Lavie et al., 1996). This machine-translation system currently deals with the scheduling domain. The dialogues that provide input to the system are spontaneous conversations between two individuals who are attempting to schedule a meeting.

6.1 ROSE's Two Stage Interpretation Process

The ROSE approach, displayed in Figure 6.1, interprets extragrammatical input in two stages. Data objects, such as sentences and repair hypotheses, are displayed here as rounded rectangles. Processes, such as parsing, are displayed as solid rectangles. And status flags are displayed as hexagons and are used for directing the flow of data objects through the separate processes which make up the ROSE approach. Dashed and dotted rectangles indicate structure.

ROSE's first stage, Repair Hypothesis Formation, is responsible for assembling a set of hypotheses about the meaning of the ungrammatical utterance. This stage is itself divided into two steps, Partial Parsing and Combination. The Partial Parsing step is similar to the concept of the listener "casting his net" for comprehensible fragments of speech. Lavie's GLR* parser (Lavie, 1995; Lavie and Tomita, 1993) is used to obtain an analysis of portions of the speaker's sentence in cases where it is not possible to obtain an analysis for the entire sentence. This parser is capable of skipping over any portion of an input utterance that cannot be incorporated into a grammatical analysis, in order to recover the analysis for the largest grammatical subset of the utterance. The parse for the largest segment plus analyses for the skipped portions together form the set of chunks which are

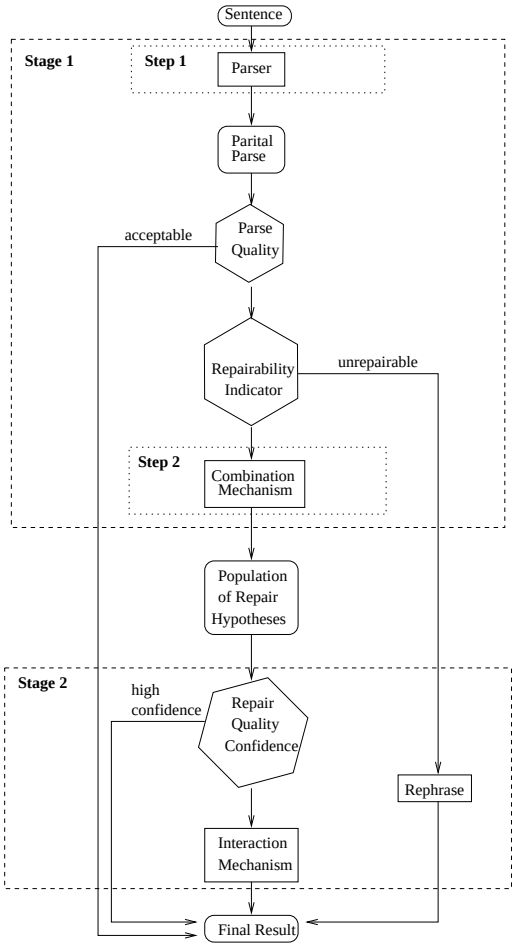

Figure 6.1: **Overview of the ROSE Interpretation Process**

input to the Combination step. In the Combination step, the fragments from the partial parse are assembled into a set of alternative meaning representation hypotheses. A genetic programming approach is used to search for different ways to combine the fragments in order to avoid requiring any hand-crafted repair rules. In ROSE's second stage, Interaction with the user, the system generates a set of queries, negotiating with the speaker in order to narrow down to a single best meaning representation hypothesis.

In this chapter the Combination Mechanism and the Interaction Mechanism are treated as black boxes. The inner workings of these two key components are explored in Chapters 7 and 8 respectively. The GLR* parser is described in greater depth in Appendix A.

6.2 Domain Independence

As mentioned previously, one of ROSE's attractive distinguishing features is that it does not rely on any hand crafted repair rules. Instead it has the ability to search for an acceptable combination of partial analyses by making reference to the meaning representation specification[1] which describes the meaning representation language. This ability makes ROSE completely portable. It can therefore be used in any system where the meaning representation language can be described in a similar format[2]. Though this meaning representation specification is knowledge that must be encoded by hand, it is knowledge that can be used by all aspects of the system, not only the repair module as is the case with repair rules. Arguably, any well designed system would include such a specification, making it clear what is the range of legal meaning representation structures.

Any approach to recovery from parser failure that is not domain independent is impractical in this age where funding is scarce and changing system specifications are the norm, even on the level of which domain to work in. A domain independent approach like ROSE makes it possible to do *only once* most of the work necessary in order to introduce the recovery process. Then, if the system specifications change, the work which has already been done will not need to be duplicated. The ROSE approach only makes use of knowledge sources that are already part of a natural language understanding system, such as a parsing grammar and a meaning representation specification, or ones that can be acquired automatically. Because ROSE makes use of these resources, but is otherwise independent of them, it can be used when these are still under development. Its performance is limited by the parser's ability to produce meaningful chunks and the meaning representation's ability to express the meaning of sentences, but neither the grammar nor the meaning representation specification are required to be complete. This makes it possible to introduce the capability of repair early in the system development process.

The machine learning component makes it possible for ROSE to automatically acquire statistical information used for biasing its search. This component consists of a set of networks that compute connection strength between slots and fillers. This information is used to bias the genetic search in the Combination Mechanism. These networks can "learn by doing" and therefore continue learning while they are being used. An approach to repair that can "learn by doing" is ideal since it makes it possible for ROSE to improve its performance over time. Some types of "learning by doing" are not practical for a large system, however. In particular, learning approaches that add new knowledge in terms of

[1] See Appendix B for an in-depth description of the meaning representation specification used by ROSE.

[2] The ROSE approach can also be used with meaning representation languages that are represented in a different format as long as it is possible to test in which slots it is legal to insert particular chunks.

rules for a parsing grammar become impractical, particularly where multiple users, each with their own idiosyncratic language patterns, are concerned. As new rules are added, the time and space requirements of the system increase. If, on the other hand, new knowledge is added by changing weights, as with a statistical or neural net approach, the time and space requirements of the system are exactly the same after learning as they were before learning. This makes it possible to "learn by doing" with multiple users without the system becoming bogged down. Unlike Lehman's approach (Lehman, 1989), however, its learned knowledge does not save it the trouble of performing the same repair actions on examples that occur multiple times.

6.3 The Key to Efficiency: Channeling Resources

Not every sentence that a human perceives requires the same amount of effort in order to achieve an understanding. Longer and more complex sentences require more effort than short and simple sentences. Other things being equal, grammatical sentences require less effort than corresponding ungrammatical sentences. And sentences that make sense in context require less effort than sentences that are apparently irrelevant or incoherent. Humans appear to be able to adjust their communication strategies to their level of confidence in understanding an utterance spoken by an interlocutor (Clark and Wilkes-Gibbs, 1986; Clark and Schaefer, 1989). When it is necessary, humans ask questions until they are satisfied that they understand. However, if they constantly exerted the maximum amount of effort for understanding every sentence, communication would become a tedious and burdensome process. As it is, however, humans are able to communicate at an acceptable level almost effortlessly most of the time.

In the same way, it is clear that extragrammatical utterances require more effort, or computational resources, on the part of a natural language understanding system than grammatical ones. Ideally, the extra effort is only applied in the cases where it is necessary. The ability to distinguish between cases which require different levels of computational resources is the key to efficiency in the ROSE approach. It is this ability which makes ROSE more efficient than both the Minimum Distance Parsing (MDP) approach and the Incremental Repair with Local Hypotheses (IRLH) approach.

6.3.1 Related Work on Channeling Resources

Since it is impossible for a natural language understanding system to determine whether a sentence is grammatical before it is parsed, "casting and combining" models like

ROSE have the unique ability to withhold additional resources in the case where the sentence is grammatical. If the sentence parses acceptably during the "casting" stage, there is no need to enter the "combining" stage. Models such as standard Minimum Distance Parsing (MDP) in which meaning representation hypotheses are formed in a single stage must either allow for backtracking, which may require large amounts of redundant computation, or exert the full amount of effort on every sentence whether it is required or not.

Once meaning representation hypotheses are formed, the question becomes one of how much, if any, interaction is required in order to be certain that the correct hypothesis is selected. Smith addresses this question in (Smith, 1997). He reports that interaction is necessary since flexible interpretation algorithms can sometimes arrive at the wrong analysis, leading to miscommunications between system and user. However, he makes the point that the best strategy for interaction with the user is one in which the system only engages in verification in the case where the accuracy of the system's best interpretation is seriously in question, or a precise understanding is required for the successful continuation of the dialog. He uses two different flags in order to attempt to accomplish this goal. The first flag, the *parse cost* flag, measures how far Hipp's MDP parser (Hipp, 1992) needed to deviate from its grammar in order to form an analysis of the sentence. Analyses with higher costs are considered more suspect than ones with lower associated cost. The second flag, the *expectation cost* flag, measures how coherently the analysis fits into the discourse model at the current state in the dialogue. Analyses which seem incoherent are considered more suspect than those that receive a rating indicating that they are highly expected.

As in Smith's approach, interaction in ROSE is used conservatively, only in the case where the system is not confident that its best hypothesis is the correct one. This is in contrast to the Incremental Repair with Local Hypotheses (IRLH) approach in which every local repair is confirmed with the user before it is performed. Because in this incremental approach it is difficult to evaluate the confidence with which individual repair actions are performed, it is difficult to distinguish between repairs that need to be confirmed with ones that do not. In contrast, ROSE's global approach makes it possible to evaluate the system's confidence about repair hypotheses, avoiding the necessity to over-burden the user with tedious questions.

6.3.2 Channeling Resources in ROSE

ROSE's three status flags, displayed as hexagons in Figure 6.1, allow ROSE to make efficient use of its resources, channeling them to those portions of the process that are likely to yield an improvement in interpretation quality. For example, if the parser is

able to obtain a high quality parse for the input sentence, then it would be a waste of resources to make use of the Combination Mechanism or the Interaction Mechanism. For this reason, if the `Parse-Quality` flag indicates that the parse quality is likely to be high, the result of the parser is passed on as the final result, bypassing the Combination step and the Interaction with the User stage. Similarly, if the parse result is so bad that a repair cannot be made, then it is equally useless to make use of the Combination Mechanism and Interaction Mechanism. So if the `Repairability Indicator` shows that the partial parse does not provide useful building blocks to be used by the Combination Mechanism, again the Combination step and the Interaction with the User phase are bypassed, and a rephrase is requested. But if the `Repairability Indicator` indicates that useful building blocks for combination were constructed during the Partial Parsing step, then ROSE makes use of the Combination Mechanism. After a population of hypotheses are constructed by the Combination Mechanism, a third status flag, the `Repair Quality Confidence` flag, is used to determine whether interaction is necessary. Since 83% of the time the best of the top set of repair hypotheses is the best possible hypothesis, it would be a waste of time to engage in a verification subdialogue after each repair. If the `Repair Quality Confidence` indicates that interaction is necessary, ROSE makes use of the Interaction Mechanism. Otherwise the result of the top repair hypothesis created by the Combination Mechanism is returned as the final result. Therefore, ROSE tailors its approach to each specific type of case.

6.4 Examples

ROSE's three status flags, the `Parse-Quality` flag, the `Repairability Indicator` flag, and the `Repair Quality Confidence` flag, are introduced above. In this section they are explored in greater depth in the context of several examples which demonstrate their usefulness.

6.4.1 Example 1

Similar to Smith's parse cost flag, the `Parse-Quality` flag evaluates how much of the sentence was skipped during parsing, and how well the analysis rated statistically. For an in-depth discussion of statistical evaluation of parses and how the statistical score is combined with the percentage of sentence skipped in GLR* see (Lavie, 1995). The only difference between the original formula used by Lavie in (Lavie, 1995) and that used in ROSE is that ROSE is stricter on how much of the sentence can be acceptably skipped for an analysis rated as good. Whereas in (Lavie, 1995), skipping up to 25% of the input sentence

Sentence: at like from two to four
Words Skipped: at like
Statistical Score: none computed because not ambiguous
Parser's Analysis:

```
(((SENTENCE-TYPE *FRAGMENT)
  (WHEN
      ((END ((FRAME *SIMPLE-TIME) (HOUR 4)))
       (START ((FRAME *SIMPLE-TIME) (HOUR 2)))
       (INCL-EXCL INCLUSIVE)
       (FRAME *INTERVAL)))))
```

Parse-Quality-Flag: Good
Interpretation: FROM TWO O+CLOCK TILL FOUR O+CLOCK

Figure 6.2: **ROSE: Example 1**

is acceptable, in ROSE skipping only 10% is allowed. This was decided experimentally over a set of 100 examples in an attempt not to miss examples needing repair and at the same time avoiding marking as bad those examples where repair is not necessary. If the `Parse-Quality` flag indicates that the quality of the parse is good, the parser's result is returned and no resources are wasted on repair. This is the case in the the example in Figure 6.2.

In this case, the first two words, "at like", must be skipped in order to derive an analysis for this string. But since the number of words skipped is small and the analysis derived is not suspect statistically speaking, the `Parse Quality` flag rates this sentence as good. No repair is necessary and no repair is performed. Thus, no resources are wasted on unnecessary repair.

6.4.2 Example 2

In the next example, displayed in Figure 6.3, the `Parse Quality` flag indicates that the sentence does not parse acceptably.

In this case, since the entire sentence is skipped, the `Parse Quality` flag indicates that repair is necessary because some essential meaning is likely to have been skipped. Once the `Parse Quality` flag indicates that repair is necessary in order to derive an acceptable interpretation, the `Repairability Indicator` flag indicates whether repair is possible with ROSE. In order to make this determination, it tests whether the parser was able to extract at least two chunks. If this is not the case, there is no point in making use of the Combination Mechanism. For this sentence, the parser was not able to extract any meaningful chunks.

Sentence: fat albert's on
Words Skipped: fat albert's on
Statistical Score: none because not ambiguous
Parser's Analysis:
nil **Parse-Quality-Flag:** Bad
Interpretation: *none*
Chunks: *none*

Figure 6.3: **ROSE: Example 2**

No repair is possible, so no repair is attempted, though a rephrase is requested. Again, no unnecessary resources are wasted. In cases like these, the best ROSE can do is to ask the user to rephrase his utterance.

6.4.3 Example 3

Often when the parse quality is bad, the parser does manage to extract meaningful chunks. In these cases, the `Repairability Indicator` flag indicates that repair is likely to yield an improvement in interpretation quality. Such is the case with the example in Figure 6.4. In the evaluation of ROSE with GLR with restarts presented in Chapter 10, repair occurred about one third of the time, but in only about 25% of those cases did the parser produce sufficient chunks for constructing an acceptable hypothesis. This indicates that a more sophisticated `Repairability Indicator` might have the potential for making the ROSE approach more efficient.

Once a set of repair hypotheses have been generated, the question becomes how much, if any, interaction is required in order to verify which of the set of hypotheses generated is the best to return. In ROSE, the `Repair Quality Confidence` flag indicates whether interaction is likely to yield an improvement in interpretation accuracy. It makes this determination based on whether there were any differences between the results of the top set of repair hypotheses generated by the Combination Mechanism and how much of a difference exists between the parse result and the best repair result[3]. ROSE's result for the example in Figure 6.4 can be found in Figure 6.5. In this case, each of the top hypotheses generated the same resulting structure. So the `Repair Quality Confidence` flag indicates that no interaction is necessary.

[3]If the best ranked hypothesis contains more than 4 more frames and atomic fillers than the second ranked hypothesis, the **Repair Quality Confidence** flag indicates that no interaction is necessary. This heuristic was determined to be useful experimentally.

Sentence: HOW +BOUT THE AFTER TEN THIRTY IN THE MORNING
Words Skipped: THE AFTER TEN THIRTY IN THE MORNING
Statistical Score: 7.225746720528208
Parse Quality: Bad
Parser's Analysis:
(((FRAME *HOW)
(SENTENCE-TYPE *QUERY-REF)))
Interpretation: HOW ABOUT
Repairability: Yes
Chunks:
((FRAME *HOW))
((FRAME *INTERVAL)
(INCL-EXCL EXCLUSIVE)
(START
((FRAME *SIMPLE-TIME)
(AM-PM AM)
(MINUTE 30)
(HOUR 10))))
((FRAME *SIMPLE-TIME)
(AM-PM AM)
(MINUTE 30)
(HOUR 10))

Figure 6.4: **ROSE: Example 3 Part 1**

Since the top set of hypotheses all generated identical results, and because the result appears to be an improvement over the parser result, the confidence that this is the best hypothesis is high. No interaction is necessary, and none is performed. So no time is wasted on unnecessary interaction.

6.4.4 Example 4

In the majority of the cases, however, the `Repair Quality Confidence` flag indicates that interaction is required in order to be certain that the best result is being returned. In the example in Figure 6.6, for instance, the Hypothesis Formation stage yielded five alternative translations which are displayed in Figure 6.7. The best one happens to be the one that was ranked by the Hypothesis Formation stage as third. So the only way ROSE can return the optimal result is by determining through interaction that this is the preferred interpretation. The interaction is displayed in Figure 6.8.

Best analysis after repair:

```
(((FRAME *HOW)
  (WHEN ((FRAME *INTERVAL)
         (INCL-EXCL EXCLUSIVE)
         (START
             ((FRAME *SIMPLE-TIME)
             (AM-PM AM) (MINUTE 30)
             (HOUR 10)))))))
```

Interpretation: HOW ABOUT AFTER TEN THIRTY A.M

Figure 6.5: **ROSE: Example 3 Part 2**

In this case interaction was necessary in order to select from a set of alternative hypotheses returned after the Hypothesis Formation stage. In Chapter 8, other functions of interaction will be discussed, in particular assessing ROSE's confidence that any of its hypotheses are correct and asking for a rephrase in cases where it determines that it does not have sufficient confidence in any of its hypotheses. Although it is true that in 85% where ROSE produces an acceptable hypothesis in the top ten, it is ranked as first, unless it would be possible for ROSE to determine some other way that none of its hypotheses are acceptable, it is unlikely that using interaction more frugally would allow ROSE to maintain its high level of performance.

In Chapter 9, Discourse ROSE, an alternative version of ROSE which makes use of a plan based discourse processor will be discussed. It makes it possible to generate some of ROSE's queries to the user in terms of the task rather than on the level of the speaker's literal meaning.

Sentence: HI HOW +BOUT MEETING ANY TIME < BEFORE TWO PM
Words Skipped: HI < BEFORE TWO PM
Statistical Score: 9.079212043065768
Parse Quality: Bad
Parser's Analysis:
((SENTENCE-TYPE *QUERY-REF)
 (WHAT ((FRAME *MEETING)))
 (WHEN ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME)))
 (FRAME *HOW))
Interpretation: HOW ABOUT MEETING ANY TIME
Repairability: Yes
Chunks:
((FRAME *HOW)
 (WHEN ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME)))
 (WHAT ((FRAME *MEETING)))
((END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
 (INCL-EXCL EXCLUSIVE)
 (FRAME *INTERVAL))
((UNIT-NAME *TIME) (FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2))
((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME))
((FRAME *HOW) (WHAT ((FRAME *MEETING))))
((FRAME *GREET) (TYPE ((FRAME *HELLO))))
((FRAME *HOW))
((FRAME *MEETING))
((FRAME *HELLO))

Figure 6.6: **ROSE: Example 4**

```
Alternative Repair Hypotheses:

"HELLO – HOW ABOUT IF MEET ANY TIME – BEFORE TWO P.M."
(((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((ATTITUDE *HOW-ABOUT)
 (WHEN ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME)))
 (FRAME *MEET))
((WHEN ((END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
 (INCL-EXCL EXCLUSIVE)
 (FRAME *INTERVAL)))))

"HELLO – HOW ABOUT MEETING TWO P.M. ANY TIME – BEFORE"
(((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((WHAT ((FRAME *MEETING)))
 (WHEN (*MULTIPLE* ((UNIT-NAME *TIME) (FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2))
                   ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME))))
 (FRAME *HOW))
((WHEN ((FRAME *SPECIAL-TIME) (NAME BEFORE)))))

"HELLO – HOW ABOUT MEETING BEFORE TWO P.M. ANY TIME "
(((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((WHAT ((FRAME *MEETING)))
 (WHEN (*MULTIPLE* ((END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
                    (INCL-EXCL EXCLUSIVE) (FRAME *INTERVAL))
                   ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME))))
 (FRAME *HOW)))

"HELLO – HOW ABOUT BEFORE TWO P.M. MEETING ANY TIME"
(((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((WHAT (*MULTIPLE* ((END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
                    (INCL-EXCL EXCLUSIVE) (FRAME *INTERVAL))
                   ((FRAME *MEETING))))
 (WHEN ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME)))
 (FRAME *HOW)))

"HOW ABOUT – HELLO – AFTER ANY TIME TO MEET BEFORE TWO P.M."
(((FRAME *HOW))
((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((START ((PURPOSE ((FRAME *MEET) (VERB-FORM ING)))
                   (FRAME *SPECIAL-TIME) (NAME TIME) (SPECIFIER ANY)))
          (FRAME *INTERVAL)
          (INCL-EXCL EXCLUSIVE)
          (END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))))
```

Figure 6.7: **ROSE: Example 4 Alternative Repair Hypotheses**

Interaction: **What you said:** HI HOW 'BOUT MEETING ANY TIME < BEFORE TWO PM
What ROSE understood: HELLO – HOW ABOUT MEETING ANY TIME – BEFORE TWO P.M.
Interaction:
ROSE: Was something like HOW ABOUT ANY TIME part of what you meant?
SA1TJ: Yes.
ROSE: Was something like HOW ABOUT BEFORE part of what you meant?
SA1TJ: Yes.

Result:

HELLO - HOW ABOUT MEETING BEFORE TWO P.M. ANY TIME

Figure 6.8: **ROSE: Example 4 Interaction**

Chapter 7

The Combination Mechanism

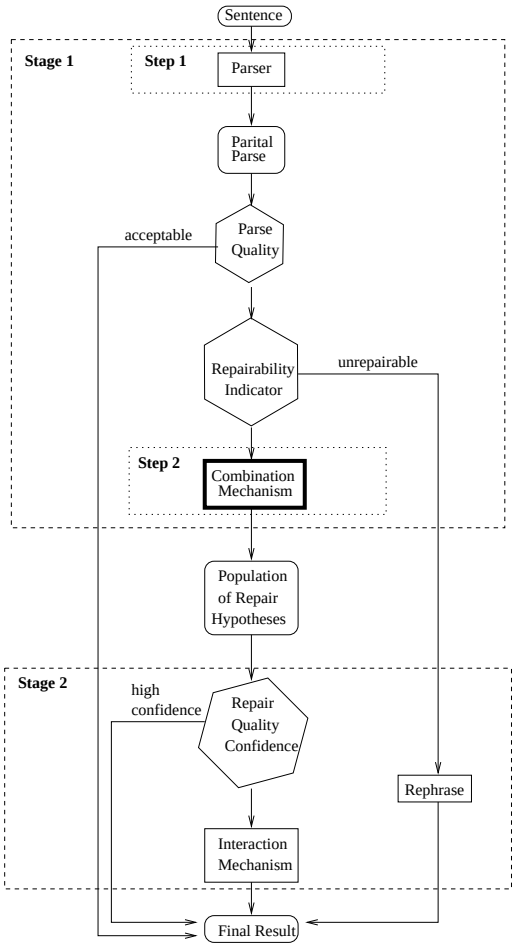

Figure 7.1: **Overview of the ROSE Interpretation Process**

As discussed in the previous chapter, the ROSE approach, displayed in Figure 7.1, interprets extragrammatical input in two stages. ROSE's first stage, Repair Hypothesis Formation, is responsible for assembling a set of hypotheses about the meaning of the ungrammatical utterance. This stage is itself divided into two steps, Partial Parsing and

Combination. The Partial Parsing step is similar to the concept of the listener "casting his net" for comprehensible fragments of speech. In the Combination step, which is the focus of this chapter, the fragments from the partial parse are assembled into a set of alternative meaning representation hypotheses. A genetic programming approach is used to search for different ways to combine the fragments in order to avoid requiring any hand-crafted repair rules. In ROSE's Interaction with the user stage, discussed in depth in the next chapter, the system generates a set of queries, negotiating with the speaker in order to narrow down to a single best meaning representation hypothesis.

7.1 Combination vs. IRLH

Both ROSE and Incremental Repair with Local Hypotheses (IRLH) are "casting and combining" approaches to handling extragrammatical input. The repair processes both in ROSE and in IRLH are analogous in some ways to fitting pieces of a puzzle into a mold which contains receptacles for particular shapes. In this analogy, a meaning representation specification, described in depth in Appendix B, acts as the mold with receptacles of different shapes, making it possible to compute all of the ways partial analyses can fit together in order to create a structure that is legal in this frame based meaning representation. Readers not familiar with this meaning representation specification are encouraged to read Appendix B.

What distinguishes ROSE from IRLH is that it separates Hypothesis Formation and Interaction with the User into two separate stages. In IRLH, queries are generated to verify each repair step (Local Repair Hypotheses) rather than first constructing a set of alternative ways of fitting together the whole set of fragments (Global Repair Hypotheses). The IRLH approach searches for the complete meaning representation structure by generating and testing individual local hypothesised repair actions. When these hypotheses are confirmed to be correct through interaction with the speaker, the repair module then makes the specified repair. Because IRLH only asks questions about local repair hypotheses, it requires built in strategies for determining which repairs to try first, and thus which questions to ask first, etc. A meta-strategy decides which of its eight individual strategies to employ at different points in the repair process.

In ROSE, on the other hand, global repair hypotheses are constructed in the Combination Mechanism. Genetic programming search (Koza, 1992; Koza, 1994) replaces the meta strategy employed by the IRLH approach. The Combination Mechanism's strategy is an emergent property of the semi-random program generation process and the fitness function which evaluates the programs that are evolved in each generation.

7.2 Why Genetic Programming

Genetic programming (Koza, 1992; Koza, 1994) is an opportunistic search algorithm that can search a large space efficiently by first sampling widely and shallowly, and then narrowing in on the regions surrounding the most promising looking points. It is most appropriate for problems where the solution can be represented easily as a computer program and, though it is not known ahead of time what the ideal solution will look like, it is possible to evaluate the goodness of potential solutions efficiently.

Recovery from parser failure is a natural application of genetic programming. The most compelling reason why genetic programming is more appropriate than other potential search algorithms is that it provides a natural representation for the problem. One can easily conceptualize the process of constructing a global meaning representation hypothesis as the execution of a computer program that assembles the set of chunks returned from the parser. This program would specify the operations required for building larger chunks out of smaller chunks and then even larger ones from those. Because the programs generated by the genetic search are hierarchical, they naturally represent the compositional nature of the repair process. Additionally, genetic programming is appropriate because although it is not known a priori what the ideal set of repairs will be, it is possible to evaluate how well one particular set of repair actions does at constructing a meaning representation structure.

Number of Chunks	Search Space Size
1	1
2	4
3	36
4	96
5	735

Figure 7.2: **Number of possible repair hypotheses corresponding to the number of chunks produced by the parser**

Genetic programming is also appropriate because it has the ability to search a large space efficiently. An efficient search algorithm is necessary because the space of alternative repair hypotheses is too large to search exhaustively. Figure 7.2 demonstrates how the total number of possibilities grows as the number of chunks produced by the parser increases. This growth is calculated by considering that for any set of chunks produced by the parser, the Combination Mechanism must both decide which subset of those chunks should be included in the final hypothesis and how the selected subset should be composed. The number of alternatives examined by the genetic programming algorithm can be fixed ahead of time

by setting the population size (the number of programs constructed for each generation), and the maximum number of generations. By limiting the population size to 32 and the number of generations to 4, the Combination Mechanism is constrained to search only 128 alternatives. Thus, if the number of chunks produced by the parser is any more than 4, the genetic programming approach yields a significant savings in terms of number of alternatives explored. The average number of chunks produced by the parser in the evaluations presented in this dissertation was 5.66. In 55.7% of the cases, the parser produced more than 4 chunks. Nevertheless, in the evaluation presented in Chapter 10, the genetic programming search successfully ranked an acceptable hypothesis in the top ten list of hypotheses returned by the Combination Mechanism in 93.0% of the cases where the parser produced chunks sufficient for building an acceptable solution[1]. And in 84.6% of these cases, the best hypothesis was ranked as first. These results are explored in greater depth in Chapter 10.

Comparing genetic search with other efficient search algorithms is beyond the scope of this dissertation research. However, it would be an interesting direction for future research.

7.3 Applying the Genetic Programming Paradigm

The genetic programming algorithm starts out with a set of functions and terminals which can be composed in order to build a program. Additionally, a fitness function is provided which evaluates the goodness of each hypothesized program generated during the genetic search. In the 0th generation, a set of programs are generated randomly. Usually the programs in this initial population have a low fitness. But some programs have a better fitness than others. Those that are relatively more fit are selected for reproduction. *Crossover* and *Mutation* are used to produce the next generation from the most fit programs from the current generation. The *Crossover* operation swaps a subprogram from one parent program with a subprogram of the other parent program. The *Mutation* operator swaps a function with another randomly chosen function or a terminal with another randomly selected terminal. Through this process, the subprograms responsible for making certain programs better than others become more common in the population. Eventually the genetic programming search converges upon one or more acceptable solutions.

[1]Out of 500 sentences used in the Combination Mechanism evaluation, there were only 4 such that the parser produced sufficient chunks for producing an acceptable solution and the Combination Mechanism did not include the acceptable solution in the top ten list of solutions returned. This is 3% of the cases where repair occurred overall.

The genetic programming paradigm can be applied to any problem that can be solved with a computer program. There are five steps involved in applying the genetic programming paradigm to a particular problem:

- Determine a set of terminals
- Determine a set of functions
- Determine a fitness measure
- Determine the parameters and variables to control the run
- Determine the method for deciding when to stop the evolution process.

An in depth discussion of these five steps as they are applied to a wide range of problems can be found in Chapter 7 of (Koza, 1992).

7.3.1 The Set of Terminals for Repair

All of the programs evolved with the genetic programming technique are composed of the set of terminals and functions that are initially provided. Terminals can be variables, constants, or structures of any kind. In the Combination Mechanism, the terminals are chunks that are constructed during the Partial Parsing step.

In the example found in Figure 7.3 the GLR* parser attempts to handle the sentence "That wipes out my mornings." The expression "wipes out" does not match anything in the parsing grammar. The grammar also does not allow time expressions to be modified by possessive pronouns. So "my mornings" also does not parse. Although the grammar recognizes "out" as a way of expressing a rejection, as in "Tuesdays are out," it does not allow the time being rejected to follow the "out". However, although the parser was not able to obtain a complete parse for this sentence, it was able to extract four chunks.

The chunks are feature structures in which the parser encodes the meaning of portions of the user's sentence. The four chunks extracted by the parser each encode a different part of the meaning of the sentence "That wipes out my mornings." The first chunk represents the meaning of "that". The second one represents the meaning of "out". Since "out" is generally a way of rejecting a meeting time in this domain, the associated feature structure represents the concept of a response which is a rejection. Since "wipes" does not match anything in the grammar, this token is left without any representation among the fragments returned by the parser. The last two chunks represent the meaning of "my" and "mornings" respectively.

Sentence: *That wipes out my mornings.*

Partial Analyses:

Chunk1: that

((ROOT THAT)
(TYPE PRONOUN)
(FRAME *THAT))

Chunk2: out

((TYPE NEGATIVE)
(DEGREE NORMAL)
(FRAME *RESPOND))

Chunk3: my

((ROOT I)
(TYPE PERSON-POSS)
(FRAME *I))

Chunk4: mornings

((TIME-OF-DAY MORNING)
(NUMBER PLURAL)
(FRAME *SIMPLE-TIME)
(SIMPLE-UNIT-NAME TOD))

Figure 7.3: Parse Example

7.3.2 The Function Set for Repair

Ideal Repair Hypothesis:

```
(MY-COMB                                                    ;insert arg2 into arg1 in slot
  ((FRAME *RESPOND) (DEGREE NORMAL) (TYPE NEGATIVE))        ;arg1
  ((TIME-OF-DAY MORNING) (NUMBER PLURAL)                    ;arg2
   (FRAME *SIMPLE-TIME) (SIMPLE-UNIT-NAME TOD))
  WHEN)                                                     ; slot
```

Ideal Structure:

```
((FRAME *RESPOND)
 (DEGREE NORMAL)
 (TYPE NEGATIVE)
 (WHEN ((FRAME *SIMPLE-TIME)
        (TIME-OF-DAY MORNING)
        (NUMBER PLURAL))))
```

Gloss: Mornings are out.

Figure 7.4: Combination Example

The functions provided to the genetic programming search can be either primitive functions in the language in which the programs are being written, e.g.. Lisp, or they can be custom-made functions for the particular task for which the program is being evolved. Clearly, it is more efficient and straightforward to start with functions that are directly relevant for the task, whenever possible, rather than evolving them from first principles.

For repair, a single operator is provided. `MY-COMB` is a simple function taking three arguments, a parent chunk, a child chunk, and a slot. It attempts to insert the child chunk into some slot in the parent chunk. If it is successful, it instantiates the slot parameter to the value of the slot selected. An example of its use can be found in Figure 7.4. It selects a slot, if a suitable one can be found, and then instantiates the third parameter to this slot. In this case, the `WHEN` slot is selected. So the feature structure corresponding to "mornings" is inserted into the `WHEN` slot in the feature structure corresponding to "out". The result is a feature structure indicating that "Mornings are out." Though this is not an exact representation of the speaker's meaning, it is the best that can be done with the

available feature structures. In some cases, the second structure can not be inserted into any slot in the first structure, or the first structure may not have any slots at all. In that case, if the top level semantic frame of both chunks is the same, the two structures are merged into a single chunk. Otherwise, the largest chunk is returned. This way, no matter what the relationship between the two chunks which form the input, the output is always a single chunk. The output of the `MY-COMB` operator can thus always be one of the inputs to another instantiation of the `MY-COMB` operator. Therefore, instantiations of the `MY-COMB` operator can fit together compositionally, forming a program to assemble the set of chunks into a single chunk.

The internal representation of a chunk includes information about what portion of the sentence is covered by the corresponding chunk. This information is used during the operation of the `MY-COMB` operator in order to insure that resulting structures will never contain a representation for the same portion of the sentence more than once.

7.3.3 The Fitness Function for Repair

```
(defun fitness-eval (NUM_CONCEPTS NUM_STEPS STAT_SCORE PERCENT_COV)
  ((1 * (1 / STAT\_SCORE)) +
   (0.55 * NUM\_STEPS) +
   (1.15 * (1 / NUM\_CONCEPTS)) +
   (1.15 * (1 / PERCENT\_COV))))
```

Figure 7.5: **Hillclimbing Trained Linear Combination Fitness Function**

Once a population of hypotheses is generated, each individual in the population is evaluated for its fitness. Since the purpose of the repair module is to evolve a hypothesis that generates the ideal meaning representation structure, hypotheses that produce meaning representation structures closer to the ideal representation should be ranked as better than others that produce structures that are more different. Of course, the repair module does not have access to that ideal structure while it is searching for the best combination of chunks. So a fitness function is trained that must estimate how close the result of a particular repair hypothesis is to the ideal structure by considering secondary evidence. In this section I describe two alternative fitness functions used in the Combination Mechanism and how they were trained.

The fitness measure is the most critical part of the genetic search since the fitness evaluations are what guide the search process. Since survival of the fittest is the key to the

evolutionary process, the determination of which hypotheses are more fit is absolutely crucial. The fitness function measures how well each evolved program performs at a particular task or fits some task-specific criteria. The fitness function assigns a better score to those programs that perform better or conform better to the stated criteria. The programs that score better are more likely to be selected for reproduction in order to generate the next generation.

```
(defun fitness-eval (NUM_CONCEPTS NUM_STEPS STAT_SCORE PERCENT_COV)
 (+ (% 5.55396848337444 NUM_CONCEPTS) (% (+ (% NUM_SCORES (+ (- (+
 STAT_SCORE (* PERCENT_COV STAT_SCORE)) (% (% STAT_SCORE (- (-
 (- (* (+ NUM_SCORES NUM_SCORES) NUM_CONCEPTS) NUM_CONCEPTS) (+
 NUM_CONCEPTS 0.6630144755649448)) (% NUM_SCORES NUM_CONCEPTS))) (-
 (+ (+ (- 1.0 (% NUM_SCORES (+ STAT_SCORE NUM_SCORES))) (+ (% (+
 NUM_CONCEPTS 0.17354198692997125) NUM_CONCEPTS) (* STAT_SCORE
 NUM_CONCEPTS))) (* NUM_CONCEPTS (% (* (+ (% NUM_CONCEPTS (*
 STAT_SCORE (- 3.474824611318681 NUM_SCORES))) (- (* 1.392524484727776
 (+ (- NUM_SCORES NUM_CONCEPTS) (% (% (% 8.318143522980689 PERCENT_COV)
 NUM_SCORES) PERCENT_COV))) (- (- (+ (- (% (- -8.59992566918951 STAT_SCORE)
 (* STAT_SCORE STAT_SCORE)) NUM_CONCEPTS) (* PERCENT_COV NUM_CONCEPTS))
 NUM_SCORES) 18.39528226908547))) PERCENT_COV) (* (- 0 (- (% (+ (%
 NUM_CONCEPTS STAT_SCORE) (- 4.149918809137269 NUM_SCORES)) (- STAT_SCORE
 (* (- NUM_SCORES (- (* NUM_CONCEPTS STAT_SCORE) PERCENT_COV))
 (- (- (* STAT_SCORE STAT_SCORE) (- (- NUM_SCORES NUM_CONCEPTS) STAT_SCORE))
 (- (* (% NUM_CONCEPTS 1.441482643336749) NUM_SCORES) (+ NUM_SCORES
 -1.7897609727409494))))))) (+ (% NUM_CONCEPTS NUM_CONCEPTS) (* (- (* (-
 (% PERCENT_COV STAT_SCORE) NUM_CONCEPTS) 1.0) PERCENT_COV) (- (+
 PERCENT_COV (+ STAT_SCORE 5.268861434547631)) NUM_CONCEPTS)))))
 NUM_CONCEPTS)))) (% (+ (- 1.0 (* 11.594150078293938 (% NUM_SCORES
 -2.663707176066799))) PERCENT_COV) (- (+ PERCENT_COV NUM_CONCEPTS) (*
 (- (+ NUM_CONCEPTS STAT_SCORE) 6.8380438521681555) (- PERCENT_COV
 (+ NUM_CONCEPTS (+ STAT_SCORE NUM_CONCEPTS)))))))))
 (- (* 1.4742101265463097 NUM_SCORES) (% (* STAT_SCORE NUM_SCORES)
 (+ PERCENT_COV NUM_CONCEPTS))))) PERCENT_COV) (- (% (+ (% (* STAT_SCORE
 (* 0.9641986712646657 PERCENT_COV)) (- STAT_SCORE 1.7003116676119676))
 (% STAT_SCORE (- (- (- (* (+ NUM_SCORES NUM_SCORES) NUM_CONCEPTS) NUM_CONCEPTS)
 (+ PERCENT_COV PERCENT_COV)) (% NUM_SCORES NUM_CONCEPTS)))) (%
 -0.05754932577607619 STAT_SCORE)) (+ (- (* (- (- NUM_SCORES (% (%
 (- (- PERCENT_COV (% NUM_CONCEPTS PERCENT_COV)) PERCENT_COV) NUM_CONCEPTS)
 STAT_SCORE)) PERCENT_COV) NUM_SCORES) (% NUM_SCORES NUM_CONCEPTS)) (+ PERCENT_COV
 (* NUM_SCORES NUM_SCORES)))))))
```

Figure 7.6: **Genetic Programming Trained Fitness Function**

7.3.3.1 Components of the Fitness Score

An ideal fitness function for repair would rank hypotheses that generate structures closer to the target structure better than those that are more different. *Lower* fitness values are preferred over higher ones, so the best ranking hypothesis will be assigned the lowest fitness score. Since the Combination Mechanism does not have access to this information, however, it must rely upon indirect evidence for determining which hypotheses are better than others. The two alternative trained fitness functions used in ROSE are displayed in Figures 7.5 and 7.6 respectively. The function displayed in Figure 7.5 is a linear combination trained with a hillclimbing approach. The function in Figure 7.6 was trained using genetic programming, and thus is difficult for a human to understand. The training procedures used to create these two alternative fitness functions are described below.

The trained fitness functions combine four pieces of indirect evidence about the goodness of repair hypotheses. These four pieces of information include the number of frames and atomic fillers in the resulting structure (NUM_CONCEPTS), the number of repair actions involved (NUM_STEPS), statistical goodness of repair actions (STAT_SCORE), and the percentage of the sentence that is covered by the repair (PERCENT_COV). These pieces of information are generally useful in ranking hypotheses. Intuitively, one would prefer more complete hypotheses over less complete ones. And following the principle of Occam's razor, other things being equal, one would prefer simpler solutions over more complex ones. Since simpler solutions may be less complete, and more complete hypotheses might be more complex, the trained fitness function must learn how to balance these two qualities. Normally hypotheses containing more frames and atomic fillers and covering more of the sentence are more complete, and thus more correct. Thus, both NUM_CONCEPTS and PERCENT_COV provide an estimate of the completeness of solutions. Simpler hypotheses with fewer repair actions are less likely to contain a mistaken repair action. A program with fewer steps is simpler than a program with more steps. Thus NUM_STEPS provides an estimate of the simplicity of the program. And as much as the statistical information gives a reliable indication of goodness of fit between slots and fillers and goodness of chunks, repair hypotheses with better average statistical score are more likely to be better hypotheses. Thus, STAT_SCORE also provides the fitness function with useful information.

I attempted to use two other scores without success. The first one was used to estimate the likelihood that the produced structure contained the top level semantic frame. It was hoped that this would help to avoid the case where hypotheses with large temporal expressions but lacking the top level semantic frame looked much better to the fitness function than hypotheses producing structures with the top level semantic frame

but lacking a representation for part or all of the temporal expression. In cases like this, the danger is that none of the hypotheses produced in the next generation will contain the top level semantic frame. The problem with this score was that it was difficult to estimate accurately since temporal expressions can also be the representation for a whole sentence by themselves. So it was unable to be effective in the one case it was designed to improve. Another score I attempted to use estimated for each slot how important it was to be filled. The purpose of this parameter was to prefer hypotheses where the essential slots were filled in over those where less important slots were filled in. It was also difficult to estimate this accurately because of sparseness of training data. With more training data, this might have been a useful score. What makes these particular scores attractive is that they can be determined easily and accurately, and they provide useful information.

7.3.3.2 Training the Fitness Function

The first step in training a fitness function is to decide which pieces of information to make available to the fitness function for it to use in making its decision. The fitness function, once it is trained, combines these pieces of information into a single score that can be used for ranking the hypotheses. As mentioned above, in the current version of ROSE, four pieces of information are given: the number of operations in the repair hypothesis, the number of frames and atomic slot fillers in the resulting meaning representation structure, the average of the statistical scores for the set of repairs that were made, and the percentage of the sentence covered by the resulting structure.

Both alternative fitness functions were trained over a corpus of 48 sentences needing repair coupled with their corresponding ideal meaning representation structures. To generate training data for the fitness functions, the Combination Mechanism was run using an ideal fitness function that evaluated the goodness of hypotheses by comparing the meaning representation structure produced by the hypothesis with the ideal structure. It assigned a fitness score to the hypothesis equal to the number of frames and atomic fillers in the largest substructure shared by the produced structure and the ideal structure. With the programs in each generation ranked by the ideal fitness function, the four scores that would be used by the trained fitness functions were extracted from each program. The result was a set of ranked lists of sets of scores. The goal of the training process in both cases was then to learn a function that combines these scores in such a way that when a list of sets of scores is sorted based on the combined score, the ordering comes out as similar as possible to the ideal ordering. Correctly sorting the sets of scores is equivalent to ranking

the hypotheses themselves. Therefore, a function that can successfully sort the scores in the training examples will be correspondingly good at ranking repair hypotheses.

The linear combination fitness function was trained using a hillclimbing technique. Since lower fitness values are better than higher ones, the multiplicative inverse of the number of concepts, the percentage of the sentence covered, and the average statistical score were passed into the linear combination function rather than the original values which are such that bigger scores are better than smaller scores. All of the coefficients were initially set to 1. On each iteration of the training algorithm, a set of alternative new fitness functions were proposed by either adding or subtracting .15 from one of the coefficients. The set of alternative new fitness functions included every way of doing this. Each new fitness function was tested over the training set, and the best of these replaced the current one if its performance exceeded that of the current one. The performance for each function was measured by computing the length of the greatest common subsequence[2] between the ideal ordering and the generated ordering for each list of sets of scores in the training set. The hillclimbing algorithm terminated after the first iteration where none of the alternative new hypotheses performed better than the current one. The entire training process took less than an hour.

The genetic programming trained fitness function took as terminals the four scores plus a random real number function. The function set included addition, subtraction, multiplication, and division. The fitness of alternative proposed fitness functions generated during the genetic search was computed the same way as for the linear combination fitness function. A population size of 1000 was used. And the training process continued for approximately one week, until subsequent generations didn't produce a function with performance better than the previous generation.

The two alternative fitness functions were evaluated two different ways. Interestingly, the simpler linear combination fitness function, which required only a fraction of the training time, performed slightly better than the genetic programming trained fitness function. Both fitness functions were first evaluated using the greatest common subsequence performance metric over a set of 100 test examples. The performance of the two was comparable. Whereas the linear combination fitness function achieved an average common substring length of 5.29 out of 10, the genetic programming trained fitness function achieved 4.72 out of 10.

Next they were evaluated in an end-to-end translation evaluation inside the Combination Mechanism on a set of 133 test sentences needing repair. It was found that the

[2]Greatest Common Subsequence was computed using Dijkstra's well known algorithm.

linear combination fitness function produced hypotheses of slightly higher quality, although in the total number of acceptable translations produced by the system was affected only slightly. ROSE using the linear combination trained fitness function produced two more acceptable translations than the genetic programming trained fitness function.

Neither of the trained fitness functions were able to rank the lists of sets of scores nearly the same way as the ideal fitness function. This is to be expected since the trained fitness function must make its decisions based on indirect evidence. However, although neither of these trained fitness functions were able to rank more than about 53% of the elements of each list correctly on average, they both performed well inside the Combination Mechanism. It can be concluded from this that it is not necessary for the trained fitness function to be extremely accurate in its ability to rank the lists of sets of scores. What appears to be the most important is that better hypotheses in general are ranked closer towards the top of the list while worse hypotheses in general are ranked closer to the bottom.

The problem of optimally combining multiple predictors is an unsolved problem in computer science, so the solutions I offer here are only two possible approaches, and certainly leave room for improvement. The evaluations presented in Chapter 10 were conducted using the genetic programming trained fitness function. The results presented there indicate that this genetic programming trained fitness function performs effectively. And as already stated above, the Combination Mechanism using this fitness function includes an acceptable hypothesis among its top ten hypotheses in 93.0% of the cases where it is possible to do so with the chunks produced by the parser. And in 84.6% of these cases, the best hypothesis is ranked first. Genetic search in general, similar to simulated annealing, offers the theoretical advantage over hillclimbing approaches that it avoids the problem of converging on locally optimal solutions. And since its set of possible target functions includes but is not limited to linear combinations, it would seem to have a greater potential for learning an effective function for combining the four provided pieces of information. It is for this reason that it offers an attractive approach to the problem of combining multiple predictors.

7.3.3.3 Applying the Fitness Function

The job of the Combination Mechanism is both to determine which fragments to include in the best hypotheses as well as how to combine the selected ones. The job of the fitness function is to rank alternative hypotheses in such a way that the best ranking hypotheses generated eventually contain the correct chunks fitted together in the correct way. In this section I demonstrate how to apply the genetic programming trained fitness function. Fitness functions trained with other techniques could be applied in the same way.

Look again at the example in Figure 7.4. The fitness value for this ideal hypothesis can be easily computed. The resulting structure has 6 concepts, one for each frame and atomic filler. In order to emphasize the difference between structures with different numbers of concepts, the number passed into the fitness function is 2 raised to the power of the number of concepts minus 1. So in this case, 32 is passed into the fitness function as the NUM_CONCEPTS parameter. Since selecting a chunk is considered a repair action as well as deciding how to combine them, this ideal program contains 3 steps. Each step is assigned a statistical score. The statistical score for selecting a chunk is equal to the statistical score assigned by the parser for the associated partial analysis. In this case, the parser assigned a score of 3.1634 to the chunk corresponding to "out" and 2.1224 to the chunk corresponding to "mornings". Statistical scores for combinations of chunks are computed by scaling the information gain between the type of chunk inserted and the slot in which it is inserted. This information gain score is scaled relative to the information gain scores for other potential slots such that sum of scaled information gain scores over all possible slots equals 1. Therefore, these scores are always between 0 and 1. Since the temporal expression could only be inserted into the `WHEN-O` slot in the chunk corresponding to "out", the statistical score for this repair is 1.0[3]. Therefore, the average statistical score for the repair actions involved is 2.0953. So this value is passed into the fitness function as the STAT_SCORE parameter. Since this repair covers 2 out of the 5 words in the sentence, the PERCENT_COV parameter will be equal to .4. The resulting fitness value for this ideal repair program is 0.1636. Low fitness values are preferred over higher ones, so this low value indicates that this repair hypotheses is good.

Consider the alternative hypotheses found in Figure 7.7. Notice that the fitness values assigned to these alternatives correctly rank them as less desirable than the ideal hypothesis. Given the four pieces of information the fitness function has to go by, however, the fitness function might be expected to have had a tough time ranking the ideal hypothesis over the first alternative displayed in Figure 7.7. However, the purpose of the trained fitness function is to balance these pieces of information effectively, and this example demonstrates well its ability to do so.

[3]In hindsight, it would have been better to pass the absolute connection strength rather than the normalized one. As it is, it is difficult for the fitness function to rank hypotheses correctly in the case where in one hypotheses a slot was selected because it was the only slot in the parent chunk where the child chunk could be inserted and in the other hypothesis, a slot was selected because it was the best one out of a set of alternatives. In this case, the first hypotheses, though it might actually be a worse choice, will always be preferred since it will receive a statistical score of 1.0. In order to avoid complications resulting from this in the evaluation, the `modifier` slot was removed from all temporal types in the interlingua specification since it is almost never used.

The first alternative hypothesis in Figure 7.7 corresponds to the interpretation, "Mornings and that are out." What is wrong with this hypothesis is that it includes the "that" chunk which in this case should be left out [4]. However, this hypothesis contains more frames than the ideal one, and since "that" is commonly used in scheduling dialogues to refer to time expressions, the structure corresponding to "that" fits reasonably into the `WHEN` slot. However, this first alternative contains two repair actions more than the ideal hypotheses, making it less desirable on one count. The purpose of the trained fitness function is to make trade-offs between the predictions made by the four pieces of information. In this case, the trained fitness function makes the correct trade-off, ranking the ideal hypothesis very slightly better. The similarity in the scores indicates that both of these hypotheses seem good, but the ideal case is slightly better.

The second hypothesis is more straightforward to rank as less desirable. In this second hypothesis, the Combination Mechanism attempted to insert the rejection chunk into the time expression chunk, the opposite of the ideal order. No slot could be found in the time expression chunk in which to insert the rejection expression chunk. In this case, the slot remains uninstantiated (indicated by `??` in Figure 7.7) and the largest chunk, in this case the time expression chunk, is returned. This hypothesis produces a feature structure that is indeed a portion of the correct structure, though not the complete structure. Since it clearly has fewer frames than the ideal structure, and since it covers a smaller portion of the sentence, it can straightforwardly be ranked as less fit. It is assigned a much larger fitness value than either of the other two alternatives.

[4] Though this would be a perfectly acceptable paraphrase into Pittsburghese, most speakers of Standard English would find this unacceptable.

Some Alternative Repair Hypotheses:

Hypothesis1: (Fitness Value: .16703)

```
(MY-COMB
   (MY-COMB
      ((FRAME *RESPOND))
      ((FRAME *SIMPLE-TIME) (TIME-OF-DAY MORNING)
       (NUMBER PLURAL) (SIMPLE-UNIT-NAME TOD))
      WHEN)
   ((FRAME *THAT) (ROOT THAT) (TYPE PRONOUN))
   WHEN)
```

Result1: Mornings and that are out.

```
((FRAME *RESPOND)
 (DEGREE NORMAL)
 (TYPE NEGATIVE)
 (WHEN (*MULTIPLE* ((FRAME *SIMPLE-TIME) (TIME-OF-DAY MORNING)
                    (NUMBER PLURAL) (SIMPLE-UNIT-NAME TOD))
                   ((FRAME *THAT) (ROOT THAT) (TYPE PRONOUN)))))
```

Hypothesis2: (Fitness Value: 2.7731)

```
(MY-COMB
  ((TIME-OF-DAY MORNING) (NUMBER PLURAL)
   (FRAME *SIMPLE-TIME) (SIMPLE-UNIT-NAME TOD))
  ((FRAME *RESPOND) (DEGREE NORMAL) (TYPE NEGATIVE))
  ??)
```

Result2: Mornings.

```
((FRAME *SIMPLE-TIME) (TIME-OF-DAY MORNING)
 (NUMBER PLURAL) (SIMPLE-UNIT-NAME TOD))
```

Figure 7.7: **Repair Hypotheses**

7.3.4 Other Parameters and Stopping Criteria

Parameters for the run include the method of selecting programs to "mate" with each other, and the crossover and mutation rate. They also the size of the population and the maximum number of generations. For any given problem, these parameters are generally determined experimentally. I found that four generations of population size 32 was reasonable for generating the most common types of repairs in a reasonable amount of time (on average, about 30 seconds of CPU time). I used fitness-proportionate reproduction, i.e., individuals were selected for reproduction with a frequency proportional to their fitness value. I limited the depth of programs to 15.

Note that the application of genetic programming in ROSE is strikingly different from most common applications of genetic programming which require populations on the order of thousands of individuals and months of run time before a sufficient number of individuals have been processed in order to converge upon an acceptable solution. The programs that generate repair hypotheses are much less complicated than the sorts of programs that solve the types of problems that are common applications of genetic programming. However, the genetic programming paradigm provides the right sort of opportunistic search environment required, since the system does not know a priori which repair strategy will prove most fruitful for any given set of chunks. The genetic programming algorithm samples a wide space of possibilities shallowly, and then pursues those strategies that appear to be most successful. Its use of statistical information to bias the search allows it to converge upon a reasonable solution more quickly than would otherwise have been possible with genetic search.

7.4 Chunk Formation

In order for the Combination Mechanism to produce high quality repair hypotheses, it must be presented with high quality partial analyses. The first stage in constructing a set of chunks to hand to the combination mechanism proper accesses the chart constructed inside of the parser. A greedy algorithm is used to pick out a set of adjacent chunks that span the chart. For example, in Figure 7.4, these are the chunks listed under Largest Chunks. In addition to these chunks, smaller chunks obtained by dissecting the parse trees associated with each of these chunks are also included. In Figure 7.4, these are the chunks listed under Derivative Chunks. It is important to include these smaller chunks as well because the larger chunks may contain both correct information as well as incorrect information. For example, the largest chunk under Largest Chunks in Figure 7.4 incorrectly

contains information about "June" which was a false start in the speaker's utterance. With the smaller chunks, it is possible to build the correct temporal expression, which does not include this incorrect information.

In order to repair utterances like this, it must be possible to include only correct information in the final repair hypothesis. One might wonder if it might be better to include only the smaller chunks, rather than both the larger and smaller chunks, but the larger ones are important to include as well. Though it is possible that part may be incorrect, nevertheless the parser is more likely to produce a correct analysis than the Combination Mechanism since the parser has more information (i.e., the parsing grammar). Which one is scored better in a particular case depends upon trade-offs made by the fitness function. And where the fitness function falls short of ranking hypotheses totally correctly, the Interaction with the User stage makes it possible to return the correct result in any case. It is best to leave both the largest chunks and the derivative chunks as alternatives.

While including a full set of chunks as described above is advantageous, it also has its disadvantages, which other portions of the combination mechanism proper must compensate for. The main problem is that it makes it more complicated to merge two repaired chunks because not only is it important to avoid including two copies of the same chunk in the final structure, it is necessary to avoid including chunks that overlap at all in terms of the portion of the sentence that they cover[5]. Also, the more terminal chunks there are, the larger the search space that is necessary in order to converge on a reliably good solution. In general, when the set of chunks is produced by the parser, the correctness and completeness of each chunk is not known. So it is not known a priori which of the set of chunks produced by the parser should be included in the final repair hypothesis. This is determined indirectly by having the repair module try different combinations of chunks. The goal of the fitness function is to obtain a better score for those programs in which the correct chunks are used in the correct way so that eventually the correct solution can be found.

The chunks provide building blocks for constructing the correct meaning representation of the sentence. Chunks that contain feature structures with a correct analysis contain portions of the ideal meaning representation. By putting these chunks together, a meaning representation without errors can be built. However, since portions of the sentence may lack a correct analysis in any of the chunks, the best result may end up being an incomplete meaning representation. There may still be gaps corresponding to those portions of the sentence that did not have a corresponding chunk with a correct analysis. And since

[5]It is possible to avoid including two chunks that cover the same portion of the sentence because each chunk keeps track of what portion of the sentence it covers.

some of the chunks are partially correct, if there aren't corresponding totally correct chunks, then the best feature structure might be one that contains all of the important information with a few minor errors. The best that can be done with the Combination Mechanism is to combine the chunks that are available into the most complete meaning representation that is possible with those chunks. Therefore, the only basic repair action necessary inside the Combination Mechanism is one that combines chunks the way the `MY-COMB` operator does.

Sentence: OR BETWEEN BEFORE TWO PM FRIDAY JUNE AUGUST FIFTH

Largest Chunks:

```
((ITEMS (*MULTIPLE*
        ((FRAME *INTERVAL)
         (END ((FRAME *SIMPLE-TIME) (HOUR 2)))
         (INCL-EXCL EXCLUSIVE))
        ((FRAME *SIMPLE-TIME) (DAY-OF-WEEK FRIDAY) (AM-PM PM) (HOUR 2))
        ((FRAME *SIMPLE-TIME) (MONTH 6))
        ((FRAME *SIMPLE-TIME) (MONTH 8) (DAY 6))))
 (FRAME *TIME-LIST)
 (CONNECTIVE -))
((TYPE ((FRAME *CONJUNCTION) (CONJUNCTION OR)))
 (FRAME *INTERJECT))
```

Derivative Chunks:

```
((ITEMS (*MULTIPLE*
        ((FRAME *INTERVAL)
         (END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
         (INCL-EXCL EXCLUSIVE))
        ((FRAME *SIMPLE-TIME) (DAY-OF-WEEK FRIDAY))
        ((FRAME *SIMPLE-TIME) (MONTH 6))))
 (TIME-LIST1 +) (FRAME *TIME-LIST) (CONNECTIVE -))
((ITEMS (*MULTIPLE*
        ((FRAME *INTERVAL)
         (END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
         (INCL-EXCL EXCLUSIVE))
        ((FRAME *SIMPLE-TIME) (DAY-OF-WEEK FRIDAY))))
 (TIME-LIST1 +) (CONNECTIVE -) (FRAME *TIME-LIST))
((END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
 (INCL-EXCL EXCLUSIVE) (FRAME *INTERVAL))
((FRAME *SIMPLE-TIME) (MONTH 8) (DAY 5))
((FRAME *SIMPLE-TIME) (DAY-OF-WEEK FRIDAY))
((FRAME *SIMPLE-TIME) (MONTH 6))
((FRAME *CONJUNCTION) (CONJUNCTION OR))
```

Figure 7.8: **Chunk Formation Example**

7.5 Initial Population Generation

Once the set of chunks constituting the terminal set has been constructed, the next step is to construct the initial population of programs. For each individual in the initial population, first a function is selected. In this case, there is only one function to choose from, so the *my-comb* operator is always selected. Then for each of the function's arguments, either a function or a terminal is selected. If a function is selected, the process continues recursively until all of the argument positions are filled. Though the *my-comb* operator is described as having three arguments, i.e., two chunks and a slot, the slot argument is implemented as a static variable. So only the chunk arguments are filled during the initial population generation stage.

7.5.1 Random Chunk Selection

Since both larger and smaller chunks are made available as terminal chunks for the combination mechanism, there are more chunks covering complicated portions of the sentence, such as large temporal expressions, than those covering other parts of the sentence. The problem with this is that the original genetic programming algorithm chooses terminal symbols with a uniform distribution. This means that it would be more likely to select a chunk from the complicated portion of the sentence. If the temporal expression is either moderately large or moderately complicated, temporal expression chunks overwhelm the combination mechanism, and no repairs involving the rest of the sentence are made.

To alleviate this problem, a new selection procedure was devised which selects a position in the sentence randomly with a uniform distribution. Once the position is selected, then all of the chunks covering that position in the sentence are collected, and one of them is selected randomly with a uniform distribution. In this way, every position in the sentence is highly likely to be represented in the final repair hypothesis.

7.5.2 Evaluating the Result of Hypotheses

Instantiating the chunk arguments of the `my-comb` operator is only part of the process of constructing a global meaning representation hypothesis. The next step is to determine how to combine the two chunks which are input to the `my-comb` operator. The more preferred way to combine the chunks is to insert the second chunk into a slot in the first chunk. When this is not possible, it attempts to merge the two chunks. And where this is not possible, the largest chunk is returned. The slot selection and merging operations are described below. Since the result of evaluating the `my-comb` operator is always a single

chunk, the result of evaluating each individual in the population is also a chunk made up of other chunks composed together.

7.5.2.1 Domain Independence: Making Use of the Interlingua Specification

```
Parent Chunk = ((frame *busy))

Child Chunk = ((frame *i))

(<BUSY> = ((frame *busy)
           (tense [TENSE])
           (aspect [ASPECT])
           (negative [VALUE+])
           (degree [DEGREE])
           (when-0 <*WHEN>)
           (who <*WHO>)
           (when <*WHEN>)
           (why <*EVENT>)
           (purpose <*EVENT>)
           (how-long <LENGTH>)))

(<I> = ((frame *i)))

(dominatesp [TENSE] <I>) = nil
(dominatesp [ASPECT] <I>) = nil
(dominatesp [VALUE+] <I>) = nil
(dominatesp [DEGREE] <I>) = nil
(dominatesp <*WHEN> <I>) = nil
(dominatesp <*WHO> <I>) = t
(dominatesp <*WHEN> <I>) = nil
(dominatesp <*EVENT> <I>) = t
(dominatesp <LENGTH> <I>) = nil

((frame *i)) fits in who, why, and purpose
```

Figure 7.9: **Insert Example**

As mentioned previously, the combination process is analogous in some ways to fitting pieces of a puzzle into a mold that contains receptacles for particular shapes. In this analogy, a meaning representation specification, described in depth in Appendix B, acts as the mold with receptacles of different shapes, making it possible to compute all of the

```
(<*WHO> = <I>
          <CLIENT>
          <DOCTOR>
          <EACH-OTHER>
          <SPOUSE>
          <FRIEND>
          <GIRLFRIEND>
          <BOYFRIEND>
          <PERSON>
          <PERSON-NAME>
          <PROFESSOR>
          <STUDENT>
          <THEY>
          <HE>
          <SHE>
          <WE>
          <YOU>
          <YOU-KNOW-WHO>)
```

Figure 7.10: **Subsumption Specification for Agent Types**

ways partial analyses can fit together in order to create a structure that is legal in this frame based meaning representation. It is ROSE's ability to make use of this specification rather than relying on hand-coded repair rules that makes ROSE a general solution. For example, Figure 7.9 displays ROSE computing each of the ways it is possible to insert the `*i` frame into a slot in the `*busy` frame. It first locates the rules in the interlingua representation which represent the `*busy` frame and the `*i` frame. It then tests each of the types associated with slots attached to the `*busy` frame to see which of these slots the `*i` frame can be inserted into. The `dominatesp` relation tests whether there is a rule or chain of rules such that the type that is the second argument can be an instantiation of the type that is the first argument. An example of the relevant type of interlingua rule is found in Figure 7.10. Based on the information gained from the interlingua specification using the `dominatesp` relation, it can be determined that the `*i` frame can be inserted into either the `who`, the `why`, or the `purpose` slot. In reality, the `who` slot is the only reasonable choice. But because the JANUS interlingua representation specification overgenerates, other possibilities are also indicated. The next section contains an explanation of how statistical information compensates for this weakness.

This interlingua representation specification allows ROSE to determine what the space of meaningful interlingua structures is. With this knowledge, it can constrain the space of repair hypotheses under consideration to those that will produce interlingua structures that are meaningful according to the specification. This meaning representation specification contains rules that specify types for classes of feature structures and the relationships between these types. It is the rules specifying relationships between types, as in Figure 7.10, that makes it possible to compute the `dominatesp` relation. The rules in this specification generate all of the possible feature structures in the meaning representation. The dependency on this resource makes the repair module customizable to any meaning representation by simply indicating which meaning representation specification it should use. Nowhere in the code is there anything specific to any particular meaning representation[6]. In particular, the `MY-COMB` operator uses the interlingua specification to compute the full set of slots associated with the type of the first chunk in which the second chunk can be inserted based on its type.

ROSE can be used straightforwardly in any domain where the meaning representation specification can be constructed using the same format as the scheduling domain ILT specification used in the context of this dissertation research. If the desired domain was such that a more elaborate meaning specification language was necessary for specifying the meaning representation, all that would need to change would be the code inside of ROSE which is dedicated to interpreting that specification. ROSE simply needs to be able to compute a type for a structure, determine based on that type what is the full set of slots associated with that type, and for each slot determine what are acceptable types for fillers.

7.5.2.2 Statistical Slot Guessing

Besides the interlingua representation specification, the combination mechanism makes use of statistical information stored in networks which compute connection strength between symbols in terms of information gain. The information gain is computed in terms of how much information is contributed about whether the associated output node is activated when the associated input node is activated. These networks are similar to the networks introduced in Chapter 4 which were used in the IRLH approach. During the development of ROSE it was found that information gain, a measure related to mutual information,

[6]The only exception is that there are patches added to the code in particular places to compensate for inconsistencies in how knowledge is represented in the ILT specification. Specifically, although most ILTs have a top level semantic frame, ILTs with sentence-type `*fragment` do not. Instead, one or more structures are inserted into slots that are not attached to any frame. ROSE converts fragments like this into a list of chunks, one corresponding to each slot with the filler of that slot as the structure. In order to convert all ILTs used as chunks inside ROSE to a consistent format, these anomalous fragment chunks must be detected. Thus a function for detecting fragment chunks was added to the code.

provided better biasing information for the search than mutual information which was used previously[7]. Information gain is more useful because it always evaluates to a positive number between 0 and 1. And thus, it is easier to compare scores for alternative choices. The genetic programming algorithm requires the fitness function to evaluate to a positive number. By using information gain rather than mutual information, it is easier to train the fitness function, which uses these scores as part of its input, to produce reasonable fitness values. Just as the networks described in Chapter 4, these networks store information about such things as connection strength between non-terminal symbols in the parsing grammar and types in the interlingua specification. They also compute connection strength between slots and likely slot fillers. This information is used to guide the genetic programming system so that when it makes random choices, any of the choices within the conceivable space of choices will be possible, but the more likely choices will be made more often.

The primary use of this statistical information is for selecting from the set of alternative possible slots in the first chunk parameter in which to insert the second chunk parameter. ROSE first computes the set of slots in the first chunk in which the second chunk can be inserted. Then, using a network that computes connection strength between slots for particular types and particular filler types, it computes the relative goodness of fit between the type of the second chunk and each respective slot. In the case of attempting to insert the `*i` frame into the `*busy` frame, we determined that either the `who`, the `why`, or the `purpose` slot would be possible slots. The network indicates that the goodness of fit with the `who` slot is 5.16E-4, the goodness of fit with the `why` slot is 2.0E-6, and the goodness of fit with the `purpose` slot is 1.0E-6. It then randomly selects a slot such that the probability of selecting any particular slot is proportional to its relative goodness of fit. In this case, the `who` slot will be selected 99.43% of the time. The `why` slot will be selected .38% of the time, and the `purpose` slot will be selected .19% of the time. The result is that while any of the possible slots might be selected, the more likely slots are chosen more often. In this way, the correct solution is likely to be converged upon sooner.

Doing the calculation of which slot was more likely, and by how much, is a computationally expensive task. First it involves computing the statistical score for each choice. Then it involves making a selection from among the weighted alternatives. In order to save time, since a decision from among the same set of choices occurs over and over in the process of letting the genetic programming system run its course, the first stage is only calculated once for each pair of types, i.e., one type for each chunk involved. The result of the first

[7]Because information gain was found to provide more useful information, these networks were also used for IRLH in the Interaction Evaluation presented in Chapter 10.

stage is then stored in a global list for future reference both within the current example and for processing subsequent examples.

7.5.2.3 The Merge Operation

The merge operation makes it possible to combine two chunks that have the same top level frame and each incorporate a different set of desirable repair actions. The merge process is not trivial since it is possible that two alternative hypotheses may insert the same chunk into different slots. If these two structures are merged in the most straightforward manner, the resulting structure will contain the same partial analysis in two different slots. See Figure 7.11 for an example. Here alternative 1 and alternative 2 both have the time expression chunk inserted somewhere. In alternative 1 it is inserted into the when slot. In alternative 2 it is inserted into the when slot inside of the why slot. When they are merged, two copies of the time expression chunk end up in the result.

The solution is not as simple as just deleting one of the duplicate time expressions. For example, if the time expression inserted in the when slot was removed, what would happen to the ((frame *out-of-town)) structure which modifies it? It couldn't be left where it is because if the time expression was deleted, then the slot where it is inserted would no longer exist. It could be deleted, but this would be a waste of the effort it took to insert the ((frame *out-of-town)) structure into the time expression structure. In circumstances like this, it is necessary for the merge operator to consider which copy is better to delete.

The first stage of the merge process is to determine what dependencies exist between the terminal chunks that make up the two alternative chunks being merged. This makes it possible to calculate how many terminal chunks will need to be left out of the final structure as a result of deleting a chunk they depend upon. See Figure 7.12.

The second stage is to determine how much of each of the two alternatives can automatically be included in the merged structure before any conflict resolution, as discussed above, needs to be done. This is computed by first sorting the lists of dependencies obtained in the previous stage so that all of the chunks that the current chunk depends upon will come earlier than it in the list. The lists in Figure 7.12 are already sorted. Once the lists are sorted, it is easy to find the appropriate set of dependencies that the two lists share. This is done by beginning at the front of the list, i.e., with (CH1 -> nil). The algorithm works through the list and selects those dependencies which both lists have and which depend only on chunks inside at least one of the dependencies already selected. In this case, that includes ((CH1 -> nil) (CH3 -> CH1)). Although both structures contain CH2, it does not

depend upon the same thing in both structures. An intermediate structure can be built then from this dependency list.

Next, the dependencies that have only chunks on the left hand side and that are not included in both lists and that depend only on chunks included in a dependency already selected either in this stage or in the previous one are then picked out. In the example, this includes only (CH5 -> C1). Therefore, chunk CH5 can then be added to the intermediate structure.

In the final stages, two lists are left for conflict resolution. In this example, both lists have CH2. But in the first case, CH2 has one chunk which depends on it where in the second list, CH2 doesn't have any other chunks depending on it. Therefore, it is better to select the CH2 from the first list because then two chunks instead of one can be included in the final structure.

```
CHUNKS

((frame *busy))
((frame *simple-time)
(day-of-week tuesday))
((frame *i))
((frame *out-of-town))
((frame *conference))

ALTERNATIVE 1

((frame *busy)
 (who ((frame *i)))
 (when
      ((modifier ((frame *out-of-town)))
       (frame *simple-time)
       (day-of-week tuesday))))

ALTERNATIVE 2

((frame *busy)
 (why
      ((frame *conference)
       (when ((frame *simple-time) (day-of-week tuesday))))))

RESULT FROM MERGING

((frame *busy)
 (who ((frame *i)))
 (when
      ((modifier ((frame *out-of-town)))
       (frame *simple-time)
       (day-of-week tuesday)))
 (why
      ((frame *conference)
       (when ((frame *simple-time) (day-of-week tuesday))))))

(who ((frame *i))))
```

Figure 7.11: **Merge Problem Example**

```
CHUNKS

CH1 = ((frame *busy))

CH2 = ((frame *simple-time) (day-of-week tuesday))

CH3 = ((frame *i))

CH4 = ((frame *out-of-town))

CH5 = ((frame *conference))

ALTERNATIVE 1

((frame *busy)
 (who ((frame *i)))
 (when
      ((modifier ((frame *out-of-town)))
       (frame *simple-time)
       (day-of-week tuesday))))

((CH1 -> nil) (CH3 -> CH1) (CH2 -> CH1) (CH4 -> CH2))

ALTERNATIVE 2

((frame *busy)
 (why
      ((frame *conference)
       (when ((frame *simple-time) (day-of-week tuesday))))))

((CH1 -> nil) (CH3 -> CH1) (CH5 -> CH1) (CH2 -> CH5))

CHUNKS SHARED BY THE TWO ALTERNATIVES:

CH1, CH3, and CH2
```

Figure 7.12: **Dependencies**

```
AFTER STAGE TWO

((CH1 -> nil) (CH3 -> CH1))

((frame *busy)
 (who ((frame *i))))

AFTER STAGE 3

((CH5 -> CH1))

((frame *busy)
 (who ((frame *i)))
 (why ((frame *conference))))

FINAL STAGES

((CH2 -> CH1) (CH4 -> CH2))

((CH2 -> CH5))

result: ((CH2 -> CH1) (CH4 -> 2))

final struct:

((frame *busy)
 (who ((frame *i)))
 (when
      ((modifier ((frame *out-of-town))
       (frame *simple-time)
       (day-of-week tuesday))))
 (why ((frame *conference))))
```

Figure 7.13: **Intermediate Stages**

7.6 Generating Subsequent Generations

Usually the programs in the initial population do not produce acceptable repair hypotheses, although they do tend to produce chunks made by composing smaller chunks together. But some programs do a better job than others. Those which are relatively more fit are selected for reproduction. As in the standard Genetic Programming algorithm, *Crossover* and *Mutation* are used to produce the next generation from the most fit programs from the current generation. The *Crossover* operation swaps a subprogram producing a chunk from one parent program with a subprogram of the other parent program producing a different chunk. The *Mutation* operator swaps a chunk argument with a randomly chosen chunk or randomly constructed subprogram producing a chunk. Once these operations have constructed the new population, the programs are evaluated to produce a new result and are assigned a new fitness value. This process continues for 4 generations.

7.7 Returning the Final Result

After this final generation of repair hypotheses have been evaluated, the top ten best ranked hypotheses are returned. Since the parsing grammar allows the parser to analyze a sentence as a list of ILTs, but the Combination Mechanism only has the ability to repair a single ILT, if there are remaining chunks after repair which have their corresponding partial parse dominated by a <start> symbol, they are returned along with the repaired ILT. Just as the parser returns a list of ILTs, each hypothesis returned from the Combination Mechanism is a list of ILTs that includes the one repaired ILT. A good example of this can be found in the alternative hypotheses constructed for Example 4 in Chapter 6 (Figure 6.7).

Chapter 8

The Interaction Mechanism

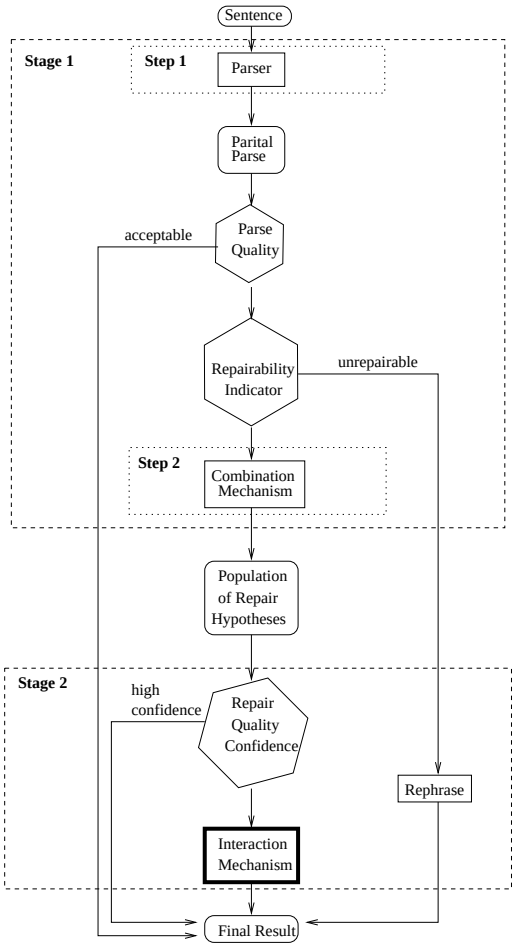

Figure 8.1: **Overview of the ROSE Interpretation Process**

As discussed in Chapter 6, the ROSE approach, displayed in Figure 8.1, interprets extragrammatical input in two stages. ROSE's first stage, Repair Hypothesis Formation, is responsible for assembling a set of hypotheses about the meaning of the ungrammatical utterance. In ROSE's Interaction with the User stage, the system generates a set of queries,

negotiating with the speaker in order to narrow down to a single best meaning representation hypothesis.

Approaches that do not rely on interaction with the user to guide its understanding make the implicit assumption that the system has sufficient information for concluding this search successfully. It has been well established (Clark and Wilkes-Gibbs, 1986; Clark and Schaefer, 1989), however, that even humans do not possess sufficient knowledge for this task and that if not given the opportunity to collaborate with their fellow conversational participants in order to agree upon an interpretation for an utterance, the level of understanding is reduced.

Because of this phenomenon, I rely on focused interaction with the speaker in the final stages of searching for the correct meaning representation of the speaker's utterance. Thus, responsibility for achieving robustness is distributed between the speaker and the system. Inspired by (Clark and Wilkes-Gibbs, 1986; Clark and Schaefer, 1989), the goal of the Interaction Mechanism is to minimize collaborative effort between the system and the speaker while maintaining a high level of interpretation accuracy. It uses this principle in determining which portions of the speaker's utterance to question. Thus, it focuses its interaction on those portions of the user's meaning that it is particularly uncertain about. In its questioning, it attempts to display the state of the system's understanding, acknowledging information conveyed by the speaker as it becomes clear.

8.1 Components of Interaction

The Interaction Mechanism is responsible for deciding which, if any, questions to ask the user and then using the information gained from this to select from among the competing alternative structures for the one best representing what the speaker has said. The interaction process can be summarized as follows:

1. Extract Distinguishing Features
2. Loop:
   (a) Select a Feature
   (b) Generate Query Text Based on Selected Feature
   (c) Update List of Alternative Structures Based on User's Answer
   (d) Update List of Distinguishing Features Based on Remaining Distinguishing Features

3. Return Best Result

The Interaction Mechanism first assesses the state of its understanding of what the speaker has said by extracting features that distinguish the top set of hypotheses from one another. It then cycles through a four step process of first selecting a question based on its set of distinguishing features and then updating its set of alternatives and distinguishing features based on the user's response.

8.1.1 Assessment of Understanding

The first task of the Interaction Mechanism is to determine what the system knows about what the speaker has said and what it is not certain about. It does this by comparing the top set of repair hypotheses and extracting a set of features that distinguish them from one another. These distinguishing features form the basis for the queries that the Interaction Mechanism generates.

8.1.1.1 Using Features

EXAMPLE FEATURES OF LENGTH ONE:

((f *busy))
distinguishes structures that contain the *busy frame from
those that do not

((f *simple-time))
distinguishes structures that contain the *simple-time frame
from those that do not

((f wednesday))
distinguishes structures that contain the atomic filler
wednesday from those that do not

Figure 8.2: **Features of Size One**

The features used in ROSE to distinguish alternative meaning representation structures from one another specify paths through the compositional meaning representation. Its recursive structure is made up of frames with slots that can be filled either with other frames or with atomic fillers. These compositional structures can be thought of as trees, with the top level frame being the root of the tree and branches attached through slots. The

((f *busy) (s who) (f *i))
has the *i frame in the who slot of the *busy frame

((f *free) (s when) (f *simple-time) (s day-of-week) (f wednesday))
has wednesday as the filler of the day-of-week slot of the
simple-time frame which is a filler of the when slot of the *free frame

Figure 8.3: **Odd Lengthed Features**

distinguishing features that are extracted are always anchored in a frame or atomic filler. When a feature is applied to a meaning representation structure, a value is obtained. In this way, features can be used to assign meaning representation structures to classes according to what value is obtained for each when the feature is applied. The simplest of features, features of length one, can be used to distinguish structures that contain somewhere a particular frame or atomic filler from structures that do not. They evaluate to T when applied to structures containing the frame or atomic filler and nil otherwise. See Figure 8.2 for examples of features of length one. Notice that each feature is a list containing a single pair. The first element of the pair, `f`, indicates that the second element in the pair is either a frame or atomic filler.

Longer features describe more structure. Odd lengthed features represent slot fillers. See Figure 8.3 for examples of odd lengthed features. Notice that each odd lengthed feature is a list consisting of an odd number of pairs. Every other pair begins with `f` indicating that the second element in the pair is a frame or atomic slot filler. Alternate pairs begin with `s`, indicating that the second element in the pair is the name of a slot associated with the last frame mentioned. When these odd lengthed features are applied to meaning representation structures, they can have as their value either `T` or `nil` . They would evaluate to `T` for meaning representation structures that have the specified filler in the specified slot and `nil` otherwise. See Figure 8.4 for a set of examples. In each example, the value of the feature `((f *busy) (s who) (f *i))` is given for the associated structure. The value for the structures in examples 1 and 4 is `T` because each of these structures include the `*busy` frame and in both cases, the `who` slot of the `*busy` frame has the frame `*i` as its filler. In example 2, the `who` slot of the `*busy` frame is filled with a different frame. In example 3, the `*busy` frame does not have a filler in the `who` slot. And example 5 does not even have a `*busy` frame.

Whereas odd lengthed features represent slot fillers, even lengthed features represent slots. See Figure 8.5 for examples of even lengthed features.

When features that specify a slot are applied to meaning representation structures, the value is equal to the text generated from the filler of the associated slot in the associated meaning representation structure. Take the feature `((f *busy) (s who))` as an example. It specifies the filler of the `who` slot of the `*busy` frame. Its value is a string containing the generated text from the filler. It would have as a value "I" for a feature structure that has the `*i` frame as a filler of the `who` slot of the `*busy` frame as in examples 1 and 4 in Figure 8.4. It would be "YOU" for a feature structure that has frame `*you` filling that `who` slot as in example 2. And it would be "nil" for feature structures that either did not have a `*busy` frame, such as example 5, or that had a `*busy` frame but the `who` slot was not filled in as in example 3.

Thus, these features are useful because they can be used to divide the set of alternative meaning representation hypotheses into equivalence classes. Features that do not divide the search space represent substructures that all of the alternative hypotheses share, and thus substructures that ROSE is relatively confident about. Features that divide the search space represent substructures that are not shared and thus need to be questioned since the system is not confident about them. Each single feature corresponds to a question that the system can ask. If the user confirms a particular value for one of the features that divide the search space, the search space can be reduced accordingly. Ideally, through a series of questions, the Interaction Mechanism can narrow down to the correct feature structure.

Feature: ((f *busy) (s who) (f *i))

Example 1:
Structure:
((frame *busy)
 (who ((frame *i)))
 (when ((frame *simple-time) (day-of-week thursday))))
Value: T

Example 2:
Structure:
((frame *busy)
 (who ((frame *you)))
 (when ((frame *simple-time) (day-of-week thursday))))
Value: nil

Example 3:
Structure:
((frame *busy)
 (when ((frame *simple-time) (day-of-week thursday))))
Value: nil

Example 4:
Structure:
((frame *simple-time)
 (day-of-week thursday)
 (modifier ((frame *busy) (who ((frame *i))))))
Value: T

Example 5:
Structure:
((frame *simple-time)
 (day-of-week thursday))
Value: nil

Figure 8.4: **Odd Lengthed Features and Values**

EXAMPLE FEATURES:

((f *busy) (s who))
the filler of the who slot in the *busy frame

((f *simple-time) (s modifier) (f *busy) (s who))
the filler of the who slot in the *busy frame which is a filler
in the modifier slot of the *simple-time frame

Figure 8.5: **Even Lengthed Features**

8.1.1.2 Extracting Features

Sentence: HI HOW +BOUT MEETING ANY TIME < BEFORE TWO PM
Words Skipped: HI < BEFORE TWO PM
Statistical Score: 9.079212043065768
Parse Quality: Bad
Parser's Analysis:
((SENTENCE-TYPE *QUERY-REF)
 (WHAT ((FRAME *MEETING)))
 (WHEN ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME)))
 (FRAME *HOW))
Interpretation: HOW ABOUT MEETING ANY TIME
Repairability: Yes
Chunks:
((FRAME *HOW)
 (WHEN ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME)))
 (WHAT ((FRAME *MEETING))))
((END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
 (INCL-EXCL EXCLUSIVE)
 (FRAME *INTERVAL))
((UNIT-NAME *TIME) (FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2))
((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME))
((FRAME *HOW) (WHAT ((FRAME *MEETING))))
((FRAME *GREET) (TYPE ((FRAME *HELLO))))
((FRAME *HOW))
((FRAME *MEETING))
((FRAME *HELLO))

Figure 8.6: **Interaction Example: Parsing Stage**

Remember that each hypothesis returned from the Combination Mechanism is a list of one or more meaning representation structures. The first step in extracting the set of features for a given set of hypotheses is to build a forest of trees out of the largest meaning representation structure from each of the hypotheses. Consider the example from Chapter 6 which is redisplayed here in Figure 8.6 and Figure 8.7. Five alternative repair hypotheses are listed, and the third one is the most desirable one.

At the topmost level, feature structures are divided into groups according to their top level frame. These groups are mutually exclusive. Each feature structure must be a member of exactly one of these groups. From there, each group is divided along several

orthogonal lines according to their fillers for particular slots. For example, assume that one top level group has feature structures that have as their top level frame `*how`. One possible classification of this set would divide these feature structures into smaller classes according to what the filler of the `when` slot is. Another orthogonal classification would divide them into classes according to the frame filling the `what` slot. The same feature structure will be a member of more than one set at this level. For example, it could both be in the set of feature structures with `*meeting` as a filler of the `what` slot and the set of structures with `*interval` filling the `when` slot. There may also be further divisions of these classes into smaller classes. For example, the `*interval` frame may have fillers of its own. So the same sort of classification can take place recursively until each leaf class has only one feature structure or contains a set of feature structures such that for each of them the portion associated with the classification is identical.

Once this forest is built, features can be extracted by tracing through all possible paths from a root to a leaf. Each path from a root to a leaf is a distinct feature. The resulting list of features for the example in Figure 8.7 can be found in Figure 8.8.

```
Alternative Repair Hypotheses:

"HELLO – HOW ABOUT IF MEET ANY TIME – BEFORE TWO P.M."
(((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((ATTITUDE *HOW-ABOUT)
 (WHEN ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME)))
 (FRAME *MEET))
((WHEN ((END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
 (INCL-EXCL EXCLUSIVE)
 (FRAME *INTERVAL)))))

"HELLO – HOW ABOUT MEETING TWO P.M. ANY TIME – BEFORE"
(((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((WHAT ((FRAME *MEETING)))
 (WHEN (*MULTIPLE* ((UNIT-NAME *TIME) (FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2))
                   ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME))))
 (FRAME *HOW))
((WHEN ((FRAME *SPECIAL-TIME) (NAME BEFORE)))))

"HELLO – HOW ABOUT MEETING BEFORE TWO P.M. ANY TIME "
(((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((WHAT ((FRAME *MEETING)))
 (WHEN (*MULTIPLE* ((END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
                    (INCL-EXCL EXCLUSIVE) (FRAME *INTERVAL))
                   ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME))))
 (FRAME *HOW)))

"HELLO – HOW ABOUT BEFORE TWO P.M. MEETING ANY TIME"
(((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((WHAT (*MULTIPLE* ((END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
                    (INCL-EXCL EXCLUSIVE) (FRAME *INTERVAL))
                   ((FRAME *MEETING))))
 (WHEN ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME)))
 (FRAME *HOW)))

"HOW ABOUT – HELLO – AFTER ANY TIME TO MEET BEFORE TWO P.M."
(((FRAME *HOW))
((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((START ((PURPOSE ((FRAME *MEET) (VERB-FORM ING)))
                   (FRAME *SPECIAL-TIME) (NAME TIME) (SPECIFIER ANY)))
          (FRAME *INTERVAL)
          (INCL-EXCL EXCLUSIVE)
          (END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))))
```

Figure 8.7: **Interaction Example: Alternative Repair Hypotheses**

((F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S PURPOSE) (F *MEET))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *HOW) (S WHEN) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *HOW) (S WHEN) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHEN) (F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *HOW) (S WHEN) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *HOW) (S WHEN) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHAT) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *HOW) (S WHAT) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHAT) (F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *HOW) (S WHAT) (F *MEETING))
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *MEET) (S ATTITUDE) (F *HOW-ABOUT))

Figure 8.8: **Initial Features**

8.1.1.3 Multiplying Features

((F *HOW-ABOUT))
((F *MEET) (S WHEN))
((F *MEET) (S WHEN) (F *SPECIAL-TIME))
((F *MEETING))
((F *HOW) (S WHAT))
((F *HOW) (S WHAT) (F *INTERVAL))
((F *HOW) (S WHAT) (F *INTERVAL) (S END))
((F *HOW) (S WHAT) (F *INTERVAL) (S END) (F *SIMPLE-TIME))
((F *HOW) (S WHEN) (F *SIMPLE-TIME))
((F *HOW) (S WHEN) (F *SPECIAL-TIME))
((F *HOW))
((F *HOW) (S WHEN))
((F *HOW) (S WHEN) (F *INTERVAL))
((F *HOW) (S WHEN) (F *INTERVAL) (S END))
((F *HOW) (S WHEN) (F *INTERVAL) (S END) (F *SIMPLE-TIME))
((F EXCLUSIVE))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S PURPOSE))
((F *SPECIAL-TIME) (S PURPOSE))
((F *MEET))
((F *SPECIAL-TIME) (S PURPOSE) (F *MEET))
((F *INTERVAL) (S START))
((F *INTERVAL) (S START) (F *SPECIAL-TIME))
((F 2))
((F *SIMPLE-TIME) (S HOUR) (F 2))
((F *INTERVAL))
((F *INTERVAL) (S END))
((F *INTERVAL) (S END) (F *SIMPLE-TIME))
((F *SIMPLE-TIME))
((F PM))

Figure 8.9: **Distinguishing Features for Example with 5 Alternatives (Part 1)**

More general features are also useful for deriving different segmentations on the set of alternative meaning representation hypotheses. These more general features can be constructed by shortening the original features from either end. For example, the more general feature `((F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM))` can be constructed by shortening the feature `((F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))` . The only constraint on features is that they must begin by specifying a frame or an atomic filler, not a slot. It does not have to begin with the top level frame. So `((F *SIMPLE-TIME)`

((F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *MEET) (S ATTITUDE) (F *HOW-ABOUT))
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *HOW) (S WHAT) (F *MEETING))
((F *HOW) (S WHAT) (F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *HOW) (S WHAT) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHAT) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *HOW) (S WHEN) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHEN) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *HOW) (S WHEN) (F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *HOW) (S WHEN) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHEN) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S PURPOSE) (F *MEET))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))

Figure 8.10: **Distinguishing Features for Example with 5 Alternatives (Part 2)**

`(S AM-PM) (F PM))` is another possible feature that can be derived from `((F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))`.

Once the full set of features has been constructed, some of them can be eliminated. First, all of the duplicates are weeded out. There is no need to include more than one copy of the same feature since it would correspond to the same question, and there is no need to ask the user the same question twice. Next, all of the ones that do not partition the set of alternative interlingua structures are eliminated. If the feature does not partition the search space, the answer to the corresponding question will not help narrow down the set of alternative structures and thus can safely be eliminated. The reamining features for the example in Figure 8.7 after this process has taken place are found in Figure 8.9 and Figure 8.10.

8.1.2 Generating a Question

Once the system has narrowed down on the set of features worth asking about, it must determine which order to ask the questions in. Besides eliminating questions which it does not need to ask, part of minimizing collaborative effort is asking more informative questions first in order to keep the number of questions that need to be asked at a minimum. Each time the Interaction Mechanism generates a query, it selects a feature from the list of distinguishing features such that the feature is:

- **Askable:** It possible to ask a natural question from it.
- **Evaluatatable:** It refers to a single repair or set of repairs that always occur together.
- **In Focus:** It involves information from the common ground.
- **Most Informative:** It is likely to result in the greatest search space reduction.

8.1.2.1 Narrowing Down to Askable Questions

First, the set of features is narrowed down to those features which represent askable questions. In particular, some wh-questions are not askable. For example, wh-questions that presuppose something that the ROSE system does not know yet are not askable. What this means in terms of features is that every odd-lengthed initial segment of the feature corresponding to the wh-question should have value T for every alternative hypothesized structure. It is awkward to generate a wh-question based on a feature of length greater than two. For example, a question corresponding to `((F *HOW) (S WHAT) (F *INTERVAL) (S END))` might be phrased something like "How is the time ending when?". So even-lengthed features more than two elements long are also eliminated at this stage. This does not hinder the system's ability to distinguish between alternative hypothesized structures since some set of Yes/No questions must also exist in the set that result in the same segmentation. In this example, however, all of the wh-questions are eliminated for one of the above listed reasons. For example, `((F *HOW) (S WHEN))`, i.e., "How about when?", is eliminated because it presupposes that the sentence is asking "How about" something. And although all of the alternative hypotheses generate text including this substring, not all of the alternative hypotheses include the `*how` frame. So this question is filtered out. Wh-questions that are equivalent to a Yes/No question in the set are also eliminated. For example, `((F *INTERVAL) (S START))` is equivalent to `((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S NAME) (F TIME))` since whenever the `START` slot of the `*INTERVAL` frame is filled in the set of alternatives it is always filled with the same filler. Both of these

would correspond to the question, "Was something like STARTING ANY TIME part of what you meant?"

In the Clark et al. model the collaboration between the speaker and the listener is dependent on the listener making his state of understanding clear to the speaker. One natural way that ROSE accomplishes this is by encoding as much common ground knowledge in its questions as possible. In order to do this, it prefers to use features that overlap with structures that all of the alternative hypotheses have in common. The structures that all of the alternative hypotheses share are called non-controversial structures. As the negotiation continues, these tend to be structures that have been confirmed through interaction. This has the affect of having questions tend to follow in a natural succession. For the five hypotheses in Figure 8.7, the only substructure the largest chunk in each hypothesis shares with the other largest chunks is the structure corresponding to "any time". So any query generated at this point in the negotiation will contain the phrase "any time" in it. The relevant features, called in-focus features, are found in Figure 8.12.

Those features that are filtered out at this stage are not eliminated entirely. They are reconsidered again the next time a question is selected. Because the determination of which features represent askable questions depends upon the current set of alternatives, a question that is not currently askable may be askable once the set of alternatives has been modified based on interaction with the user. The features that remain after this filtering process for the example in Figure 8.7 can be found in Figure 8.11.

((F *HOW-ABOUT))
((F *MEET) (S WHEN) (F *SPECIAL-TIME))
((F *MEETING))
((F *HOW) (S WHAT) (F *INTERVAL))
((F *HOW) (S WHAT) (F *INTERVAL) (S END) (F *SIMPLE-TIME))
((F *HOW) (S WHEN) (F *SIMPLE-TIME))
((F *HOW) (S WHEN) (F *SPECIAL-TIME))
((F *HOW))
((F *HOW) (S WHEN) (F *INTERVAL))
((F *HOW) (S WHEN) (F *INTERVAL) (S END) (F *SIMPLE-TIME))
((F EXCLUSIVE))
((F *MEET))
((F *SPECIAL-TIME) (S PURPOSE) (F *MEET))
((F *INTERVAL) (S START) (F *SPECIAL-TIME))
((F 2))
((F *SIMPLE-TIME) (S HOUR) (F 2))
((F *INTERVAL))
((F *INTERVAL) (S END) (F *SIMPLE-TIME))
((F *SIMPLE-TIME))
((F PM))
((F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *MEET) (S ATTITUDE) (F *HOW-ABOUT))
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *HOW) (S WHAT) (F *MEETING))
((F *HOW) (S WHAT) (F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *HOW) (S WHAT) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHAT) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *HOW) (S WHEN) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHEN) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *HOW) (S WHEN) (F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *HOW) (S WHEN) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHEN) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S PURPOSE) (F *MEET))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))

Figure 8.11: **Askable Questions**

Non-Controversial Substructures:
((FRAME *SPECIAL-TIME) (NAME TIME) (SPECIFIER ANY))

Features Overlapping with Non-Controversial Substructures:
((F *MEET) (S WHEN) (F *SPECIAL-TIME))
((F *HOW) (S WHEN) (F *SPECIAL-TIME))
((F *SPECIAL-TIME) (S PURPOSE) (F *MEET))
((F *INTERVAL) (S START) (F *SPECIAL-TIME))
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S PURPOSE) (F *MEET))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S NAME) (F TIME))

Figure 8.12: **In Focus Features**

8.1.2.2 Narrowing Down to Evaluatable Questions

((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S PURPOSE) (F *MEET))
((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))
((F *INTERVAL) (S START) (F *SPECIAL-TIME))
((F *SPECIAL-TIME) (S PURPOSE) (F *MEET))
((F *HOW) (S WHEN) (F *SPECIAL-TIME))
((F *MEET) (S WHEN) (F *SPECIAL-TIME))

Figure 8.13: **Acceptable Questions**

In order for a Yes/No question to be evaluatable, it must confirm only a single repair action. Otherwise, if the user responds with "No." it cannot be determined whether the user is rejecting both repair actions or only one of them. If the same two repair actions always occur together among the set of alternative hypotheses, then it is not a problem. But if they don't, then only hypotheses which include both actions will be eliminated if the user responds with no. Then the system may have to ask about each of the actions involved separately. If the user thinks he already answered that question, he might become frustrated with this. So it is better to restrict the questions asked to ones which constitute a single question. In the case of our example, the case where a feature represents two different repair actions that did not always occur together did not occur. But one can imagine a feature like **((F *INTERVAL) (S START) (F *SPECIAL-TIME) (S PURPOSE) (F *MEET) (S WHO) (F *I))** which would indicate that the frame `*I` was sometimes the filler of the `WHO` slot of the `*MEET` frame which was the filler of the `PURPOSE` slot of the `*SPECIAL-TIME` frame and so forth. In this case, this feature would be eliminated. But separate features would be included to indicate, for example, that `*I` was the filler of the `WHO` slot of the `*MEET` frame and separately that `*MEET` was the filler of the `PURPOSE` slot of the `*SPECIAL-TIME` frame.

Filtering out features which represent two repairs does not hinder the system's ability to distinguish between alternative hypothesized structures since shorter features corresponding to single repairs also always exist within the set of alternative features. So

one or more shorter features considered together can make the same distinction as the larger feature that will be filtered out at this stage.

Again, those features eliminated here are reconsidered again for subsequent questions. For the example in Figure 8.7, the remaining features after this filtering process has taken place can be found in Figure 8.13. In this case, it is the same set as that found in Figure 8.11.

8.1.2.3 Selecting the Most Informative Question

((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))	2.4
((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))	2.4
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))	1.6
((F *MEET) (S WHEN) (F *SPECIAL-TIME) (S SPECIFIER) (F ANY))	1.6
((F *HOW) (S WHEN) (F *SPECIAL-TIME))	2.4
((F *MEET) (S WHEN) (F *SPECIAL-TIME))	1.6

Figure 8.14: **Search Reduction of Features Distinguishing Between Top Two Hypotheses**

Because the set of alternative hypothesized structures are sorted based on their fitness, and because the fitness function is good enough that the best structure is normally one of the top two hypotheses, if any of the remaining features distinguish between the first two alternatives, the most informative feature is selected from this subset, otherwise the entire remaining set is used. The features in Figure 8.14 are ones which distinguish between the top two hypotheses in Figure 8.7.

The final piece of information used in selecting between those remaining features is the expected search reduction. The expected search reduction indicates how much the search space can be expected to be reduced by once the answer to the corresponding question is obtained from the user. Since it is possible that the best hypothesized structure is not one of the first two alternatives, in order to keep the number of necessary questions down, it is preferable to eliminate as many alternative hypothesized structures with a single question as possible. The number next to each feature in Figure 8.14 indicates the associated expected search reduction.

Expected search reduction is calculated by the following formula:

$$S_f = \sum_{i=1}^{n_f} (\frac{l_{i,f}}{L}) \times (L - l_{i,f}) \tag{8.1}$$

Equation 8.1 is for calculating S_f, the expected search reduction of feature number f. L is the number of alternative interlingua structures. In the current example, the number of alternatives is five, displayed in Figure 8.7. n_f is the number of equivalence classes imposed by feature f. Take the feature `((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))` as an example. It has two associated classes since it is a yes/no feature. $l_{i,f}$ is the number of alternative interlingua structures in the ith equivalence class of feature f. The first class of interlingua structures associated with this feature is the set of structures where the value for this feature is `T`. This includes the second, third, and fourth alternatives in Figure 8.7. So the size of this class is 3. The second class contains those structures where the value of this feature is `nil`. This includes the first and fifth structures in Figure 8.7. If the value for feature f associated with the class of length $l_{i,f}$ is the correct value, $l_{i,f}$ will be the new size of the search space. In this case, the actual search reduction will be the current number of alternative interlingua structures, L, minus the number of alternative interlingua structures in the resulting set, $l_{i,f}$. Intuitively, the expected search reduction of a feature is the sum over all of a feature's equivalence classes of the percentage of interlingua structures in that class times the reduction in the search space if the associated value for that feature turns out to be correct. In this case, $((.40 * (5-2)) + (.60 * (5-3)))$, or 2.4.

After the expected search reduction is calculated for each feature, the feature with the highest expected search reduction is selected. The strategy of selecting the feature with the highest expected search space reduction is not guaranteed to be the best strategy in all cases. But it is analogous to a binary search. A binary search divides the search space in half on each iteration. It has the best average case complexity of any of the search algorithms in its class[1]. But it is not guaranteed to be the fastest search algorithm in every specific case. For example, no one would argue that a linear search would be the best strategy for any sizable list. But if the element you are searching for happens to be the first element in the list, then the linear search will be faster in that case than the binary search although the binary search has a better complexity. Since one never knows ahead of time where in the list of elements is the desired one, the binary search is preferred. There can never be a way to guarantee that one has the best strategy for asking questions because what the best strategy is always depends upon information which is not known yet. If this were not the case, there would be no need to ask any questions of the user.

Selected Feature:
(((F *HOW) (S WHEN) (F *SPECIAL-TIME) (S NAME) (F TIME))

Non-controversial Structs if Answer to Question is Yes:
((FRAME *HOW) (WHEN ((FRAME *SPECIAL-TIME) (NAME TIME) (SPECIFIER ANY)))))

Question Structure:
((FRAME *HOW) (WHEN ((FRAME *SPECIAL-TIME) (NAME TIME) (SPECIFIER ANY)))))

Question Text:
Was something like HOW ABOUT ANY TIME part of what you meant?

Figure 8.15: **Question Text Generation Example**

8.1.2.4 Generating the Text

The selected feature is used to generate a query for the user. Queries are based on the selected feature, but in some cases the feature does not specify enough information to include in the meaning representation structure used for generating the question. Additionally, the interlingua representation specification itself does not indicate which slots are necessary to fill and which are not in order to generate natural sounding text. So ROSE attempts to include all non-controversial substructures overlapping the selected feature. These are the substructures which all of the alternative structures have in common. If the question is a Yes/No question, it includes all of the substructures which would be non-controversial assuming the answer to the question is Yes. Once the set of non-controversial structures is extracted, a skeleton structure is built based on the feature. It's top level semantic frame is determined based in the root of the feature. If it has length greater than two, the algorithm is called recursively, inserting the result of calling it on the CDDR of the features into the slot indicated by the CADR of the feature. If it has length greater than one, a wh-structure is inserted into the slot indicated by the CADR of the feature. Otherwise the process is complete. The non-controversial substructures are used to flesh out the skeleton structure. Since information confirmed by the previous question is now considered non-controversial, the result of the previous interaction is made evident in how the current question is phrased. An example of a question generated with this process can be found in Figure 8.15.

[1] Binary search has logarithmic time complexity.

If the selected feature corresponds to a wh-question, i.e., if it is an even lengthed feature, the sentence type *query-ref is inserted into the resulting structure and the text is generated. Otherwise the sentence type *state is inserted into the top and the generated text is inserted into the following formula: "Was something like XXX part of what you meant?", where XXX is filled in which the generated text[2], as in Figure 8.15. The set of alternative answers based on the set of alternative hypotheses is presented to the user. For wh-questions, a final alternative, "None of these alternative are acceptable", is made available.

8.1.3 Processing the Response

```
"HELLO – HOW ABOUT MEETING TWO P.M. ANY TIME – BEFORE"
(((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((WHAT ((FRAME *MEETING)))
 (WHEN (*MULTIPLE* ((UNIT-NAME *TIME) (FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2))
                   ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME))))
 (FRAME *HOW))
((WHEN ((FRAME *SPECIAL-TIME) (NAME BEFORE)))))

"HELLO – HOW ABOUT MEETING BEFORE TWO P.M. ANY TIME "
(((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((WHAT ((FRAME *MEETING)))
 (WHEN (*MULTIPLE* ((END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
                    (INCL-EXCL EXCLUSIVE) (FRAME *INTERVAL))
                   ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME))))
 (FRAME *HOW)))

"HELLO – HOW ABOUT BEFORE TWO P.M. MEETING ANY TIME"
(((TYPE ((FRAME *HELLO))) (FRAME *GREET))
((WHAT (*MULTIPLE* ((END ((FRAME *SIMPLE-TIME) (AM-PM PM) (HOUR 2)))
                    (INCL-EXCL EXCLUSIVE) (FRAME *INTERVAL))
                   ((FRAME *MEETING))))
 (WHEN ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME)))
 (FRAME *HOW)))
```

Figure 8.16: **Remaining Hypotheses**

[2]Text is generated in the JANUS system using the GENKIT software package and a generation grammar.

((F *HOW) (S WHAT))
((F *HOW) (S WHAT) (F *INTERVAL))
((F *HOW) (S WHAT) (F *INTERVAL) (S END))
((F *HOW) (S WHAT) (F *INTERVAL) (S END) (F *SIMPLE-TIME))
((F *HOW) (S WHEN) (F *SIMPLE-TIME))
((F EXCLUSIVE))
((F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHEN))
((F *HOW) (S WHEN) (F *INTERVAL))
((F *HOW) (S WHEN) (F *INTERVAL) (S END))
((F *HOW) (S WHEN) (F *INTERVAL) (S END) (F *SIMPLE-TIME))
((F *INTERVAL))
((F *INTERVAL) (S END))
((F *INTERVAL) (S END) (F *SIMPLE-TIME))
((F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *HOW) (S WHAT) (F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *HOW) (S WHAT) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHAT) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *HOW) (S WHEN) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHEN) (F *SIMPLE-TIME) (S AM-PM) (F PM))
((F *HOW) (S WHEN) (F *INTERVAL) (S INCL-EXCL) (F EXCLUSIVE))
((F *HOW) (S WHEN) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S HOUR) (F 2))
((F *HOW) (S WHEN) (F *INTERVAL) (S END) (F *SIMPLE-TIME) (S AM-PM) (F PM))

Figure 8.17: **Remaining Distinguishing Features**

Once the user has responded with the correct value for the feature, only the alternative interlingua structures that have that value for that feature are kept, and the rest are eliminated. In the case of a wh-question, if the user selects "None of these alternatives are acceptable", all of the alternative hypothesized structures are eliminated. After that, all of the features which no longer partition the search space into equivalence classes are also eliminated. In the example, since the answer to the user's answer to the generated question in Figure 8.15 was Yes, the result is that three of the original five hypotheses are remaining, displayed in Figure 8.16, and the remaining set of features which still partitions the search space can be found in Figure 8.17.

Then the cycle begins again by selecting a feature, generating a question, and so on. If we assume unless otherwise indicated that the correct structure is one of the

alternatives constructed in the Hypothesis Formation Stage, then when we get down to the last structure, we can normally assume it is the right one since it is the only one consistent with everything the user has told us, and we assume what the user told us is consistent with the right structure. However, if there are no remaining distinguishing features and all of the user's answers to questions have been negative, then ROSE cannot return the remaining hypothesized structure with any confidence. In these cases, as well as in the case where all alternative hypotheses are eliminated, it asks the user for a rephrase.

8.2 Qualitative Evaluation

The Interaction Mechanism insures that the user will never have to answer a question who's answer was implied by the answer to a previous query. It insures this by removing all of the structures that are not consistent with the information that the user has confirmed explicitly and then removing the features that no longer partition the search space. This claim can be proven with a proof by contradiction. We know that after the user answers a question, all of the interlingua structures that are not consistent with the user's answer are then eliminated. We also know that all of the questions that do not partion the search space after all of the inconsistent structures have been removed are also eliminated. Now assume that one of the remaining questions has an answer that was implied by the answer to a previous question. If this remaining question was not removed, it must partition the search space since otherwise it would have been removed. This means that among the alternative structures, there are at least two different values for this feature. But if a previous question implied the answer to the question associated with this feature, then there must be only one value that is consistent with what is already known. Since a contradiction arises when the assumption is made that one of the remaining questions has an answer implied by a previous question, it must be true that by following the procedure outlined above we can be certain that all of the questions that are left are ones that we do not yet have an answer for. In this sense, we know it will not ask questions about anything except for what is not known yet.

The other question is whether it will not fail to ask about what it does not know. Since the purpose of asking questions is to distinguish between different alternative interlingua structures, all of the relevant questions involve differences between the alternative structures. Since the set of initial features contain all of the full paths from the top level frames to the terminal fillers, every interlingua structure can be distinguished from every other non-identical alternative structure with some subset of these features. Therefore just this set of most specific features would be enough to generate all of the questions necessary

for getting the needed information. However more general versions of these features group the alternatives into larger groups. These more general features are not strictly speaking necessary, but they make it easier to eliminate a large portion of the search space with a single question. Since the full set of features contains both the most specific features as well as the more general ones, one can conclude that the set of questions is complete. The only question which remains is whether the strategy of eliminating questions as information becomes available, thus eliminating the necessity of asking every question, will in any case eliminate a question which should have been asked. But the only reason why any question would need to be asked is because it would distinguish between at least two non-identical interlingua structures. If a question distinguishes between at least two non-identical interlingua structures, then it would by definition partition the set of alternative interlingua structures. In this case, it would not have been eliminated with the strategy described above since the only circumstance under which a question is eliminated is if it no longer partitions the search space.

8.3 Limitations

The most serious limitation of ROSE's distinguishing feature based interaction is its invalid simplifying assumption that the correct interlingua structure for the sentence is included in the top list of alternative interlingua structures generated by the Combination Mechanism. In the case that the correct structure is not one of the alternatives, then if the user answers all of the possible questions that would distinguish the given alternatives from one another, all of the alternative interlingua structures would be eliminated. But because of the way questions and structures are eliminated during the interaction process, it is not guaranteed to ask the final question which would eliminate the last remaining interlingua structure. And the structure returned would not be an entirely correct representation of the speaker's meaning although it is consistent with the user's answers to the system's questions. In order to compensate for this, when ROSE no longer has any distinguishing features, it attempts to determine whether it has sufficient confidence in the remaining structure to return it. Currently, it makes this determination based on whether any of the questions were answered positively. If none of them were answered positively, it asks the user for a rephrase, assuming the remaining structure is not acceptable either.

A more serious consequence of this invalid simplifying assumption is that the structure returned from the Interaction Mechanism is not guaranteed to be the one that is closest to the ideal interlingua structure. Take as an example a case where there are two alternative structures. One of them is 90% like the ideal interlingua structure and

one is only 10% like the ideal structure. Of course, one would rather have the interaction mechanism select the one that is 90% correct over the one that is 10% correct. But the one that is 10% correct may have something in common with the ideal structure that the one that is 90% does not have in common with it. For example, the 10% correct structure may contain a temporal expression that is missing from the 90% correct structure. If the first question is regarding the temporal expression that is in the 10% correct structure but that is missing in the 90% correct structure, the 90% structure would be thrown away and the 10% correct structure would be returned instead. This situation, though theoretically possible, was never observed to occur in the Interaction evaluation described in this dissertation.

A similar situation is when the user misunderstands the system's question or simply responds incorrectly. This occurs particularly when the system asks if a particular expression represents part of the speaker's meaning. In some cases the best that the system can do at representing the speaker's meaning based on the sentence that was uttered does not sound like perfect English. If it does not sound good to the user, the user may reject it even if it is technically correct. If communication between the system and the user breaks down during the interaction process, and inaccurate information is communicated, the correct or best alternative interlingua structure could be inadvertently thrown out. In order to avoid this situation as much as possible in the Interaction experiment described in this dissertation, users were first trained briefly to know how to answer the questions. First, they were exposed for ten minutes to a set of sample translations generated by the JANUS machine translation system for a different set of sentences than those used in the experiment itself. They were also coached through interactions with the system on this separate set of sentences for ten minutes.

8.4 Interaction in ROSE Compared to IRLH

The role of interaction in ROSE is to clarify the system's confidence about the alternative meaning representation hypotheses constructed during the Hypothesis Formation stage. These hypotheses are ranked according to the fitness values assigned during the genetic search. If the fitness function were perfect, there would be little need for interaction. Since the fitness function can only rely on indirect evidence about the relative goodness of hypotheses, however, there is no guarantee that the hypothesis ranked as best is in fact the best hypothesis. In Chapter 6, for example, we saw an example where the ideal repair hypothesis was ranked by ROSE's Repair Hypothesis Formation stage as third out of five. With two questions, ROSE was able to return the correct result. It is also possible than none of the constructed hypotheses are acceptable. Through interaction with the user, some

or possibly all of the constructed hypotheses may be eliminated. If some of the constructed hypotheses remain after interaction and ROSE has not lost confidence in its set of repair hypotheses, the best ranked one out of the remaining hypotheses is returned. Otherwise, if the maximum number of questions to the user has not been reached, the user is asked to rephrase. Though anything short of confirming every repair action leaves open the possibility of making an incorrect repair, results show that even very limited interaction improves ROSE's ability to increase translation quality.

As mentioned previously, what distinguishes ROSE from the earlier IRLH approach is that ROSE separates Hypothesis Formation and Interaction with the User into two separate stages. In contrast, in IRLH, queries are generated to verify each repair step (Local Repair Hypotheses) rather than first constructing a set of alternative ways of fitting together the whole set of fragments (Global Repair Hypotheses) as in ROSE. No repairs are made without interaction. The IRLH approach searches for the complete meaning representation structure by generating and testing individual local hypothesised repair actions. When these hypotheses are confirmed to be correct through interaction with the user, the repair module then makes the specified repair. Because of the way interaction is interleaved with hypothesis formation, every repair made is guaranteed to be correct with respect to the user's answers to the system's queries. On the down side, however, the number of possible repairs is bounded by the maximum number of questions the system is allowed to ask the user. When hypothesis formation and interaction are separated into two separate stages, the number of possible repairs is no longer bounded by the maximum number of questions, but on the downside, there is no longer any guarantee that every repair will be correct.

In Figure 8.18 we see how the two alternative approaches compare on the same example. In this transcript we see first what the original sentence was and a gloss of the largest set of contiguous chunks spanning the chart inside the parser. Below that we see the interaction between the system and the user, first in ROSE mode with one user and then in IRLH mode with a different user. For this example, ROSE only needed to ask a single question about the one aspect of the alternative hypotheses that differed. IRLH, on the other hand, must test each potential repair, so it generates three questions (the maximum allowed in ROSE's evaluation presented in this dissertation[3]). In the first question, it establishes what the top level frame of the target structure is. In the next question, it determines that the chunk corresponding to the expression "on the twenty-ninth of September" should be

[3]This limitation was placed on the system in order to keep the task from becoming too tedious and time consuming for the users. It was estimated that three questions was approximately the maximum number of questions that users would be willing to answer per sentence. Considering that repair occurs in for about a third of the sentences, and interaction takes place in the majority of cases where repair takes place, in an interactive system, users might not even be willing to answer that many questions.

inserted into the WHEN-O slot. Though the resulting structure is acceptable at this point, IRLH has no way of knowing that it would be acceptable to stop questioning at this point. It continues until it either reaches the maximum number of questions or runs out of potential repairs. In the last question it asks if the chunk corresponding to "September" should be inserted into the WHEN slot. The answer in this case is NO. The resulting structures for both approaches are of comparable quality, however in the ROSE case, only a single question was required where the IRLH approach required three.

Another distinction between ROSE and IRLH is in regard to portions of the sentence for which the parser was not able to construct a partial analysis. In these cases, IRLH is able to fill in the gaps by generating guesses and confirming them through questions to the user. Through experimentation it was determined to be impractical to attempt something similar in ROSE. Without having the opportunity to confirm guesses with the user before incorporating them into hypothesized meaning representation structures, it is likely that incorrect information will find its way into the majority of otherwise good repair hypotheses. Though this would seem to make IRLH preferable to ROSE, in practice IRLH requires too large a number of questions before it yields any advantage with this technique. Such an example can be found in Chapter 4 (section 4.5). If IRLH is limited to a small number of questions, it rarely benefits from this technique.

As discussed in Chapter 4, in the IRLH case, specific strategies were employed to determine which repairs to attempt before others, and so forth. In ROSE, the genetic search guides the repair process in the Hypothesis Formation stage, and questions are selected in the Interaction with the User stage based on their potential for reducing the set of alternative Global Repair hypotheses once they are already constructed. In IRLH, repairs are attempted systematically by first determining what is the top level semantic frame in the target meaning representation structure and then progressively refining the structure by inserting the other fragments into the appropriate slots. By first determining what is the top level semantic frame, the number of possible repairs that need to be considered is drastically cut down since it is possible to then eliminate from consideration any potential repair where the top level chunk is inserted into another chunk. It is only necessary to test hypotheses about other chunks inserted into that top level chunk. This strategy provides a useful starting point for the search. So first determining which of the chunks returned from the parser, if any, contains the top level semantic frame is a logical strategy. However, since in the IRLH approach, repairs are only performed when they are confirmed by the user, this approach is only as good as the user's ability to identify what the central concept of the sentence is. If the user answers incorrectly, the ideal structure can never be constructed, even

with infinitely many questions. Since in ROSE, hypotheses are constructed independently of Interaction with the User, the user needs not think in such linguistic terms. Though the possibility still exists that ineffective interaction between the system and the user during the interaction stage could make the result come out worse than if ROSE simply returned the hypothesis it ranked independently as best, the role of the user becomes one more of confirming acceptable versus unacceptable results rather than guiding the search for good repairs in the first place.

On the one hand, this difference in emphasis in ROSE would seem to give the user less control over the final result through the questioning process. The full set of alternatives under consideration is fixed before interaction, and it is not guaranteed that one of those results is the ideal result, or even an acceptable result. However, interaction in ROSE makes it possible to determine based on a very few questions whether the system has any confidence in the range of global hypotheses which it constructed. If it loses confidence in its hypotheses, it can ask the user for a rephrase. In the IRLH case, it is not clear how the system can determine quickly when it should abandon trying to work with the chunks it has been given and ask for a rephrase instead. Keeping this in mind, a different picture of the comparison between IRLH and ROSE emerges. With the ability to quickly determine whether it is more likely to be fruitful to ask for a rephrase, it is possible for ROSE to effectively give the user *more control* over the final result with *less effort* than with the IRLH approach.

In Figure 8.19[4] we see a case where ROSE rightly loses confidence in the alternative repair hypotheses it was able to construct. It then asks for a rephrase which was able to be correctly interpreted by the system. IRLH, on the other hand, is not able to make any progress on this sentence. It alternatively tests three different chunks to see if they are the top level chunk, and all three are rejected.

[4]In this example, the parser constructs a misleading set of chunks for the sentence. The phrase "going to" parses into the frame `*considering` which then generates as "thinking". This is a good example of the types of confusions that can take place when only partial analyses can be constructed.

ROSE Interaction:

What you said: HOW 'BOUT THE AFTERNOON OF SEPTEMBER OF SEPTEMBER THE TWENTY NINTH
What ROSE understood: HOW ABOUT THE AFTERNOON – SEPTEMBER – THE TWENTY-NINTH OF SEPTEMBER
Interaction:
ROSE: Was something like HOW ABOUT THE AFTERNOON OF SEPTEMBER part of what you meant?
RE3DG: Yes.

ROSE Result: HOW ABOUT THE TWENTY-NINTH OF SEPTEMBER THE AFTERNOON – SEPTEMBER

IRLH Interaction:

What you said: HOW 'BOUT THE AFTERNOON OF SEPTEMBER OF SEPTEMBER THE TWENTY NINTH
What ROSE understood: HOW ABOUT THE AFTERNOON – SEPTEMBER – THE TWENTY-NINTH OF SEPTEMBER
Interaction:
ROSE: Is HOW ABOUT THE AFTERNOON close to the gist of your sentence?
SA1TJ: YES.
ROSE: Was something like ON THE TWENTY-NINTH OF SEPTEMBER HOW ABOUT THE AFTERNOON at least part of what you meant?
SA1TJ: YES.
ROSE: Was something like ON THE TWENTY-NINTH OF SEPTEMBER HOW ABOUT SEPTEMBER THE AFTERNOON at least part of what you meant?
SA1TJ: NO.

IRLH Result: ON THE TWENTY-NINTH OF SEPTEMBER HOW ABOUT THE AFTERNOON

Figure 8.18: **ROSE Compared with IRLH**

ROSE Interaction:

What you said: AND I'M NOT OBVIOUSLY NOT GOING TO MEET WITH YOU ON THE WEEKEND
What ROSE understood: AND I – I WILL SEE YOU ON THE WEEKEND
Interaction:
ROSE: Was something like I WILL SEE YOU ON THE WEEKEND SO THAT I AM THINKING par t of what you meant?
USER: No.
ROSE: Was something like I WILL SEE YOU ON THE WEEKEND BE NOT THINKING part of what you meant?
USER: No.
ROSE: I didn't quite understand your sentence, can you please rephrase more simply?
USER: I WILL NOT SEE YOU ON THE WEEKEND.

ROSE Result: I WILL NOT SEE YOU ON THE WEEKEND.

IRLH Interaction:

What you said: AND I'M NOT OBVIOUSLY NOT GOING TO MEET WITH YOU ON THE WEEKEND
What ROSE understood: AND I – I WILL SEE YOU ON THE WEEKEND
Interaction:
ROSE: Is I WILL SEE YOU ON THE WEEKEND close to the gist of your sentence?
USER: NO.
ROSE: Is AND I close to the gist of your sentence?
USER: NO.
ROSE: Is BE NOT OBVIOUSLY THINKING close to the gist of your sentence?
USER: NO.

Figure 8.19: **Rephrase Example**

Chapter 9

Discourse ROSE: Using Discourse Processing in Repair

Evidence from the study of human language comprehension makes a strong case for the useful role of contextual information in understanding under non-ideal circumstances. In keeping with the analogy of human/computer interaction through a natural language interface as being similar to communication between humans with a small shared language base, we would expect contextual information to be useful for robust natural language processing.

The essential role of contextual information in communication even under ideal circumstances has been well documented (Just and Carpenter, 1987; Clark and Wilkes-Gibbs, 1986; Schiffrin, 1987; Rosé, 1995; Fox, 1987). Just and Carpenter (1987) report that context and world knowledge play such a significant role in understanding that humans consistently distort the meaning of what they read in order to make it consistent with their knowledge of the world and the discourse context. Evidence from the neurolinguistics community (Rosenbeck, LaPointe, and Wertz, 1989) indicates that contextual cues may be particularly helpful to conversational participants with understanding deficits, such as aphaisics. Pierce and Beekman (1985) report that in a study of 20 aphaisic subjects, the subjects who performed lower on comprehension tests involving sentences out of context tended to perform significantly better on tests involving context. On the other hand, the results for the other subjects were not significantly altered. Since humans with understanding deficits find the information provided locally in one sentence to be insufficient for understanding, they need to rely on other sources of information for support. Similarly, if a listener's language competence does not include enough knowledge to fully process a sentence, then the sentence itself does not provide the listener with enough information for understanding. We would expect context to be more important in these cases as well.

Though discourse processing is not essential to the basic ROSE approach, the above cited evidence indicates that discourse information potentially has a lot to offer an

approach to robust interpretation. In this chapter I discuss one successful way discourse information has been used in ROSE. A version of ROSE without discourse processing is compared with a version with discourse processing (Discourse ROSE). In Discourse ROSE, discourse processing provides contextual expectations which guide the Interaction Mechanism as it formulates queries to the user in order to narrow down the set of alternative hypotheses produced in the Hypothesis Formation stage. By computing a structure for the discourse, the discourse processor is able to identify the speech act performed by each sentence as well as augment temporal expressions from context. Based on this information, it computes the constraints on the speaker's schedule expressed by each sentence that must be entered into a calendar structure that keeps track of what each speaker has revealed about his schedule in the course of the negotiation. This task information can then be used for focusing the interaction between system and user on the task level rather than on the literal meaning of the user's utterance. This chapter contains a discussion of the theoretical foundation for the Enthusiast discourse processor (Rosé et al., 1995) used in Discourse ROSE as well as a discussion of how this information is used in formulating queries on the task level. A discussion of some other possible uses for discourse information in robust interpretation is also included.

9.1 Practical Discourse Processing

The Enthusiast discourse processor was developed independently of this dissertation research. Nevertheless, because it provides information to the machine translation system that forms the context for this work, it is useful for the ROSE approach to take advantage of the information that it provides.

The Enthusiast discourse processor is one of a number of systems that have been developed to build a representation of the discourse context either with a plan-based or finite state automaton based discourse processor (Allen and Schubert, 1991; Smith, Hipp, and Biermann, 1995; Lambert, 1993; Reithinger and Maier, 1995; Rosé et al., 1995; Levin et al., 1995). Of these, only the Verbmobil discourse processor (Reithinger and Maier, 1995) and the Enthusiast discourse processor (Rosé et al., 1995) used in Discourse ROSE are designed to be used in a wide coverage, large scale, spontaneous speech system[1]. And of those two, only the Enthusiast discourse processor builds a plan based representation of the whole discourse. The Enthusiast discourse processor was patterned primarily after Lambert's recent work (Lambert, 1993; Lambert and Carberry, 1992) because of its relatively broad coverage in comparison with other computational discourse models and because of the way it

[1]The speech system described in (Smith, Hipp, and Biermann, 1995) operates on a much smaller scale.

represents relationships between sentences, making it possible to recognize actions expressed over multiple sentences. Aspects of Lambert's model that are too knowledge intensive for the kind of coverage necessary in such a large-scale system are left out.

```
((frame *busy)
 (sentence-type *state)
 (a-speech-act (*multiple* *reject *state-constraint))
 (who ((frame *i)))
 (when ((frame *simple-time) (day-of-week tuesday))))
```

Figure 9.1: **Sample Interlingua Structure for "I am busy Tuesday."**

Development of this discourse processor was based on a corpus of 20 spontaneous Spanish scheduling dialogues containing a total of 630 sentences. These dialogues were transcribed and then parsed with the GLR* skipping parser (Lavie, 1995). The resulting interlingua structures (See Figure 9.1 for an example, or Appendix B for a detailed discussion) were then processed by a set of matching rules. These matching rules, similar to those described in (Hinkelman, 1990), assigned a set of possible speech acts based on the interlingua representation returned by the parser. Notice that the list of possible speech acts resulting from the pattern matching process are inserted in the `a-speech-act` slot. It is the structure resulting from this pattern matching process that forms the input to the discourse processor. Goals for the discourse processor include disambiguating speech acts, resolving ellipsis and anaphora, and generating useful predictions for disambiguation and repair.

A strength of this discourse processor is that because it was designed to take a language independent meaning representation (interlingua) as its input, it runs without modification in any language that is parsed into the interlingua representation. It has been tested both in English and in Spanish. Although development and initial testing of the discourse processor was done with Spanish dialogues, the results presented in (Rosé et al., 1995) were obtained by testing spontaneous English dialogues. Results in (Rosé, 1995b) indicate that the discourse processor performs comparably in both languages.

9.2 Overview of The Enthusiast Discourse Processor

S1: 1. When can you meet next week?

S2: 2. Tuesday afternoon looks good.

3. I could do it Wednesday morning too.

S1: 4. Wednesday works.

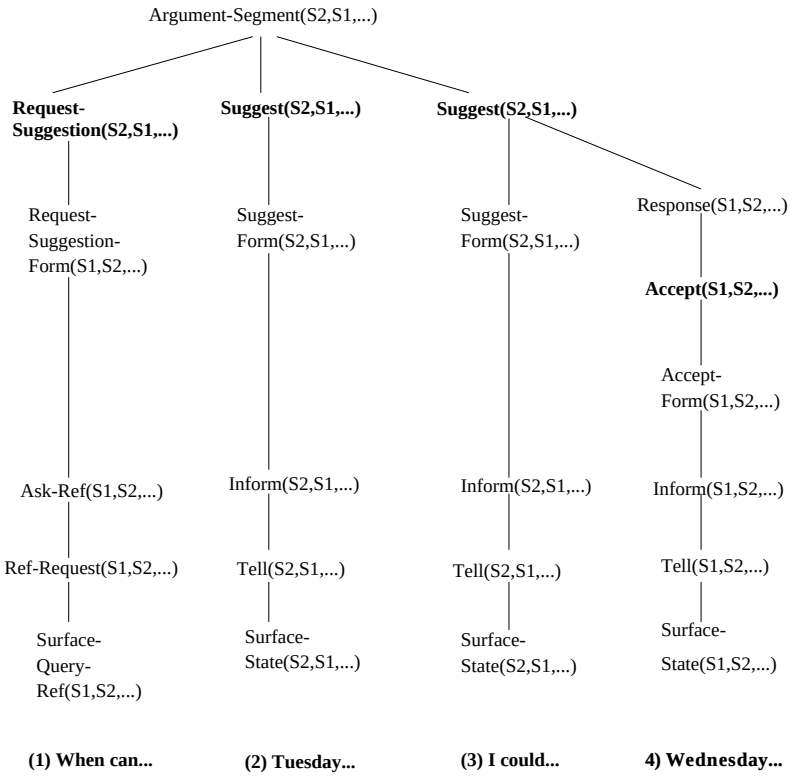

Figure 9.2: **Sample Discourse Structure**

The focus of the development of the Enthusiast discourse processor[2] has been to draw upon techniques developed recently in the computational discourse processing community (Hinkelman, 1990; Lambert, 1993; Lambert and Carberry, 1992), developing a discourse processor flexible enough to cover a wide range of spontaneous dialogues in the scheduling domain. The design and implementation of this discourse processor is based on Lambert's

[2]The original design and implementation of the Enthusiast discourse processor was largely my own work in the context of the Enthusiast system. Since Fall of 94, however, this work has continued as a collaborative effort between Yan Qu and myself, with helpful input from the other members of the Enthusiast team. A special thanks to Yan particularly for her work refining the Temporal Expert program described later in this chapter.

work discussed in (Lambert, 1993; Lambert and Carberry, 1992). The discourse processor has been described in greater depth in (Rosé et al., 1995; Rosé, 1995b). See Figure 9.2 for the representation obtained by this discourse processor for an example dialogue[3]. Notice that there is an inference chain corresponding to each sentence which is built from instantiations of plan operators. These inference chains are built by first selecting a terminal operator corresponding to the mood of the sentence, i.e., declarative, interrogative, imperative. It then instantiates this selected operator with the meaning representation structure constructed from parsing the sentence. From this, the inference chain is built by backwards chaining, attempting to attach the newly constructed inference chain to the plan tree representing the discourse context. This plan-based structure is explained in greater depth below.

Though the Enthusiast discourse processor is built using a slight modification on Lambert's inference algorithm and plan operator formalism, the design behind the plan based structures are necessarily different from hers. Later in this chapter I will describe how the focusing mechanism in the Enthusiast discourse processor is also different from hers. These differences are made necessary by differences in the domain and in the nature of the discourse handled by this system.

Lambert's model was designed to handle information-seeking dialogues in which some planning agent, in an attempt to construct a plan to carry out some domain action, seeks information from an advising agent who is an expert in carrying out the desired plan. Top level discourse nodes in her model are always focussed on obtaining information since the goal of each segment is for the planning agent to learn some bit of information which will allow him/her to instantiate some variable in the domain plan that he/she is building. Negotiation dialogues, such as the one in Figure 9.2, are different in several respects. First of all, both conversational participants are planning agents who will jointly execute the plan they are attempting to build together. Secondly, rather than one participant being an expert, both are experts about their own schedules, but are assumed to not know the other's schedule. So they must each seek information from the other in an attempt to learn enough about each other's schedules that they can find a mutually convenient time to meet. Note that this renders the Master/Slave assumption invalid in this domain. There is no such thing as an executing agent who will always eventually agree with everything an advising agent says. Rather than passively query for information, the transcriptions collected in the scheduling domain indicate that the participants actively make suggestions for meeting times, which the other participant will subsequently either accept or reject. Most commonly

[3]Note that although a complete tripartite structure is computed as in (Lambert, 1993), only the discourse level is displayed here.

Speech Act	**Example**
Opening	Hi, Cindy.
Closing	See you then.
Suggest	Are you free on the morning of the eighth?
Reject	Tuesday I have a class.
Accept	Thursday I'm free the whole day.
State-Constraint	This week looks pretty busy for me.
Confirm-Appointment	So Wednesday at 3:00 then?
Negate	no.
Affirm	yes.
Request-Response	What do you think?
Request-Suggestion	What looks good for you?
Request-Clarification	What did you say about Wednesday?
Request-Confirmation	You said Monday was free?

Figure 9.3: **Speech Acts covered by the system**

the goal of each discourse segment is one of attempting to come to some agreement, and the actions are primarily suggestions, acceptances, and rejections. Though Lambert's model represents how complex discourse actions are built out of individual sentences, and some sentences are assigned speech acts such as Express-Doubt, the primary purpose of her model is not to assign speech acts, and the majority of sentences are not assigned speech acts by her model.

The goal of the Enthusiast discourse processor, on the other hand, is to identify the speech act performed by each sentence and to keep track of the information each speaker has communicated about his/her schedule. Both of these pieces of information result from the execution of the plan recognition algorithm and are then used in Discourse ROSE for

reformulating queries on the task level. There are a total of thirteen possible speech acts identified by the Enthusiast discourse processor. See Figure 9.3 for a complete list.

It is often impossible to tell out of context which speech act might be performed by some utterances, since without the disambiguating context they could perform multiple speech acts. For example, "I'm free Tuesday." could be either a Suggest or an Accept. "Tuesday I have a class." could be a State-Constraint or a Reject. And "So we can meet Tuesday at 5:00." could be a Suggest or a Confirm-Appointment. That is why it is important to construct a discourse model, which makes it possible to make use of contextual information for the purpose of disambiguating this.

Some speech acts have weaker forms associated with them in this model. Weaker and stronger forms very roughly correspond to direct and indirect speech acts. Because every suggestion, rejection, acceptance, or appointment confirmation is also giving information about the schedule of the speaker, State-Constraint is considered to be a weaker form of Suggest, Reject, Accept, and Confirm-Appointment. Also, since every acceptance expressed as "yes" is also an affirmative answer, Affirm is considered to be a weaker form of Accept. Likewise Negate is considered a weaker form of Reject.

When the discourse processor computes a chain of inference for the current input sentence, it attaches it to the current plan tree. Where it attaches determines which speech act is assigned to the input sentence. For example, notice that in Figure 9.2, because sentence 4 attaches as a response, it is assigned a speech act which is a response (i.e., either Accept or Reject). Since sentence 4 chains up to an instantiation of the Response operator from an instantiation of the Accept operator, it is assigned the speech act Accept. After the discourse processor attaches the current sentence to the plan tree thereby selecting the correct speech act in context, it inserts the correct speech act in the `speech-act` slot in the interlingua structure. Some speech acts can be recognized even if they do not attach to the previous plan tree. These are speech acts such as Request-Suggestion or Suggest which are not primarily responses to previous speech acts, though they can be in some cases. These are recognized in cases where the plan inference algorithm chooses not to attach the current inference chain to the previous plan tree.

When the chain of inference for the current sentence is attached to the plan tree, not only is the speech act selected, but the meaning representation for the current sentence is augmented from context. Currently only a limited version of this process is implemented, namely one which augments the time expressions based on previous time expressions. For example, consider the case where Tuesday, April eleventh has been suggested, and then the response only makes reference to Tuesday. When the response is attached to the suggestion,

the rest of the time expression can be filled in. More details about this process are described later in this chapter.

9.3 Focusing in the Enthusiast Discourse Processor

The decision of which chain of inference to select and where to attach the chosen chain, if anywhere, is made by the focusing heuristic. The focusing heuristic in the Enthusiast discourse processor is a version of the one described in (Lambert, 1993). The Enthusiast focusing heuristic has been modified to reflect a new approach to discourse structure. This new approach challenges an assumption underlying the best known theories of discourse structure (Grosz and Sidner, 1986; Scha and Polanyi, 1988; Polanyi, 1988; Mann and Thompson, 1986), namely that discourse has a recursive, tree-like structure. Note that it is primarily because of the fact that dialogues in our corpus do not conform to a strict tree structure that it is necessary to compute the structure of the discourse with the Enthusiast discourse processor in order to compute the information used for reformulating queries to the user on the task level in Discourse ROSE. In this section I describe the multiply threaded structure of some of the negotiations in our corpus and how they are modeled in the Enthusiast discourse processor.

Webber (1991) points out that Attentional State[4] is modeled equivalently as a stack, as in Grosz and Sidner's approach, or by constraining the current discourse segment to attach on the rightmost frontier of the discourse structure, as in Polanyi and Scha's approach. This is because *attaching* a leaf node corresponds to *pushing* a new element on the stack; *adjoining* a node D_i to a node D_j corresponds to *popping* all the stack elements through the one corresponding to D_j and *pushing* D_i on the stack. Grosz and Sider (1986), and more recently Lochbaum (1994), do not formally constrain their intentional structure to a strict tree structure, but they effectively impose this limitation in cases where an anaphoric link must be made between an expression inside the current discourse segment and an entity evoked in a different segment. If the expression can only refer to an entity on the stack, then the discourse segment purpose[5] of the current discourse segment must be attached to the rightmost frontier of the intentional structure. Otherwise the entity to which the expression refers would have already been popped from the stack by the time the reference would need to be resolved.

[4] Attentional State is the representation used for computing which discourse entities are most salient.

[5] A discourse segment purpose denotes the goal that the speaker(s) attempt to accomplish in engaging in the associated segment of talk.

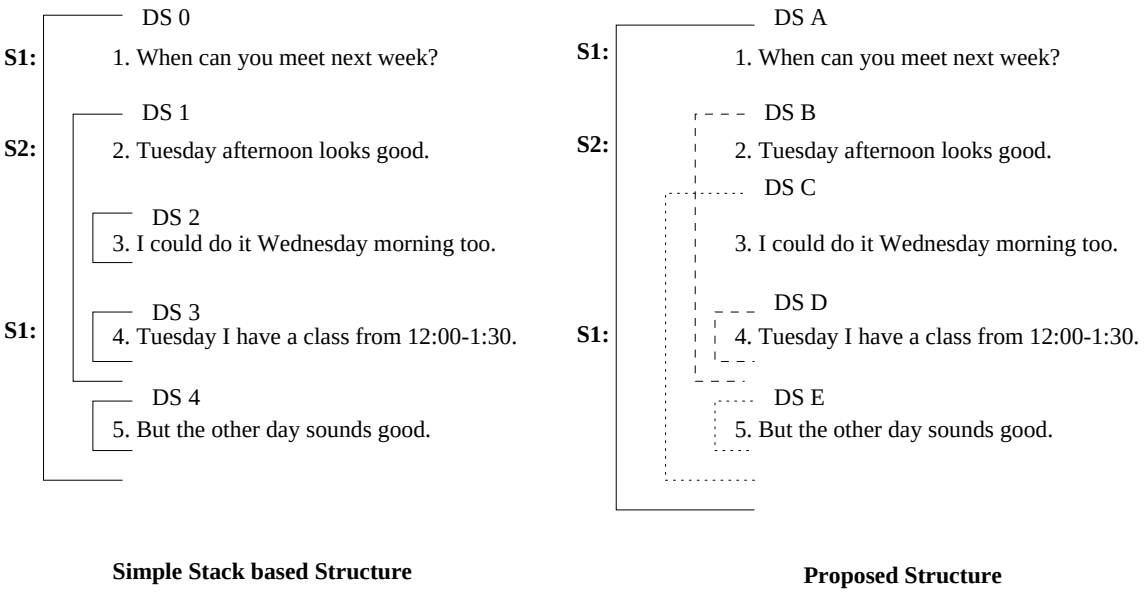

Figure 9.4: **Sample Analysis**

Figure 9.4 contains two alternative analyses for a slightly modified version of the example dialogue from Figure 9.2 assuming dialogues of this nature would be segmented at this level of granularity. Note that researchers such as Grosz and Sidner would most likely choose to include entire interactions like this in a single discourse segment. In our corpus, however, entire dialogues consisting of twenty or more turns are made up of interactions such as this. If they would not be segmented, then it would not be possible to compute any predictions based on global focus for them.

In this case, speaker2 responds to both of speaker1's suggestions in turn. The analysis on the left is the one that would be obtained if Attentional State were modeled as a stack. It has two shortcomings. The first is that the suggestion for meeting on Wednesday in DS 2 is treated like an interruption. Its focus space is pushed onto the stack and then popped off when the focus space for the response to the suggestion for Tuesday in DS 3 is pushed[6]. Clearly, this suggestion is not an interruption however. Furthermore, since the focus space for DS 2 is popped off when the focus space for DS 4 is pushed on, Wednesday is nowhere on the focus stack when "the other day", from sentence 5, must be resolved. The only time expression on the focus stack at that point would be "next week". But clearly this expression refers to Wednesday. So the other problem is that it makes it impossible to resolve anaphoric referring expressions adequately in the case where there are multiple threads, as in the case of parallel suggestions negotiated at once. Because temporal

[6] Alternatively, DS 2 could not be treated like an interruption, in which case DS 1 would be popped before DS 2 would be pushed. The result would be the same. DS 2 would be popped before DS 3 would be pushed.

information augmented from context is one key piece of information used in Discourse ROSE for reformulating queries, it is essential to handle interactions such as this one effectively.

A solution to this problem is modeling Attentional State as a graph structured stack rather than as a simple stack. A graph structured stack is a stack that can have multiple top elements at any point. Because it is possible to maintain more than one top element, it is possible to separate multiple threads in discourse by allowing the stack to branch out, keeping one branch for each thread, with the one most recently referred to more strongly in focus than the others. The analysis on the right hand side of Figure 9.4 shows the two branches in different patterns. In this case, it is possible to resolve the reference for "the other day" since it would still be on the stack when the reference would need to be resolved. A more rigorous argument and additional implications of this model of Attentional State are explored more fully in (Rosé, 1995).

In Lambert's model, the focus stack is represented implicitly in the rightmost frontier of the plan tree, called the active path. In order to have a focus stack which can branch out like a graph structured stack in this framework, Lambert's plan operator formalism is extended to include annotations on the actions in the body of decomposition plan operators[7], which indicate whether the associated action should appear 0 or 1 times, 0 or more times, 1 or more times, or exactly 1 time. When an attachment to the active path is attempted, a regular expression evaluator checks to see that it is acceptable to make that attachment according to the annotations in the plan operator of which this new action would become a child. If an action on the active path is a repeating action, rather than only the rightmost instance being included on the active path, all adjacent instances of this repeating action would be included.

For example, in Figure 9.5, after sentence 3, not only is the second, rightmost suggestion in focus, along with its corresponding inference chain, but both suggestions are in focus, with the rightmost one being slightly more accessible than the previous one. So when the first response is processed, it can attach to the first suggestion. And when the second response is processed, it can be attached to the second suggestion. Both suggestions remain in focus as long as the node that immediately dominates the parallel suggestions is on the rightmost frontier of the plan tree. The revised version of Lambert's focusing heuristic is described in more detail in (Rosé, 1995b).

[7]Decomposition plan operators are ones which specify how to build a composite action out of a sequence of steps.

S1: 1. When can you meet next week?

S2: 2. Tuesday afternoon looks good.

3. I could do it Wednesday morning too.

S1: 4. Tuesday I have a class from 12:00-1:30.

5. But the other day sounds good.

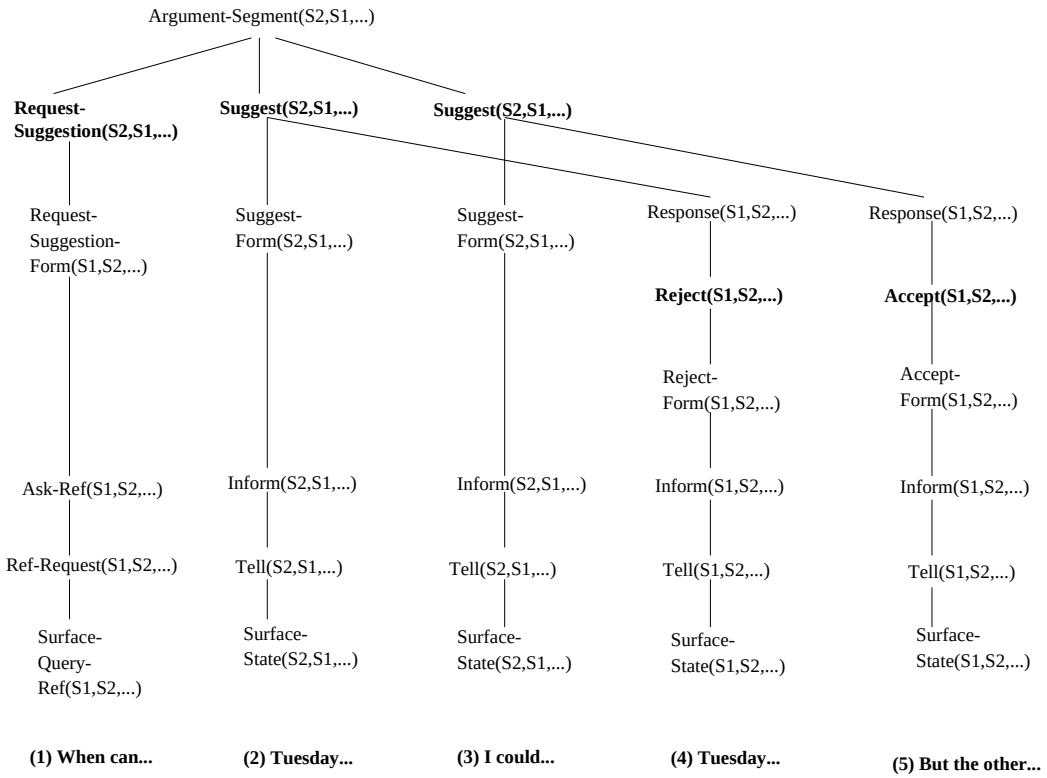

Figure 9.5: **Sample Discourse Structure**

9.4 The Temporal Expert Program

When the inference algorithm tests whether an attachment is possible between the current sentence and a previous sentence represented in the discourse plan tree, it uses the Temporal Expert program to reason about relationships between the time expression in the previous sentence and that in the current sentence. Also, once an attachment is determined to be possible, the Temporal Expert program is used to augment the temporal representation in the current sentence using information stored in the plan tree for the sentence to which it will attach.

The ability of the discourse processor to compute relationships between temporal expressions is particularly important because it provides the primary information used in computing relationships between sentences in the Enthusiast discourse processor. Its ability to augment temporal expressions from context is particularly important to Discourse ROSE since these augmented temporal representations are one key piece of information used in reformulating its queries to the user on the task level.

The Temporal Expert Program handles a large subset of the range of phenomena encountered in the scheduling dialogues. As with other portions of the system, in order to maximize coverage, the principle behind its operation for testing whether a relationship holds between two temporal expressions is to reject cases that can be verified to violate one or more constraints and to accept cases that are either known to be acceptable or have unknown status. In this way, cases that are not covered will not be rejected solely because the system lacks the knowledge necessary for making the decision.

9.4.1 Representation of Temporal Information

The Temporal Expert program accepts queries about temporal expressions stored in interlingua structures, which is how they are represented in the plan tree. The temporal expert currently successfully handles three types of temporal structures: <SIMPLE-TIME>, <TIME-LIST>, and <INTERVAL>. See Figures 9.7, 9.12, and 9.10, for examples.

The Temporal Expert does not evaluate relationships between time expressions in their interlingua form. It first maps this structure onto an internal representation called a *tstructure*. See Figure 9.13. The tstructure was patterned after the interlingua specification for <SIMPLE-TIME> (Figure 9.7) since this is the most basic structure which the other types of temporal structures build on. Because the tstructure is based on the interlingua representation for <SIMPLE-TIME>, interlingua structures matching this type are straightforwardly transformed into the tstructure format. Representations for <TIME-LIST> (see Figure 9.12) can be constructed by creating separate tstructs for each temporal

(<*WHEN> = <SIMPLE-TIME>
<SPECIAL-TIME>
<RELATIVE-TIME>
<INTERVAL>
<EVENT-TIME>
<TIME-LIST>)

Figure 9.6: **Temporal Types in the ILT Specification**

expression in the list and merging them into the fewest number of tstructs possible in order to cover the same range of time. <INTERVAL> (see Figure 9.10) can be constructed by first constructing tstructs for the endpoints and then computing a tstruct that bridges the gap between them. It is the structure that bridges the gap which is returned.

As mentioned in the previous paragraph, a merge algorithm merges lists of tstructs into the smallest number of tstructs that can represent the same period of time. The reasons why it is advantageous to merge tstructs are compactness, uniformity, and correctness. Intuitively, it is more efficient to store information in as few structures as possible. When a time expression requiring multiple tstructures is compared with another, each of those tstructures for the first expression must be compared with each of the tstructures for the second one. A more compact representation means fewer comparisons. It also makes the representation more uniform since "Tuesday, Wednesday, and Thursday" will be represented exactly the same way as "Tuesday through Thursday". It also avoids errors which would be introduced by ignoring a relevant piece of information stored in a different tstructure (such as in the case where the time is stored in a separate interlingua structure from the date). It is also important to specify tstructures as completely as possible since the calendar program, described in the next section, uses them as indices to its calendar data base. The algorithm used to merge tstructs assumes that all of the times in the list are related. So if a list contains a tstruct that specifies a day of the week (like "Tuesday") and another one that specifies a time of day (like between 3 and 4), it assumes the time of day was meant to apply to the day of the week. So it merges them into a single tstruct representing "Tuesday between 3 and 4".

To create a tstruct for an <INTERVAL> structure, the temporal expert first creates tstructs for the end-points, otherwise known as the begin struct and the end struct

Definition:

(<SIMPLE-TIME> = ((**frame *simple-time**)
(**minute** [NUMBER-VALUE])
(**hour** [NUMBER-VALUE])
(**day** [NUMBER-VALUE])
(**month** [NUMBER-VALUE])
(**day-of-week** [DAY-OF-WEEK])
(**time-of-day** [TIME-OF-DAY])
(**am-pm** [AM-PM])
(**modifier** <*MODIFIER>)
(**specifier** [SPECIFIER])))

Example:

"two o+clock on wednesday the fourth"

((**sentence-type *fragment**)
(**when** ((**frame *simple-time**)
(**hour 2**)
(**day-of-week wednesday**)
(**day 4**))))

Figure 9.7: **Simple Time expressions**

respectively. So for the expression "Tuesday through Thursday", the end-points would be "Tuesday" and "Thursday" respectively. It then creates a new tstruct that for each field contains the value in the begin structure, the value in the end structure, and every value in between. So in this case, a tstructure would be built with day field filled in with the values corresponding to "Tuesday, Wednesday, and Thursday". If the time represented in the begin-struct is logically after the time represented in the end-struct, such as "Thursday through Tuesday", the Temporal expert interprets the end-struct in such a way as to make it logically subsequent to the begin-point. So in this case, "Tuesday" is assumed to be in the next week, and a tstruct covering "Thursday, Friday, Saturday, Sunday, Monday, and Tuesday" is returned. If either or both of the start time or end time expressions are lists of tstructs, the above procedure will be performed on every pair of time expressions, one from the start-time and one from the end time. The list of tstructs which result from this process are then merged.

Definition:

(<SPECIAL-TIME> = ((**frame *special-time**)
(**specifier** [SPECIFIER])
(**modifier** <*MODIFIER>)
(**name** [NAME])
(**of** <*WHEN>)))

Example:

"on the weekend"

((**sentence-type *fragment**)
(**when** ((**frame *special-time**)
(**name weekend**)
(**specifier** (***multiple* definite plural**)))))

Figure 9.8: **Special Time expressions**

The same temporal expression may need to be evaluated a number of times in the course of processing a single sentence, and throughout the dialogue. So rather than recompute the tstruct each time, correspondences between temporal interlingua structures and tstructures are stored in an association list which is maintained for the duration of the conversation. Each time the temporal expert is asked to map a temporal interlingua structure onto a tstructure, it first checks its association list to see if it has already made that computation, and only if it hasn't will it do the mapping between the interlingua structure and the tstructure representation.

When specifiers like "next" or "first" are processed, they are not stored explicitly in the tstructure. Instead, they are handled by a mechanism that takes as input a reference time from context, the specifier, and the tstructure for the temporal expression excluding the specifier. By referencing the calendar, described in the following section, it is able to determine that the period of time in the specified relationship to the reference time which matches the tstructure is some specific time, and the details for this time are filled in on the tstruct as if they had been expressed explicitly rather than anaphorically. In other words, "next Tuesday" is represented as "Tuesday, September 6, 1994" if the reference time is "September 1, 1994", and "First Tuesday in September" will be the same with the same reference time.

9.4.2 Computing Relationships Between Temporal Expressions

The temporal expert tests whether two time expressions overlap and whether one is subsumed by the other. The overlap relationship is used in determining if a sentence can be a response to a previous sentence. For example, suppose speaker1 has said, "I can meet on Tuesday or Thursday." If speaker2 says, "I'm busy on Thursday", since "Thursday" overlaps with "Tuesday or Thursday", speaker2's utterance can be taken to reject part of speaker1's suggestion. But if speaker2 says, "Friday is a bad day for me", it could not be a response to speaker1's sentence since "Friday" doesn't overlap with "Tuesday or Thursday". The subsumption relationship is used in determining if a sentence can be a further specification or clarification of a previous sentence. For example, suppose speaker1 says "Tuesday is good for me." If speaker2 responds by saying "How about 3:00?", it can be seen as a subordinate suggestion to speaker1's suggestion. Since "3:00" is part of "Tuesday", the form is subsumed by the latter.

The algorithm that tests for overlap takes as input two lists of tstructs, Struct1 and Struct2. It returns true if any tstruct in Struct1 overlaps with any tstruct in Struct2. This overlap is computed by comparing each field in the two tstructs. Two tstructs overlap if for every field there is some overlap between the values in each or if one or the other has unspecified as its value for that field. Otherwise it returns true. So for example, "Tuesday between 2 and 5" overlaps with "3 in the afternoon", but not with "Tuesday at 1". The algorithm for testing for subsumption is applied to pairs of overlapping tstructs. For each field, if it is the case that the filler from the first tstruct is unspecified or that the filler from the second tstruct is specified in such a way that the filler from the first part does not subsume it, return false otherwise return true. A filler subsumes another one if every value contained in the other filler is contained in it but not vice versa. So, for example, "Tuesday between 2 and 3" is subsumed by "between 1 and 5", and by "Tuesday afternoon", but not by "Tuesday at 2".

When the discourse processor processes a sentence, attaching it to the discourse context, an abbreviated temporal expression in the current sentence can be augmented from the context. So, for example, if speaker1 says, "How about Tuesday" and speaker2 responds with "I'm free between 2 and 5", it can be assumed that speaker2 means "Tuesday between 2 and 5". Since the second expression overlaps with the first expression, the fields specified in the first but not in the second can be copied from the first to the second. Once this is done, the temporal representation inside of the proposition in the plan tree can augmented by generating a new temporal representation from the augmented tstruct and inserting it into the plan tree in place of the original structure.

Definition:

(<RELATIVE-TIME> = ((**frame *relative-time**)
(**origin** <*WHEN>)
(**length** <LENGTH>)
(**direction** [VALUE])))

Examples:

"in two weeks"

((**sentence-type *fragment**)
(**when** ((**frame *relative-time**)
(**direction** +)
(**origin** ((**frame *special-time**)
(**name now**)))
(**length** ((**frame *length**)
(**quantity 2**)
(**unit week**))))))

"a few minutes ago"

((**sentence-type *fragment**)
(**when** ((**frame *relative-time**)
(**direction** -)
(**origin** ((**frame *special-time**)
(**name now**)))
(**length** ((**frame *length**)
(**quantity 2**)
(**unit week**))))))

Figure 9.9: **Relative Time expressions**

Definition:

(<INTERVAL> = ((**frame *interval**)
(**incl-excl** [INCL-EXCL])
(**start** <*WHEN>)
(**end** <*WHEN>)
(**specifier** [SPECIFIER]))))

Examples:

"after eleven o+clock"

((**sentence-type *fragment**)
(**frame *interval**)
(**incl-excl exclusive**)
(**start** ((**frame *simple-time**)
(**hour 11**))))

"from two to four"

((**sentence-type *fragment**)
(**frame *interval**)
(**incl-excl inclusive**)
(**start** ((**frame *simple-time**)
(**hour 2**)))
(**end** ((**frame *simple-time**)
(**hour 4**))))

Figure 9.10: **Interval expressions**

Definition:

(<EVENT-TIME> = ((**frame *event-time**)
(**event** <*PREDICATE>)))

Example:

"when you come back"

((**sentence-type *fragment**)
(**when** ((**frame *event-time**)
(**event** ((**frame *return**)
(**who** ((**frame *you**))))))))

Figure 9.11: **Event Time expressions**

Definition:

(<TIME-LIST> = ((**frame *time-list**)
(**specifier** [SPECIFIER])
(**connective** [CONNECTIVE])
(**items** <*WHEN>)))

Examples:

"monday and tuesday"

((**sentence-type *fragment**)
(**frame *time-list**)
(**connective and**)
(**items** (***multiple*** ((**frame *simple-time**)
(**day-of-week monday**))
((**frame *simple-time**)
(**day-of-week tuesday**)))))

"morning or afternoon"

((**sentence-type *fragment**)
(**frame *time-list**)
(**connective or**)
(**items** (***multiple*** ((**frame *simple-time**)
(**time-of-day morning**))
((**frame *simple-time**)
(**time-of-day afternoon**)))))

"friday that week"

((**sentence-type *fragment**)
(**frame *time-list**)
(**connective -**)
(**items** (***multiple*** ((**frame *simple-time**)
(**day-of-week friday**))
((**frame *special-time**)
(**name week**)
(**specifier that**)))))

Figure 9.12: **Time List expressions**

(**tstruct**	*:minute*	unspecified
	:hour	unspecified
	:day	(1)
	:month	(5)
	:day-of-week	(1)
	:year	(1994))

Figure 9.13: **sample tstructure for Sunday, May 1, 1994**

9.5 The Calendar Program

The calendar program is responsible for keeping a digested version of the set of constraints on the schedules of the speakers that have been expressed in the dialogue. It keeps this digested information in a calendar structure. See Figure 9.14. The speech act and augmented temporal information which are constructed by the inference algorithm using the Temporal Expert program provide the input to this calendar program. The augmented temporal expression determines which time slots in the calendar will be filled in, and the speech act determines what status will fill those time slots.

The calendar structure is set up in such a way that knowing what today is, it is possible to easily compute the referent of *tomorrow*, *last week*, *next month*, *last Tuesday*, etc. For any day, it is easy to get the list of constraints on each speaker's schedule which apply to that day. For example, if speaker1 has said that he is busy every Tuesday, then a constraint will appear on each Tuesday in the calendar under Speaker1's schedule that indicates that he is busy. Each day structure contains a schedule for each speaker individually so that it is possible to represent the case in which one speaker is busy during a time when the other speaker is free. Each day contains a list of pointers to constraint structures. See Figure 9.15 for the representation of a constraint.

To insert a constraint into the calendar, a constraint structure is first constructed from the sentence. The temporal expression from the sentence is first mapped into a tstruct which is then inserted into the constraints times slot. The speech act combined with the top level semantic frame of the meaning representation for the sentence is then mapped onto a status which is inserted into the status slot. The full set of statuses is listed in Figure 9.15. Then the speaker associated with the constraint is inserted into the speaker slot. Once the constraint is built, a pointer to it is inserted into each day structure where it applies. Then for each day where a constraint has been inserted, the constraints for that day are sorted according to specificity with the least specific constraints coming first in the list. One time expression is more specific than another if it is included in the other. Then, for each constraint in order of specificity, for each time on that day covered by the tstruct in the constraint, fill in that time slot for the speaker associated with the constraint with the status associated with the constraint. In this way, a digested version of the set of constraints expressed so far is recorded in the calendar program after each sentence has been processed.

Figure 9.16 contains an example calendar entry. This is what the calendar entry for Friday, September 2 looks like after speaker1 says that though she is busy before 8:15, between 8:15 and 9:00 is good. Speaker2 answers that he's busy between 8:15 and 8:45.

(**day**	*:month*	name of the month
	:day	number day of date
	:day-of-week	name of day of week
	:year	year number
	:sp1.sched	array with slots filled in with speaker1's schedule
	:sp2.sched	array with slots filled in with speaker2's schedule
	:constraints	list of pointers to constraints which apply to this day
)		
(**week**	*:number*	number of days in this week structure (this is necessary since some weeks are spread out over two week structures since they span the boarder between two different months.)
	:days	list of day structures
)		
(**month**	*:month*	name of month
	:weeks	list of week structures
)		
(**year**	*:year*	year number
	:months	list of month structures
)		

Figure 9.14: **The Calendar Structure**

(**constraint**	*:status*	one of the following values:
		{accepted, proposed, good,
		considering, neutral, not-preferred,
		bad, busy, rejected}
	:times	list of tstructs which make it possible
		to compute which time slots this
		constraint applies to
	:speaker	sp1 or sp2 to indicate who's schedule
		this constraint should be applied to
)		

Figure 9.15: **Constraint structure**

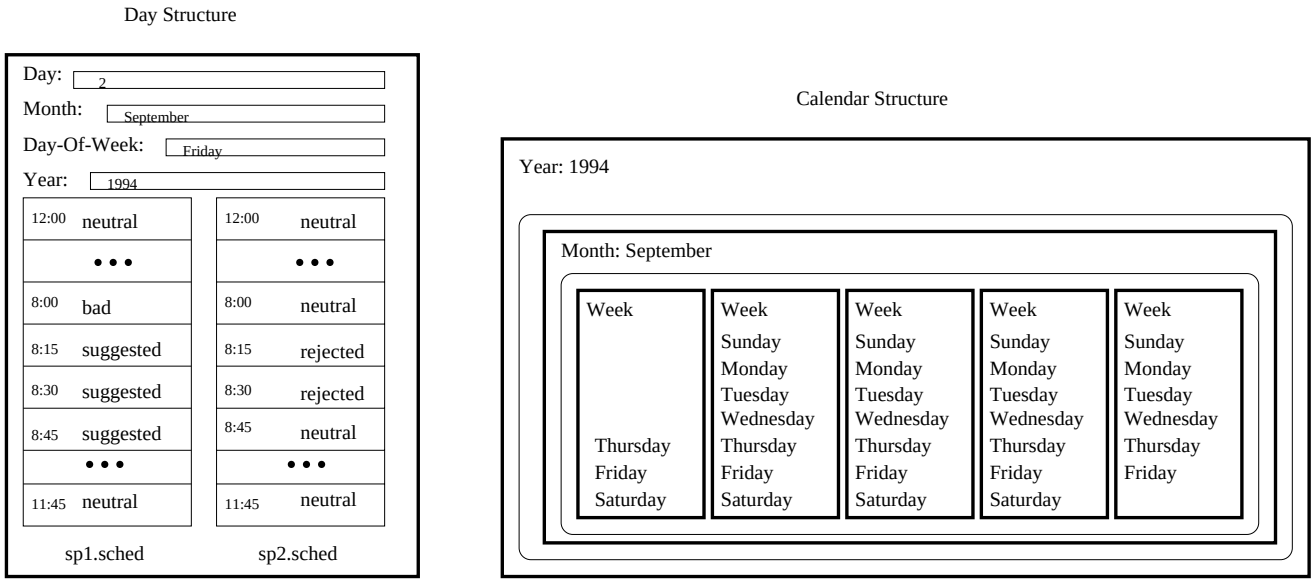

Figure 9.16: **Sample Calendar Entry**

9.6 Discourse Processing in ROSE

In Discourse ROSE, discourse processing provides contextual expectations that guide the Interaction Mechanism as it formulates queries for the user. During the Interaction with the User stage, a set of features are computed that distinguish the alternative meaning representation hypotheses from one another. The discourse processor makes it possible to compute what implications some of these features have for the task. In those cases, the information extracted can be used to refocus the associated question on the task level rather than on the level of the user's literal meaning.

9.6.1 Refocusing Questions on the Task Level

As discussed in this chapter, the discourse processor performs two main functions. First, the discourse processor identifies the speech act performed by each sentence, used to compute status information which will be entered into the on-line calendar. Secondly, it computes augmented temporal information from context which determines which slots in the on-line calendar will be filled in with the status information.

The Interaction Mechanism uses features that distinguish between alternative hypotheses to divide the set of alternatives into classes, where each member of the associated class has the same value for the associated feature. By comparing computed status and augmented temporal information for alternative repair hypotheses within the same class, it is possible to determine what common implications for the task each member or most of the members in the associated class have, and thus what implications for the task are associated with the corresponding value for the feature. By comparing this common information across classes, it is possible to determine whether the feature makes a consistent distinction on the task level. If so, it is possible to take this distinguishing information on the task level and use it for refocusing the associated question on the task level.

In the example in Figure 9.17, the parser is not able to correctly process the "but", causing it to miss the fact that the speaker intended any other time besides ten to twelve rather than particularly ten to twelve. Neither hypothesis correctly represents the meaning of the sentence. In this case, the purpose of the interaction is to indicate to ROSE that neither of the hypotheses are correct and that a rephrase is needed. This will be accomplished when the user answers negatively to the system's query. ROSE selects the feature `((f *how) (s when) (f *interval))` to distinguish the two hypotheses from one another. Its generated query is thus "Was something like 'how about from ten o'clock till twelve o'clock' part of what you meant?". The discourse processor handles these two representations differently. Only the first hypothesis contains enough information for any constraints to be

entered into the calendar structure. It would cause the status of "suggested" to be entered into the calendar for Tuesday the thirtieth from ten o'clock till twelve o'clock. The other hypothesis contains date information but no status information. Based on this difference, discourse ROSE can generate a query asking whether or not this constraint should be entered into the calendar. Its query is "Are you suggesting that Tuesday November the thirtieth from ten a.m. till twelve a.m. is a good time to meet?" The "suggested" status is associated with a template which looks like "Are you suggesting that XXX is a good time to meet?" The XXX is then filled in with the text generated from the temporal expression using the regular JANUS generation grammar.

The discourse processor was only able to provide sufficient information for reformulating 21.6% of the queries in terms of the task. The reason is that the discourse processor only provides information for reformulating questions distinguishing between meaning representations that differ in terms of status and augmented temporal information. In order to make discourse information more widely applicable to reformulating queries, the discourse processor would need to keep track of other sorts of task information, such as specific events causing the status of particular time slots to be bad. In general, whatever task level implications are computed by the discourse processor can be used similarly to reformulate queries distinguishing between meaning representation structures which differ based on the associated types of information.

Sentence: What about any time but the ten to twelve slot on Tuesday the thirtieth?

"How about from ten o'clock till twelve o'clock Tuesday the thirtieth any time"

```
((FRAME *HOW)
 (WHEN (*MULTIPLE*
              ((END ((FRAME *SIMPLE-TIME) (HOUR 12)))
               (START ((FRAME *SIMPLE-TIME) (HOUR 10)))
               (INCL-EXCL INCLUSIVE)
               (FRAME *INTERVAL))
              ((FRAME *SIMPLE-TIME) (DAY 30) (DAY-OF-WEEK TUESDAY))
              ((SPECIFIER ANY) (NAME TIME) (FRAME *SPECIAL-TIME)))))
```

"From ten o'clock till Tuesday the thirtieth twelve o'clock"

```
((FRAME *INTERVAL)
 (INCL-EXCL INCLUSIVE)
 (START ((FRAME *SIMPLE-TIME) (HOUR 10)))
 (END (*MULTIPLE*
              ((FRAME *SIMPLE-TIME) (DAY 30) (DAY-OF-WEEK TUESDAY))
              ((FRAME *SIMPLE-TIME) (HOUR 12)))))
```

Feature: ((f *how) (s when) (f *interval))

ROSE question: Was something like "how about from ten o'clock till twelve 'clock" part of what you meant?

Discourse ROSE Question: Are you suggesting that Tuesday November the thirtieth from ten a.m. till twelve a.m. is a good time to meet?

Figure 9.17: **Discourse ROSE Example**

9.6.2 Evaluating the Usefulness of Questions

It was hoped that the information provided by the Enthusiast discourse processor would also prove useful for distinguishing between useful and superfluous questions, in order to cut down on the number of questions ROSE requires in order to narrow down on the best alternative repair hypothesis. In hindsight it was determined that eliminating such questions also has disadvantages for the Interaction Mechanism. However, an analysis of how this task might have been accomplished and why it turned out not to be practical with the Enthusiast discourse processor gives insights into the challenges of using discourse information in repair. Thus, in this section, I describe the attempt to use discourse information for eliminating superfluous questions as well as present a discussion of why it was difficult.

Because ROSE is entirely domain independent, it has no way of evaluating which differences between alternative hypotheses are significant for the task that the two speakers are collaborating on. For example, several frames contain both a `WHEN` and a `WHEN-0` slot. Which slot contains a time expression structure affects how the generated text is phrased but does not affect the meaning of the sentence. The only difference between them is one of sentence level focus. `WHEN-0` is normally a slot where topicalized temporal information is stored whereas `WHEN` is used for temporal information coming after the verb in the sentence. Because this distinction does not affect the meaning of the sentence adversely, it is not important to ask the user questions which distinguish between alternatives which differ only because of this. However, since ROSE does not know what any of the slots or frames mean, it has no way of determining that this and other similar distinctions are not important to burden the user with. It was hoped that because important questions intuitively have some impact on the task that the discourse processor would provide useful information for distinguishing between important questions versus superfluous questions.

For example, since the difference between `when-0` and `when` neither affects the speech act assignment nor the calendar update, the discourse processor would indicate that this is not an important difference. However, features distinguishing between alternative versions of the temporal expression itself, for example, would make a difference since they would have different implications for which entries in the discourse processor's calendar would be updated. Therefore, the discourse processor would rightly assign this question to the status of important.

9.6.2.1 Performance of Discourse Processor at Eliminating Superfluous Questions

An evaluation demonstrated that the discourse processor made incorrect predictions about the useful/superfluous distinction almost as often as it made correct ones. During the evaluation of the Interaction with the User stage, for each question generated by ROSE, the associated feature was recorded along with its set of alternative hypotheses and the source sentence[8]. Each feature was marked by hand as making a superfluous distinction or not, where the operational definition of useful was that the associated feature divided the set of alternative hypotheses in question into classes such that the hypotheses in one or more classes generated better translations than the hypotheses in competing classes. The highest ranking meaning representation hypotheses from each previous sentences was used to construct the discourse state. Then the discourse processor was used to determine for each hypothesis which speech-act would have been assigned and what information would be entered into the calendar, including date information and status information. For each subset, Discourse ROSE examined the speech act, status, and date information for each alternative hypotheses to determine what information the whole set of alternatives had in common. Then the common information for each subset was compared to the common information for every other subset. If it was the case that for every subset there was some piece of information distinguishing it from each other subset, the question was marked as important, otherwise it was marked as superfluous.

9.6.2.2 Analysis of Discourse Processor's Limitations

One limitation of using discourse information for distinguishing between important and non-important questions is the same as the discourse processor's primary strength. What makes the Enthusiast discourse processor practical in a realistic sized system is that it makes its computations based on very simple and widely applicable information, mainly the relationship between different temporal expressions. This limits its ability to distinguish between certain types of structures, such as between rejections including a justification from rejections not including a justification, for example. Since the justification is not recorded in the discourse representation, the discourse processor will consider all questions regarding justifications of rejections to be unimportant. Objectively, however, these justifications can have an important impact on the quality of the translation. So while Discourse ROSE can be effective for eliminating some types of superfluous questions, it also eliminates some

[8]Only questions from 7 users were included in this evaluation because questions from 2 users were misplaced prior to the evaluation of Discourse ROSE.

important questions. Its ability to eliminate superfluous questions while not eliminating important questions depends upon the granularity in distinctions which the discourse processor is capable of making. It can only recognize a distinction between structures as significant if that difference results in some difference in how the discourse processor will handle those respective structures.

What makes it nearly impossible for the discourse processor to distinguish between useful and superfluous questions in general is that alternative hypotheses both generating very wrong translations may also have different implications for the discourse context. For example, in one case the input sentence was, "I can meet you from eighth to ten.". What the speaker most likely meant was "I can meet you from eight to ten." However, since "the eight" looks like a date rather than a time to the parser, none of the alternative hypotheses correctly represented this meaning. Since all of the translations were bad, no distinguishing feature could be useful by the operational definition. However, though all of the alternative hypotheses were similar in that their top level semantic frame was `*meet`, one of the hypotheses generated was such that the temporal expression was inserted into the `purpose` slot rather than the `when` slot. Since the discourse processor does not know to look in the `purpose` slot for information about which times slots in the calendar to update, this hypothesis appears to be significantly different from the other hypotheses. So a feature which distinguishes that structure from the others would be marked as useful although it does not in fact make a useful distinction by the operational definition. It is impossible for the discourse processor to distinguish between this case and the case where the important distinction is between hypotheses of varying translation quality. The reason is that it is possible for a hypothesis to appear to represent something reasonable to say in the context without that hypothesis corresponding to what the speaker meant. Many of the hypotheses in the set for this example generated sentences such as "I can see you from the eighth on" or "I can see you ten o'clock on the eighth" or "I can see you on the eighth", all of which look like reasonable things for a person to say about their schedule.

With additional information, a discourse processor might have been expected to notice that something was wrong in the previously mentioned example. In the previous sentence, the other speaker had suggested that the twenty third was a good date to meet if the other speaker was available in the morning. Since it is common for speakers in this domain to suggest an alternative rather than explicitly rejecting every undesirable suggestion, the Enthusiast discourse processor assumed from the set of hypotheses that the current sentence was such an example. In order for a discourse processor to be able instead to correctly identify the consistent translation problem, it would have needed both

to be able to evaluate relative goodness of conversational strategies and also be able to map "eighth" onto "eight" in order to realize that it would be possible to reconstruct a more logical conversational move from the speaker's utterance. It might have been able to do this by generating a representation for alternative direct responses to the previous sentence. It might then do a fuzzy match between that set of alternative representations and the current set of hypotheses, noticing a striking resemblance. From that, it might postulate that there was a translation problem, and attempt to build something fitting the direct response set from a hypothesis in the original set using the least number of edit operations. Though this ability could in certain cases make a useful contribution to the repair process, it would likely be very computationally expensive.

In other cases, none of the alternative hypotheses may generate natural sounding text. If it were possible for the discourse processor to evaluate whether a sentence made sense in context, it might be able to identify these cases. However, the amount of knowledge necessary to make this sort of determination is immense. The discourse processor would need to represent something approximating the human interlocutors's reasoning about the conversation. Nothing even close to this is represented in the Enthusiast discourse processor. It is exactly the fact that it avoids such detailed inferences that makes it possible for it to achieve the high level of generality that it has been demonstrated to achieve. In some of these cases, however, the discourse processor would treat the alternatives the same way, and thus be able to correctly avoid asking questions to distinguish between them.

In other cases, the alternatives would appear different to the discourse processor. In one such case, the speaker had said, "We're always running out of time." The two alternative hypotheses generated "Time" and "We always are continuing time." The first one is treated as any fragment, attaching to the discourse context as a clarification, not affecting the discourse state adversely. Fragments like "Time" are universally treated this way by the Enthusiast discourse processor since they do not contain enough information to reliably assign any other discourse function, and attaching low is the least likely to cause problems for subsequent attachments. The other hypothesis appears to the discourse processor as any regular statement. Since the discourse processor's matching rules overgenerate, they indicate that this sentence could be either an acceptance, a rejection, a suggestion, or only a statement of constraints on the speaker's schedule. In reality, this sentence is none of these. But the discourse processor's lenient plan inferencing causes it to come to the conclusion that this sentence attaches best as a suggestion. Any selection from the set of alternatives provided by the matching rules would distinguish this hypothesis from the previous one. The difference in speech acts assigned should indicate that a feature distinguishing between

these two hypotheses corresponds to an important question to ask although neither of these hypotheses even make any sort of sense. If the discourse processor had some sort of knowledge indicating what it means to "make sense", it might have been able to conclude that neither of these translations really says anything. It's not clear, however, how one would make such a determination since both hypotheses are legal structures according to the meaning representation specification. A more detailed or at least more accurate meaning representation specification might be expected to remedy at least part of this problem.

9.6.2.3 Reflections on Eliminating Superfluous Questions

In hindsight, it might not be desirable to eliminate superfluous questions which are considered so simply because all of the alternative hypotheses are bad. In these cases, though the answers to the questions do not assist the system in narrowing down to an appropriate hypothesis, they do assist the system in determining that it does not have confidence in any of its hypotheses, which is what is desirable in these circumstances. If only questions which distinguish between hypotheses which are all good are counted as superfluous, then only 4% of the questions asked overall are superfluous. Under this definition, if questions are eliminated only in the case where exactly the same information is entered in the calendar for each alternative hypothesis, and no alternative is such that information is not entered into the calendar, then 30.0% of the superfluous questions can be eliminated. However, this is a very small difference in the interaction overall, and along with the reduction in superfluous questions, 2.5% of the useful questions are also eliminated. Of these 2.5%, two thirds were such that what distinguished the good hypotheses from the bad ones was not something that the discourse processor pays attention to, such as the specific event being scheduled. In one third of the cases, the hypotheses appeared to be equivalent because what was missing in the bad hypotheses was filled in from context, making all of the hypotheses appear equivalent.

9.6.3 Other Uses for Discourse Information in Repair

One can easily imagine other roles discourse information could play in repair. Other researchers have used discourse information both for augmenting and ranking repair hypotheses (Smith, 1992) and for evaluating when interaction is necessary (Smith, 1997). All three of these would be interesting extensions to Discourse ROSE. With the information the discourse processor currently provides, it would be easy to augment temporal information in repair hypotheses based on contextual information. However, since missing information in the repair hypothesis could make it possible to incorrectly augment temporal

information from context, interaction should be used to verify the accuracy of such augmentations. Using discourse information for ranking hypotheses is not straightforward with the Enthusiast discourse processor. The reason is that the Enthusiast discourse processor is primarily a bottom-up discourse processor. It doesn't provide strong expectations for likely continuations of the discourse, which would be useful for evaluating how reasonable each hypothesis is in relation to the other alternatives. For the same reason, it would be difficult to use the Enthusiast discourse processor for determining when interaction is necessary. Since it is not straightforward to determine how reasonable the best ranked hypothesis seems in context, it is not straightforward to use discourse information for assessing the systems confidence about its hypotheses. Since the discourse model used in (Smith, 1992) and (Smith, 1997) makes strong top-down predictions, it is possible for Smith's discourse processor to perform each of these tasks effectively.

Part V

Evaluation

Chapter 10

Evaluation

There are many different approaches to handling the problem of extragrammaticality, but which way is most appropriate? In this chapter I present an evaluation of the results obtained with the ROSE approach in the context of the large-scale JANUS speech-to-speech translation system. These results are compared here with those of the Minimum Distance Parsing approach (MDP) and the Incremental Repair with Local Hypotheses approach (IRLH). Based on this comparison, I argue that ROSE is more appropriate than either MDP or IRLH for a system as large and complex as JANUS.

10.1 Review of Claims Made in this Dissertation

The ROSE approach is composed of two stages: Hypothesis Formation and Interaction with the User. The Hypothesis formation stage is itself divided into two steps: Partial Parsing and Combination. Lavie's GLR* parser (Lavie, 1995) is used for partial parsing. The parse for the largest segment plus analyses for the skipped portions together form the set of chunks which are input to the Combination step. In the Combination step, the chunks produced during partial parsing are assembled into a set of meaning representation hypotheses. During the Interaction with the User Stage, the user is queried, and the user's responses are used to narrow down the set of hypotheses to the single best hypothesis, which is then returned. In other words, a set of alternative ways of fitting the whole set of fragments from the partial parse (Global Repair Hypotheses) is constructed before any queries are generated rather than generating a query to verify each repair step (Local Repair Hypotheses) before moving on to the next repair step.

I argue that the ROSE approach of separating the Partial Parsing and Combination steps is more efficient than placing the full burden of robustness on a single parsing algorithm. Another goal of this work is to demonstrate that it is more efficient to separate Repair Hypothesis Formation from User Interaction rather than interleaving them.

These claims about ROSE are demonstrated in a rigorous evaluation.

- ROSE's "casting and combining" approach is demonstrated to be significantly faster than the MDP approach by comparing run times of the two alternative approaches over a large corpus of sentences holding all other factors[1] constant. ROSE is demonstrated to achieve a higher level of performance per associated unit of cost in terms of time.

- ROSE's two stage approach to interaction is demonstrated to be more efficient, in terms of improvement per number of questions, than the interleaved IRLH approach. This will be demonstrated by comparing level of translation quality achieved per maximum number of questions between ROSE and IRLH.

- Additionally, an evaluation of Discourse ROSE, a version of ROSE using a discourse processor to transform meaning level queries to task level queries, will demonstrate that discourse information can be used to improve the quality of the interaction.

10.2 Methodology

As discussed previously in Chapter 1, the most straightforward way to evaluate different approaches to handling extragrammaticality would seem to be by comparing them based on improvement in terms of percentage of sentences handled correctly or improvement of overall accuracy on a particular corpus. It is misleading to compare instantiations of different approaches this way, however, since in theory many of these approaches have the potential for yielding the same amount of improvement given sufficient resources in terms of space (both static and dynamic), time (both development time and run time), and interactional effort. The real question is which approach can use these resources more economically. But in practice it is hard to answer this question in all cases since some approaches may require less time and space but more interactional effort, where another may require little interactional effort but more time and space. Or perhaps two approaches will require the same amount of interactional effort, but one will require more time while another will require more space. In these cases it is not straightforward to determine absolutely which approach is more efficient.

One obvious solution would be to compare approaches holding all of these factors constant. For many pairs of approaches, however, it does not make sense to compare this way. For example, say approach one handles extragrammaticality by including buggy rules

[1] such as parsing grammar complexity and vocabulary size

in the parsing grammar using a standard parsing algorithm while approach two uses a more flexible parsing algorithm, eliminating the need for buggy rules. The only way to keep the static space requirements the same would be to use the same grammar for both. In that case, however, in order to keep the parse time the same, no flexibility would be allowed on the part of the second approach. In this case, the only way to hold these two factors constant would be to make these two approaches equivalent. Since what makes these approaches different is a different trade-off in distribution of resources, it doesn't make sense to compare them keeping the distribution of resources the same.

Another solution might be to hold the performance constant, and then compare the space, time, interactional effort trade off, discussing which trade off is more practical. But this is difficult to do precisely in practice. It might also be misleading since the exact trade-off will depend upon the level of performance selected to hold constant. For example, approach one above will improve as the grammar size increases while approach two will improve as the amount of flexibility in the algorithm is increased. Approach one will always take more static space than approach two since it needs a larger parsing grammar. As performance improves, both approaches will take more run-time and require more dynamic space. One interesting question is how much of an increase in run-time and dynamic space are associated with each approach as performance improves. Another is how does this increase trade off with the widening difference in development time and static space caused by the fact that only approach one requires a larger grammar development effort in order for its performance to improve. The verdict of which approach is more efficient, however, keeping all of these things in mind, may still be a judgement call.

In my evaluation, I consider not only the difference in performance between ROSE, MDP, and IRLH, but difference in performance per associated unit of cost. In an interactive system, it is important not only to achieve a high level of quality, but to do it quickly and without burdening the user with too many questions. The two relevant units of cost in this evaluation, then, are seconds of parse time and number of questions. Quality is measured in terms of grades assigned with respect to translation quality by an independent human judge.

10.3 Evaluating the Repair Hypotheses Construction Stage

The Hypothesis Construction Stage of ROSE is evaluated in comparison with the MDP approach. As described previously in Chapter 5, section 6, I make use of a version of Lavie's GLR* parser (Lavie, 1995) extended to be able to perform both skipping and inserting which I refer to as LR MDP. This makes it possible to compare the two stage

ROSE approach to MDP keeping all other factors, such as parsing grammar and lexicon size, constant.

10.3.1 Experimental Design

The GLR* parser uses a semantic grammar with approximately 1000 rules which maps the input sentence onto an interlingua representation (ILT) which represents the meaning of the sentence in a language-independent manner. This ILT is then passed to a generation component which generates a sentence in the target language which is then graded by a human judge as Bad, Partial, Okay, or Perfect in terms of translation quality. Partial indicates that the result communicated part of the content of the original sentence while not containing any incorrect information. Okay indicates that the generated sentence communicated all of the relevant information in the original sentence but not in the ideal way. Perfect indicates both that the result communicated the relevant information and that it did so in a smooth, high quality manner. The corpus used in this evaluation contains 500 sentences from a corpus of spontaneous scheduling dialogues collected in English.

In a previous experiment described in Chapter 5, it was demonstrated that the two stage approach performs about two orders of magnitude faster than LR MDP. For the purpose of the evaluation presented here I tested the effect of imposing a maximum deviation penalty on LR MDP in order to determine how much flexibility could be allowed before the computational cost would become unreasonable.

A full, unconstrained implementation of MDP can find an analysis for any sentence using a combination of insertions, deletions, and transpositions. However, in order to make it viable to test the MDP approach in a system as large as the JANUS system, I make use of a more restricted version of MDP. While the full MDP algorithm allows insertions, deletions, and transpositions, this more constrained version of MDP allows only insertions and deletions. Although this still allows the MDP parser to form some an analysis for any sentence, in some cases the result is not as complete as it would have been with the unconstrained version of MDP or with the two stage repair process. Additionally, with a lexicon on the order of 3000 lexical items, it is not practical to do insertions on the level of the lexical items themselves. Instead, only non-terminals are allowed to be inserted. An insertion penalty equivalent to the minimum number of words it would take to generate a given non-terminal is assigned to a parse for each inserted non-terminal.

In order to test the effect of imposing a maximum deviation penalty, I used a parameterized version of LR MDP, where the deviation penalty of a parse is the total number of words skipped plus the parse's associated insertion penalty as described above.

The avenues of exploration made available here are far from exhaustive. Substitutions and transpositions are not allowed in this version of the parser, nor is it possible to set a separate maximum penalty for skipping and for inserting. Additionally, insertions and deletions are weighted equally, where some researchers have weighted them differently (Hipp, 1992). These and other possibilities are left for future inquiry.

The LR MDP parser was run over the corpus at three different flexibility settings. The first setting, **MDP 1**, is Minimum Distance Parsing with maximum deviation penalty of 1. Similarly, **MDP 3** and **MDP 5** are MDP with maximum deviation penalty of 3 and 5 respectively. **MDP 5** was the most flexible version of MDP that was found to be practical to run with a grammar and lexicon as large as those used in this evaluation. I initially ran a very limited version of GLR* where only initial segments can be skipped which I refer to as **GLR with Restarts**[2]. Thus, while the parser can restart from each word in the sentence, analyses produced are always for contiguous segments of the sentence. My reason for initially running the ROSE evaluation with such a limited flexibility setting on the parser was to test what level of performance can be achieved in the fastest parsing setting. I ran **GLR with Restarts** both with and without repair. Timings for all five of these iterations over the corpus are displayed in Figure 10.1. Notice that **GLR with Restarts** is significantly faster than even **MDP 1**. And since the repair stage is run only for sentences which the repair module determines need repair, and since the repair process takes only seconds on average to run, only a small difference in time can be seen in this graph between the case with repair and the case without repair. In these experiments, ROSE was run using the genetic programming trained fitness function discussed in Chapter 7.

10.3.2 Results

The translation quality ratings for the five different iterations over the corpus are found in Figure 10.3. The timing requirements for the alternative approaches as it varies for sentences of different lengths can be found in Figure 10.1. To make this more meaningful,

[2]The original GLR* parser was meant to return the best scoring analysis of the entire input string minus the portions which were skipped. With this limited flexibility version of GLR*, the determination of what should be considered the best scoring parse becomes fuzzy. Technically, the best scoring parse in a parser which can skip initial segments should be the best scoring final segment. The results presented in this evaluation, however, are for the best scoring contiguous portion of the sentence overall. Therefore, the results presented here for **GLR w/restarts + repair** are higher than they would be if only the result for the best scoring final segment were returned. However, we find this to be a more fair estimate of the performance of a parser which can parse contiguous portions of the input. Nevertheless, in cases where there was no final segment of the sentence which parsed, the result returned was nil even if an acceptable result might have been found somewhere in the parser's chart for previous parsing stages. This occurred in 1% of the cases.

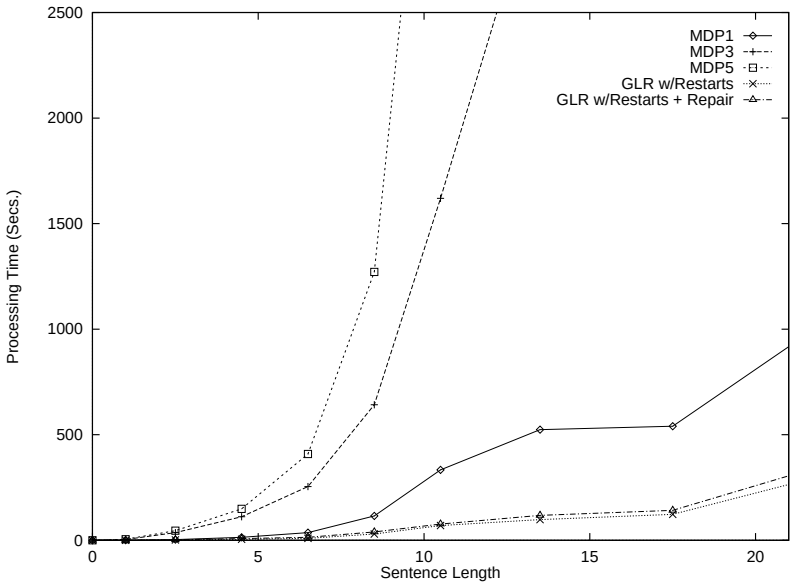

Figure 10.1: **Parse Times for Alternative Strategies**

the distribution of sentence lengths can be found in Figure 10.2. Predictably, **MDP 5** is an improvement over **MDP 1**, with an associated significant cost in run time. Also, not surprisingly, the very restricted **GLR with Restarts**, while it is faster than either of the other two, has a correspondingly lower associated translation quality. However, **GLR with Restarts + Repair** outperforms the others in terms of total number of acceptable translations while not being significantly slower than **GLR with Restarts** and no repair.

Though these results make apparent certain trends in the performance of these alternative approaches, the differences in translation quality overall are very small. For example, the difference in number of acceptable translations between **MDP 5** and **GLR with restarts + repair** is only about 1%. The largest difference between the two is that **GLR with restarts + repair** has about 7% more sentences with translation quality of Partial or better, indicating that **GLR with restarts + repair** produces analyses which are useful for furthering the conversation between the two speakers using the system 7% more often than **MDP 5**. Nevertheless, the very significant difference in efficiency between **MDP 5** and **GLR with restarts + repair** make the ROSE approach a clear winner.

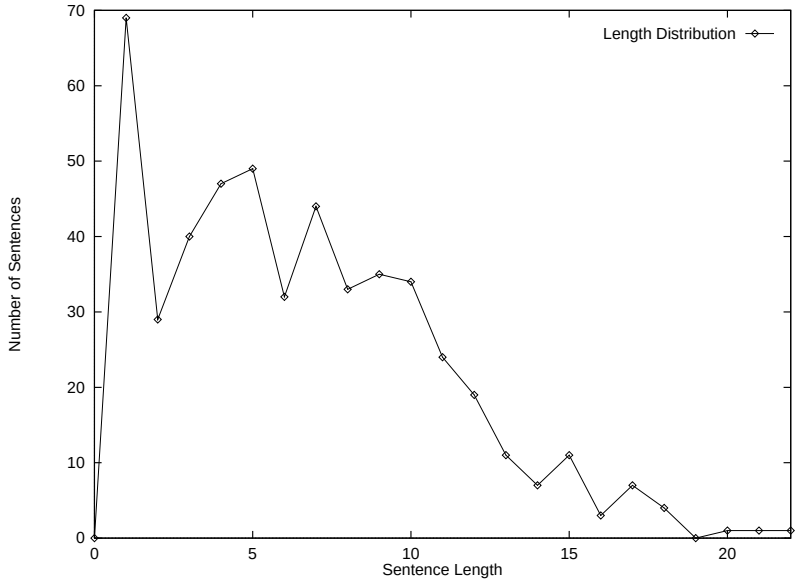

Figure 10.2: **Distribution of Sentence Lengths**

	NIL	Bad	Partial	Okay	Perfect	Total Acceptable
MDP 1	21.4%	3.4%	3.4%	18.4%	53.4%	71.8%
MDP 3	16.2%	4.2%	5.0%	19.6%	55.0%	74.6%
MDP 5	8.4%	8.2%	6.0%	21.0%	56.4%	77.4%
GLR with Restarts	9.2%	6.4%	12.8%	19.4%	52.2%	71.6%
GLR with Restarts + Repair	0.4%	9.6%	11.8%	23.4%	54.8%	78.2%

Figure 10.3: Translation Quality of Alternative Strategies

	NIL	Bad	Partial	Okay	Perfect	Total Acceptable
Linear Combination Fitness Function	0.4%	8.6%	12.4%	22.4%	56.2%	78.6%
Genetic Programming Fitness Function	0.4%	9.6%	11.8%	23.4%	54.8%	78.2%

Figure 10.4: Translation Quality of GLR with Restarts + Repair with Two Alternative Fitness Functions

The contribution made by ROSE's repair stage (the Combination Mechanism) is made more clear by considering the potential improvement made possible by the chunks produced by the parser. The percentage of the corpus where repair was necessary and the parser produced sufficient chunks for constructing an acceptable hypothesis was only 8.6%. Therefore, ROSE's improvement of 6.6% constitutes 76.7% of the potential improvement. Though this still leaves room for further development, it is a significant percentage of the total possible improvement.

As mentioned in Chapter 7 (Section 7.3.3.2), the linear combination trained fitness function performed slightly better than the genetic programming trained fitness function. A comparison of the results obtained under the experimental conditions described above can be found in Figure 10.4. Notice that the total percentage of acceptable translations for the linear combination fitness function is slightly higher than that with the genetic programming trained fitness function. While the genetic programming trained fitness function achieved 76.7% of its potential improvement, the linear combination trained fitness function achieved 81.4% of its potential.

After these results were established for ROSE running with GLR with restarts, a second evaluation was run over the same corpus making use of GLR*'s full power constrained only by a beam limit of 30 on the frontier of the parser's Graph Structured Stack. The purpose of this second evaluation was to test the limits of robustness that are achievable with skipping alone, and to look at the difference in time requirements between the full GLR* approach and the more restrictive GLR with restarts. In this evaluation, full GLR* was run both with and without repair.

In Figure 10.5, it is clear that even **GLR*** is faster than **MDP 1**. Figure 10.6 makes clearer the differences in time between **GLR***, **GLR* + repair**, **GLR with restarts** and **GLR with restarts + repair**. The results for these four alternatives along with **MDP 5** are found in Figure 10.7. Though the time required by **GLR*** is more than **GLR with restarts + repair**, since the parse quality is higher for **GLR*** than **GLR with restarts**, repair is used less often, and thus there is a smaller difference in time between the case with repair and the case without repair. Interestingly, although **GLR***

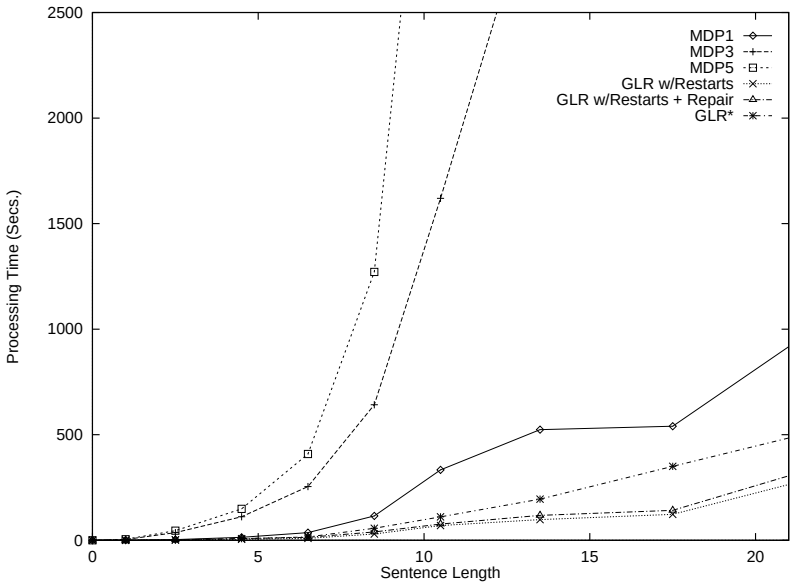

Figure 10.5: **Parse Times for Alternative Strategies**

is slower than **GLR with restarts + repair**, it has fewer overall acceptable translations. Therefore, although it is possible to get more perfect translations with more skipping, **GLR with restarts + repair** gets more acceptable translations faster. By adding repair on top of **GLR*** it is possible to achieve and even higher performance. Notice that the difference in average time for **GLR*** and **GLR* + repair** are barely distinguishable. This is because the **Parse Quality Confidence** flag rarely ever indicated that the parse needed repair, thus not giving the repair stage much room to yield a big improvement.

ROSE's Hypothesis Formation Stage was evaluated one final time to test the accuracy of it's heuristic for selecting between its best result from the Combination Mechanism versus keeping the parser's best result. Since the Combination Mechanism works with less information than the parser, the result from the Combination Mechanism is not always better than simply returning the set of largest non-overlapping, full sentence parses from the chart. In Figure 10.8, results with ROSE's internal heuristic are compared with those of an oracle which always selects the one with the best translation quality. We see that the results are very similar, indicating that ROSE's internal heuristic performs well. Nevertheless, there is still room for interaction to improve the quality of the result which ROSE ultimately returns since the best ranked result from the Combination Mechanism is not always the best result in reality and since sometimes the Combination Mechanism produces no acceptable results and a rephrase is required in order to improve translation quality.

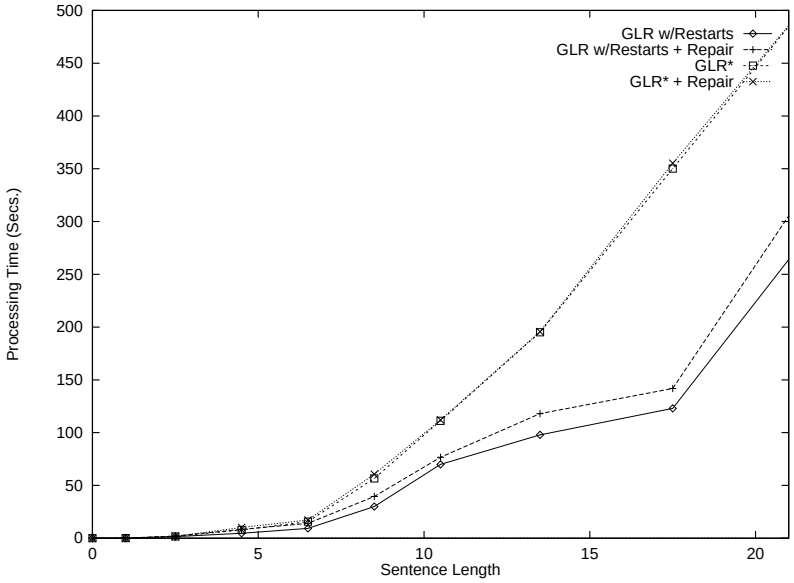

Figure 10.6: **Parse Times for Alternative Strategies**

	NIL	Bad	Partial	Okay	Perfect	Total Acceptable
MDP 5	8.4%	8.2%	6.0%	21.0%	56.4%	77.4%
GLR with Restarts	9.2%	6.4%	12.8%	19.4%	52.2%	71.6%
GLR with Restarts + Repair	0.4%	9.6%	11.8%	23.4%	54.8%	78.2%
GLR*	2.2%	8.8%	11.6%	21.4%	56.0%	77.4%
GLR* + Repair	0.6%	8.8%	10.6%	23.6%	56.4%	80.0%

Figure 10.7: Translation Quality of Alternative Strategies

	NIL	Bad	Partial	Okay	Perfect	Total Acceptable
GLR with Restarts + Repair	0.4%	9.6%	11.8%	23.4%	54.8%	78.2%
Oracle	0.4%	8.8%	10.8%	24.0%	56.0%	80.0%
GLR* + Repair	0.6%	8.8%	10.6%	23.6%	56.4%	80.0%
Oracle	0.6%	8.2%	9.8%	24.0%	57.4%	81.4%

Figure 10.8: Selection Heuristic vs. Oracle

10.4 Discussion

This evaluation demonstrates clearly that ROSE's "casting and combining" approach is significantly faster per associated improvement in translation quality than the MDP approach. While we have no doubt that increasingly more flexible versions of MDP would perform better than **MDP 5**, we have already demonstrated that even **MDP 5** is impractical in terms of its run-time performance. Thus we conclude that the two stage approach, even with a very limited flexibility parser, is a superior choice.

In fairness to the MDP approach, it should be noted that the Enthusiast parsing grammar was written keeping the limitations of Lavie's parsing algorithm in mind. It was developed based on a large data collection effort including hundreds of dialogues between speakers scheduling a meeting together. Most of the common language patterns encountered, including cases where speakers ellide portions of their sentence, are covered in the grammar. Because of this, the need for insertion in the parsing algorithm with a grammar such as this is reduced. This explains why the **GLR*** parser achieves almost the same level of performance as **MDP 5**. If the grammar had been developed for an MDP parser, on the other hand, some rules allowing the parser to form acceptable parses from fragments would not have been included. The resulting grammar in that case would have been less complex, and the time requirements of MDP would have been correspondingly reduced, as would those of GLR*. However, the comparative timing results presented here indicate the difference in time requirements of the MDP algorithm versus that of the GLR* algorithm keeping grammar complexity constant, and thus the comparison in time for the two alternative algorithms is valid for grammars of this level of complexity. Note that since the MDP parser is *more* flexible than the GLR* parser, any flexibility built into the parsing grammar would certainly not hurt the MDP parser's output quality. And though GLR* without repair would likely perform worse with a grammar written specifically for an MDP parser, since fragments would likely not parse as complete sentences, GLR* with repair would still perform as well since the parser would still produce the same partial analyses. The GLR*'s ability to produce quality partial analyses is related only to the compositionality of the grammar. There is no reason why a GLR* grammar would necessarily be written in a more compositional manner than an MDP grammar.

ROSE with **GLR*** is demonstrated to achieve almost 2% more acceptable translations that ROSE with **GLR with restarts**. Though these results indicate that more skipping yields better results, and though this is true on average, when the quality of individual sentences parsed with the alternative methods are compared, it becomes clear that in some cases more skipping makes the result worse (1.5% of the time over the whole corpus

of 500 sentences). Since more skipping makes more alternative analyses available for the parser to choose from, and since it must make a trade-off between the skipping penalty and the statistical score, more ambiguity makes more opportunity for the parser to select the wrong analysis. Thus, **GLR*** is not necessarily the ultimate partial parser for two stage approaches like ROSE. Instead, some compromise between **GLR*** and **GLR with restarts** would likely yield better results in less time than **GLR* + repair**. Interesting avenues for future research include looking into alternative beam techniques for effectively limiting skipping, testing for the optimal level of skipping, and searching for a more effective trade-off between skipping and statistical score in selecting the best parse and extracting stable chunks from the parser's internal chart. Another alternative is giving multiple analyses for ambiguous portions of the sentence to the Combination Mechanism. This would give the Combination Mechanism the means to produce a reasonable interpretation in the cases where it is currently impossible because the parser is returning chunks from the wrong ambiguity. Additionally, the Interaction Mechanism could be used to disambiguate highly ambiguous parses as well as the output from the Combination Mechanism. Since both the parser and the Combination Mechanism produce structures with the same format, this can be done without modification to the Interaction Mechanism.

The comparison between **GLR with restarts + repair** and **GLR*** with no repair indicate that the most general result made evident by this evaluation is that two step approaches yield better performance per cost in time than one stage approaches. The reason is that the more constrained parsing algorithm made use of in the two step approach is simpler than the more flexible parsing algorithm used in the one step approach, and thus more efficient in practice. The knowledge gained in the first step constrains the search in the second step. In this way, the second step can also run efficiently. Additionally, dividing the process into two steps makes it possible to avoid the unnecessary computation of the second step in cases where the second step is not necessary.

Though the ROSE approach is demonstrated here to achieve a significant percentage of its potential improvement, it is important to consider also the roughly 20% of the corpus where the ROSE approach is not able to achieve any improvement. Though a small portion of these sentences are such that improvement on the parsing side would make it possible to produce sufficient chunks to construct an acceptable hypothesis with the Combination Mechanism, in the great majority of these cases, either the meaning of the sentence cannot be represented in the Enthusiast meaning representation or at least one important portion of the sentence is worded in such a way that nothing in the parsing

	Bad	Okay	Perfect	Total Acceptable
Parser	85.0%	12.0%	3.0%	15.0%
ROSE 0	64.0%	28.0%	8.0%	36.0%
ROSE 1	61.39%	28.71%	9.9%	38.61
ROSE 2	59.41%	28.71%	11.88%	40.59%
ROSE 3	53.47%	32.67%	13.86%	46.53%
IRLH 1	83.67%	16.33%	0.0%	16.33%
IRLH 2	75.51%	22.45%	2.04%	24.49%
IRLH 3	69.39%	26.53%	4.08%	30.61%

Figure 10.9: Translation Quality For ROSE vs. IRLH

grammar makes it possible to construct any sort of correct analysis for the corresponding portion of the sentence.

A typical example from the evaluation corpus of a sentence that cannot be represented in the Enthusiast meaning representation is, "Fat Albert's on.". The Enthusiast meaning representation does not contain any concepts covering watching television programs or any specific television programs. If the meaning of the sentence cannot be represented in the meaning representation, then no approach to prepair will work. And even interaction with the user will not help. Examples like this constitute roughly 10% of the corpus.

A typical example of a sentence where the meaning can be represented but the wording prevents the parser from producing an analysis is "Why don't you just shoot a time at me?". In these cases, approaches relying on the parser to find a match, including both skipping approaches like GLR* and more flexible approaches like MDP. Though it is possible in theory for MDP to map sentences like this onto corresponding sentences with the correct meaning that do match the grammar using insertions and deletions, if a significant portion of the meaning doesn't match anything in the grammar, then the sentence with the correct meaning that matches the grammar won't necessarily be the minimum distance parse. Examples like this constitute roughly 5% of the corpus. Since the meaning of the sentence can be represented in the meaning representation language, then interaction has the potential for improving the system's performance on these sentences.

10.5 Evaluating the Interaction Stage

The purpose of evaluating the Interaction with the User stage is both to show the useful role of interaction in repair as well as to demonstrate the preferability of separating Hypothesis Formation from Interaction with the User rather than interleaving them as in IRLH.

First ROSE automatically selected 100 sentences from a different test set than that used in the previous evaluation. It selected these 100 sentences from a larger set of 500 sentences. These 100 sentences are the first 100 sentences in the set which the Parse Quality flag indicated needed repair. It should be kept in mind, then, that this corpus is considerably more noisy than the one used in the previous evaluation.

Out of these 100 sentences, 50 were randomly selected such that half of the sentences were assigned to testing the ROSE approach and the other 50 were assigned to testing IRLH. In a complement set, sentences which were assigned to ROSE in the first set were assigned to IRLH and vice versa. In this way, there were two tracks through the data such that half of the sentences were assigned to ROSE and half to IRLH.

Nine native speakers of English who had never previously used the JANUS translation system participated in the evaluation of the Interaction Mechanism. The results for the first user were not included in the tabulated results because it was determined after he participated that some informal training was necessary in order to familiarize the users with how to answer the questions. Immediately prior to their participation in the experiment, the remaining 8 users were exposed for 10 minutes to a set of 500 sentences generated by the JANUS system in order to become familiar with what level of language to expect. For 10 additional minutes the users were coached through a set of training examples in order to practice interacting with the system. The results from the training examples are not included in the tabulated results. The experiment was set up in such a way that half of the participants were assigned to the first track through the data and half were assigned to the other track through the data. In this way, each participant interacted with both the ROSE and IRLH approach, and each sentence was tested four times for each approach. The initial result before interaction, as well as the result after each question, was recorded. The translation quality of each alternative structure recorded was graded based on the quality of the generated translation. Scores of Bad, Okay, and Perfect were assigned. No distinction between Nil, Bad, and Partial translations was made in this evaluation since this distinction is only important before interaction takes place. Before interaction, it is useful to produce partial solutions, if nothing else, in order to make interaction easier for the user. Once interaction takes place, however, all that is important is the percentage of Okay and Perfect translations which are achieved.

For both ROSE and IRLH, the system was allowed to ask a maximum of three questions. This limitation was placed on the system in order to keep the task from becoming too tedious and time consuming for the users. It was estimated that three questions was approximately the maximum number of questions that users would be willing to answer per

sentence. Considering that repair occurs in about a third of the sentences, and interaction takes place in the majority of cases where repair takes place, in an interactive system, users might not even be willing to answer that many questions.

From these results it was possible to compute average performance per questions asked for each approach and to compare this with performance without interaction. The results are presented in Figure 10.9. These results indicate that although results in IRLH improve as maximum number of questions is increased, the quality after three questions never even reaches that of ROSE without interaction. With three questions IRLH achieves an 18% reduction in error rate where ROSE without interaction achieves a 25% reduction in error rate. And interaction increases ROSE's average translation quality above that of ROSE without interaction. With three questions, ROSE achieves a 37% reduction in error rate. Additionally, the improvement over the parser's performance per maximum number of questions for ROSE is higher than that of IRLH since it allows for engaging in a more diverse set of types of interactions.

10.6 Evaluating Discourse ROSE

In a final evaluation, one method for using discourse information in repair was explored. In this evaluation, each question generated during the interaction evaluation was tested to see whether the Enthusiast discourse processor was able to distinguish between the same sets of alternative meaning representation structures as the original feature corresponding to the question. In 21.6% of the cases, the discourse processor provided sufficient information for reformulating the system's query on the task level.

4 human judges were asked to grade pairs of questions, assigning a score between 1 and 5 for relevance and form and indicating which question they would prefer to answer. They were instructed to think of relevance in terms of how useful they would expect the question would be in helping a computer understand the sentence the question was intended to clarify. For form, they were instructed to evaluate how natural and smooth sounding the generated question was.

ROSE received on average 2.7 for form and 2.4 for relevance. Discourse ROSE, on the other hand, received 4.1 for form (62% reduction in error) and 3.7 for relevance (49% reduction in error). Subjects preferred Discourse ROSE's question in 73.6% of the cases, expressed no preference in 14.8% of the cases, and preferred regular ROSE in 11.6% of the cases. Though the Discourse ROSE question was not preferred universely, this evaluation supports the claim that humans prefer to receive clarifications on the task level and indi-

cates that further exploration in using discourse information in repair, and particularly in interaction, is a promising avenue for future research.

10.7 Conclusions

The results of these evaluations indicate that ROSE has met its goal of achieving a high level of robustness more efficiently than both MDP and IRLH.

Part VI

Conclusions

Chapter 11

Conclusions and Future Directions

The main contribution of this dissertation is the development of the ROSE approach to handling the problem of extragrammaticality in a effective and efficient way. The evaluation demonstrates that the two stage ROSE approach robustly extracts the meaning from the user's extragrammatical utterance efficiently and without placing an undue burden on the user in terms of interactional effort. Its smart use of flow-of-control flags allows it to avoid wasting extra resources where they are not likely to yield any improvement in parse quality. Finally, because the ROSE approach does not rely on any hand crafted repair rules or additional knowledge sources dedicated to repair, it is a completely general and portable solution, being both domain independent and language independent.

11.1 Summary Of Results

I argue that the ROSE approach of separating the Partial Parsing and Combination steps is more efficient than placing the full burden of robustness on a single parsing algorithm. The ROSE approach with a limited flexibility parser performs an order of magnitude faster than a limited version of MDP while producing 1% more acceptable translations. ROSE also performs on average about twice as fast as GLR* on sentences longer than 12 words long while producing about 1% more acceptable translations. These results are discussed in greater depth in Chapter 10.

Another goal of this work has been to demonstrate that it is more efficient to separate Repair Hypothesis Formation from User Interaction rather than interleaving them. In other words, a set of alternative ways of fitting the whole set of fragments from the partial parse (Global Repair Hypotheses) is constructed before any queries are generated rather than generating a query to verify each repair step (Local Repair Hypotheses). In Chapter 10, ROSE without interaction is demonstrated to achieve a higher translation quality than IRLH with maximum of 3 questions. With three questions IRLH achieves

an 18% reduction in error rate where ROSE without interaction achieves a 25% reduction in error rate. And interaction increases ROSE's average translation quality above that of ROSE without interaction. With three questions, ROSE achieves a 37% reduction in error rate. Additionally, the improvement over the parser's performance per maximum number of questions for ROSE is higher than that of IRLH since it allows for engaging in a more diverse set of types of interactions.

Besides being more efficient, ROSE is also arguably more effective. When humans collaborate for the purpose of understanding, they direct their questions towards information which is necessary for accomplishing their task (Clark and Schaefer, 1989; Clark and Wilkes-Gibbs, 1986). When questions are directed at clarifying the speaker's meaning, rather than furthering the shared task, speakers become agitated (Garfinkel, 1964). By delaying the interaction until hypotheses about the speaker's whole meaning are formed, it is possible to focus the interaction on the task level rather than on the language level. It is demonstrated in Chapter 10 that the Enthusiast discourse processor can be used to distinguish between classes of hypothesized structures in terms of their affect on the discourse state. This makes it possible to generate queries in terms of affect on discourse state rather than in terms of literal meaning in 21.6% of the cases. In a small pilot study it was determined that human judges grade the form and relevance of discourse generated questions significantly higher than the meaning level questions. And human judges indicated that they preferred the task level questions in 73.6% of the cases.

11.2 Future Directions

Though this dissertation represents the end of one segment of this work, exciting avenues for further investigation exist at all levels of the ROSE approach in addition to that already mentioned above.

11.2.1 Improving ROSE's performance

The first general direction for future research is improving the performance of the ROSE approach.

11.2.1.1 Flow of Control

As discussed in Chapter 6, ROSE's three status flags direct the flow of control through ROSE and determine how much effort is spent on interpreting each sentence. Though these status flags are demonstrated in this dissertation to be effective in creat-

ing an overall approach in which the system achieves a greater efficiency than either MDP or IRLH, there is nevertheless room for improvement. In particular, only 85% of the sentences which are marked as needing repair have a poor translation quality. Additionally, the Combination Mechanism is not able to construct a high quality repair in every case where the Repairability Indicator indicates that a repair is possible. In over half of the cases, the partial parser is not able to construct chunks to represent all of the important portions of the sentence, making repair with the Combination Mechanism impossible to produce an acceptable result. And in many cases, Interaction does not yield an improvement over ROSE without interaction since in 83% of the cases where an acceptable hypothesis is constructed it appears as the top ranking hypothesis. Nevertheless, the Repair Quality Confidence flag acts very conservatively, indicating that interaction is necessary in the majority of cases. Improving the accuracy of these judgements would improve ROSE's efficiency both in terms of hypothesis formation time and number of questions asked.

11.2.1.2 Chunk Construction

In some cases more skipping yields a better result than GLR with only restarts, but in some cases the added ambiguity of the more flexible approach yields a worse result. More effective beaming and a better trade off between statistics and skipping in the parse scoring mechanism would result in the production of higher quality chunks. The Hypothesis Formation stage can only construct hypotheses out of the chunks that are extracted from the parser. The higher quality the chunks, the more potential for constructing quality repair hypotheses. It would also be interesting to explore the possibility of including multiple analyses covering the same portion of the sentence as another potential solution to the ambiguity problem.

11.2.1.3 Genetic Search

Once a set of quality chunks is obtained, they must be assembled correctly. The most influential factor in the Combination Mechanism's ability to construct high quality repair hypotheses is the accuracy of the fitness function in ranking alternative repair hypotheses. What makes it difficult to train an accurate fitness function is that it must combine information from difference sources which sometimes make conflicting predictions. This is an as yet unsolved problem in machine learning. If a satisfactory solution to this problem could be found, the performance of the Combination Mechanism would be greatly improved.

Another interesting direction for future research would be to explore adjustable runtime parameters for the genetic search in terms of population size, maximum program depth, and number of generations. More complex examples require more resources than simple examples. In the current version, the same amount of resources are channeled to every example regardless of the complexity of the particular case.

Directions for improvement of the genetic search are also suggested by work in the semantic disambiguation community. In (Beale, Nirenburg, and Mahesh, 1996), an approach called Hunter-Gatherer is presented for efficiently solving complicated constraint satisfaction problems in natural language interpretation. This approach combines constraint satisfaction, branch-and-bound, and solution synthesis. Constraint satisfaction and branch-and-bound are used for eliminating highly non-optimal and impossible solutions from consideration, thus considerably reducing the search space size. It then uses solution synthesis to compose all of the optimal solutions. It differs from previous approaches in that it uses the result of the first stage to divide the problem into smaller subproblems, and then uses this breakdown to guide the solution synthesis search rather than wasting time first testing all ways of combining pairs of constraints, and then pairs of combined constraints, and so on. The idea behind this approach could lead to a natural extension of the application of genetic programming discussed in this document. In (Koza, 1994), Koza presents a more structured version of genetic programming where subroutines are evolved and then used as steps in a more abstract program. If a constraint satisfaction and branch-and-bound approach could be used to divide the repair problem into pieces, each piece could be solved with an evolved subprogram and then combined in a more abstract evolved function. Such an approach would likely yield a savings in time on very complex cases, but the overhead might make the whole process slower for simpler cases. The optimal balance between structure and complexity is a question that is yet to be answered.

11.2.1.4 Discourse in Repair and Interaction

In Discourse ROSE, discourse information is used only for determining what significance a distinction between interlingua structures has on the formation of the discourse state. One can imagine other roles discourse information could play in repair. First of all, it would be interesting to explore the possibility of using discourse information for filling in missing information in repair hypotheses from context or for confirming the accuracy of repairs without interaction by testing whether they are confirmed by context.

11.2.1.5 Other Types of Interaction

One distinction between IRLH and ROSE is that IRLH has the ability to generate hypotheses about filling in missing information. As discussed in Chapter 7, it is not feasible to fill in missing information during ROSE's Hypothesis Formation stage since without confirming these guesses with the user, the likelihood is that incorrect information will be filled in. However, it would be possible to generate and test hypotheses about missing information during the Interaction with the User stage. However, it is important to continue to keep the number of questions down to three or less in order to avoid overburdening the users.

Selecting the best from a set of alternative repair hypotheses is very similar to the problem of parse disambiguation. As already mentioned, the ROSE approach to interaction can very straightforwardly be applied to the problem of parse disambiguation. Techniques for parse disambiguation can also be adapted for the problem of selecting the best alternative repair hypothesis. In particular, an efficient method for generating smooth sounding clarifying questions for disambiguation is discussed in (Tomita, 1986b). In this approach, an explanation template is attached to each rule in the parsing grammar. Ambiguity packing, creating a packed-shared-parse forest structure, allows ambiguities to be grouped in order to keep the number of questions down to a reasonable level. The explanation templates associated with clusters of ambiguities are used for generating short multiple choice lists that the user uses in order to narrow down on the correct ambiguity.

This approach could be adapted to ROSE by attaching explanation templates to slots in the meaning representation language. Because the features in ROSE group hypothesized meaning representation structures into classes, the use of features has a similar affect to using the packed-shared-parse forest in Tomita's disambiguation approach. The main distinction between this adapted approach and Tomita's original approach is that ROSE handles ambiguity on the level of the meaning representation, whereas Tomita handles ambiguity on the level of the context-free parsed structure.

Tomita's approach has both advantages and disadvantages in comparison with the ROSE approach. While such an approach would likely result in a more natural sounding interaction with the user, it requires a lot of development time for designing and attaching the explanation templates to each grammar rule. Though the ROSE approach produces some unnatural sounding queries, it requires no such effort[1]. And though it may result in less smooth sounding interactions, it has already been demonstrated to be effective nevertheless.

[1]The regular ROSE interaction approach requires only two templates, and Discourse ROSE only requires templates for making task specific distinctions, which in the Enthusiast system are two orders of magnitude fewer than the number of grammar rules.

11.2.2 Integrating ROSE with Other Types of Repair

Whereas ROSE attempts to handle all forms of extragrammaticality at parse time, some types of speech errors have been demonstrated to be possible to repair before parse time. One example of this type of work is (Heeman and Allen, 1994), where a pre-parse-time algorithm is demonstrated to be able to fix 80% of the following three types of speech repairs:

- *fresh starts*: where the speaker abandons what he was saying and starts again.
  `EX: The current plan is we take - okay let's say we start with the bananas.`
- *modifications*: where the speech-repair modifies what was said before.
  `EX: After the orange juice is at - the oranges are at the OJ factory.`
- *abrides*: where the repair consists solely of editing terms.
  `EX: We need to - um manage to get the bananas to Dansville more quickly.`

In Heeman's corpus, it was found that these types of repairs occur in 25% of the turns, and multiple ones occur in 67% of these cases. Thus, such an approach would certainly be useful for reducing the amount of extragrammaticality in a corpus. And any pre-parse-time approach like this could be easily combined with the ROSE approach by simply filtering each sentence through the preprocessing repair module before passing it to ROSE. Though it is likely that such an approach would yield a nice speed up and improvement in results, this is yet to be demonstrated. It should also be emphasized that these speech repairs do not constitute the sum total of extragrammaticality that a natural language understanding interface needs to deal with. In particular, it only covers the cases where the speaker knows that he has said something that he didn't intend to say. Because as already mentioned the user is not aware of the limitations of the language interface, the system must also deal with other cases where the user's sentence is outside of the coverage of its grammar. Thus, for best results, such an approach should be paired with a parse-time repair approach such as ROSE.

11.2.3 Applying ROSE to Intelligent Tutoring

Though the ROSE approach has been developed in the context of a machine translation system, it has always been my intention to apply this technology to intelligent tutoring. Applying ROSE to a new domain is simple, involving only designing a new meaning

representation language, writing a parsing grammar, and automatically training the information gain networks, the parser's statistical models, and the fitness function. Applying ROSE to a new language within the same domain would be even simpler, not requiring the design of a new meaning representation language. However, the templates used for generating queries would need to be translated. Applying ROSE to a new task, on the other hand, i.e., intelligent tutoring rather than machine translation, carries with it additional challenges.

- The design of the meaning representation language must be done in such a way that the representations produced by the language interpretation portion of the system provide usable information to the instructional portion of the system.

- Whereas ROSE has been demonstrated to be able to take advantage of discourse information provided by the Enthusiast Discourse processor for guiding the interaction with the user, in an intelligent tutoring system it would be useful also to take advantage of information in the student model for guiding the interaction. How this information can be applied to language understanding is a new question.

- In an intelligent tutoring system, it is important to take pedagogical considerations into account in guiding the interaction. There is a danger that students could take the system's clarifying questions as an indication that they have said something incorrect. Possible solutions to this problem include marking clarifying questions specifically as such to avoid confusion or bypassing interaction altogether so that the best result from the Hypothesis Formation stage is always returned.

- In an intelligent tutoring system, it may not always be necessary for the system to understand the user's entire utterance. If the system can find part of the user's utterance that it has reasonable confidence in its interpretation of and that it can form a useful response to, clarifying questions regarding what the user has said may not be necessary. This is an open question and requires investigation before a solid answer can be given.

Appendices

Appendix A

GLR* Parser: Background information

In this Appendix I describe the GLR* parser which I make use of in this dissertation. The GLR* parser was developed at Carnegie Mellon University by Alon Lavie. A more in depth discussion of the GLR* parser can be found in (Lavie, 1995).

The GLR* parser is an extension of Tomita's GLR parser (Tomita, 1986a). Tomita's algorithm is based on the LR parsing approach used for programming languages.

A.1 Shift-Reduce Parsing

On the simplest level, the GLR* parser is a shift-reduce parser. In a shift-reduce parser, the grammar is compiled into a set of states and transitions which constitute a finite state pushdown automaton. The states encode uncertainty about the completion of the input string as it is being processed. In this way, the parser can limit the scope of possible completions while eliminating the need to backtrack. These states are used to control the actions of the parser. Shift-reduce parsers maintain two stacks, namely a *parse stack* and an *input stack*. On the parse stack, parse states and grammar symbols alternate. The input string is stored on the input stack. As words are popped off the input stack, the parser performs actions on the parse stack in order to construct an analysis of the input sentence. Shifting refers to popping words off the input stack and moving them onto the parsing stack. Reducing refers to applying grammar rules to the symbols in the top portion of the parsing stack. For a more in-depth, simple introduction to shift-reduce parsing, see (Allen, 1995).

A.2 LR Parsing: The Foundation for GLR

An LR parser is a particular type of shift-reduce parser which makes use of a grammar table compiled into an LR table. For example, the compiled grammar table for the example grammar found in Figure A.1 can be found in Figure A.2. The table in figure

(1) S $--> $ NP VP
(2) NP $--> $ det n
(3) VP $--> $ v NP

Figure A.1: Simple Example Grammar

	Reduce	Shift				Goto		
State		det	n	v	$	NP	VP	S
0		sh1				3		6
1			sh2					
2	r2							
3				sh4			5	
4		sh1				7		
5	r1							
6					acc			
7	r3							

Figure A.2: Compiled Grammar Table

A.2 has an entry for each of the seven states in the compiled grammar. For each state there are three associated portions of the compiled LR table. In the **Reduce** section, reductions are listed for some states. For example, *r2* listed for `state 2` indicates that when the state at the top of the parser stack indicates that the parser is in `state 2`, reduction number 2, `NP --> det n`, should be applied to the stack. The second portion of the table is the **Shift** portion. An entry exists for each pair consisting of a lexical category and a state. The entry *sh1* corresponding to lexical category `det` and `state 4` indicates that if an item of category `det` is shifted onto the parse stack when the parser is in `state 4`, the new parser state should be 1. The entry *acc* listed for lexical category `$` in `state 6` indicates that if the input stack is empty when the parser is in `state 6`, the input is accepted and the parse is completed. Empty entries in this section indicate that if a parser shifts an item of the corresponding category in the corresponding state, the input should be rejected as ungrammatical. The **Goto** portion of the table indicates what the new state should be after a reduction has been applied and a non-terminal symbol is pushed onto the parse stack in particular states.

At parse time, the parser performs shift and reduce actions on a stack as determined by the parsing table, the current state, and the next word on the input stream. Since each state represents uncertainty about possible completions of the input string, the parser's

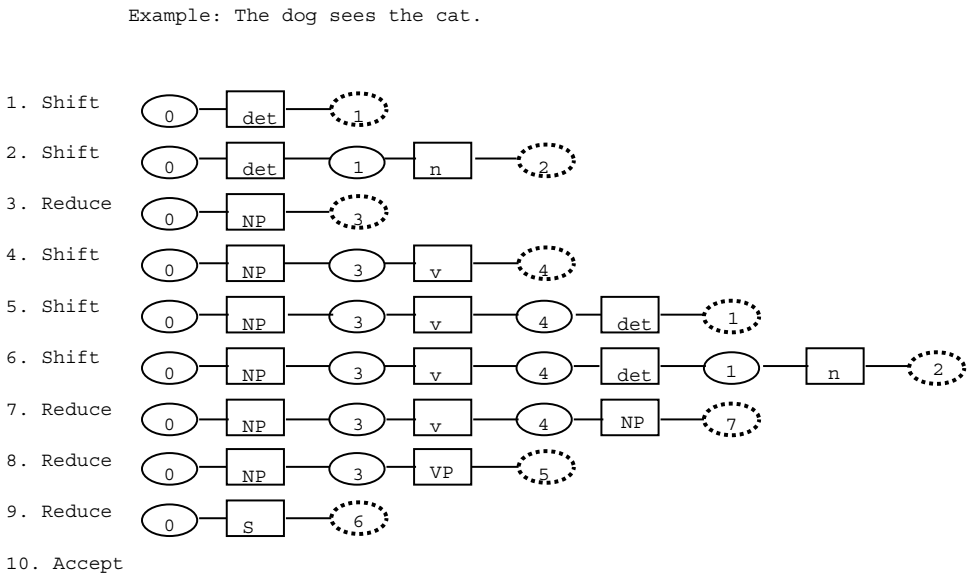

Figure A.3: **Stack at Each Derivation Step**

actions are context sensitive, making top-down predictions based on the initial segment of the sentence processed so far. A sample derivation using the compiled grammar in Figure A.2 can be found in Figure A.3.

In the original LR parsing framework, at each step, the parser can perform exactly one of the following four actions:

- **Shift** the next word on the input stream onto the parse stack, and then move to a state specified by the current state and the current word. See steps 1, 2, and 4-6 in Figure A.3.
- **Reduce** the current parse stack by applyng a specific grammar rule. Symbols are popped from the stack each corresponding to an element from the right hand side of the rule and are replaced by a single symbol representing the left hand side of the rule. After this the parser moves into a new state specified by the symbol on the left hand side of the rule and the state in which the first symbol on the right hand side of the rule was pushed. See steps 3 and 7-9 in Figure A.3.
- **Accept** the input. See step 10 in Figure A.3.
- **Reject** the input.

The sample derivation found in Figure A.3 contains 10 steps. Initially, the parser is in `state 0`. When "the" which is an element of category `det` is popped off of the input stack, it is shifted onto the parse stack. The parser then moves into `state 1`. In step 2,

"dog", an element of category `n`, is popped off of the input stack and shifted onto the parse stack. The parser then moves into `state 2`. Because reduction 2 can be applied to the parse stack in `state 2`, reduction 2 is applied in step 4. The resulting `NP` is pushed onto the parse stack and the parser moves into `state 3`. In steps 4 through 6, the remaining three lexical items are shifted onto the parse stack, in the same way the first two items were shifted. This leaves the parser in `state 2`. Just as in step 2, reduction 2 is applied. This leaves the parser in `state 7`. Since reduction 3 can be applied in `state 7`, it is applied. The `NP` and the `v` are popped off the parse stack, and the resulting `VP` is pushed. This leaves the parser in `state 5`. In step 9, since reduction 1 applies in `state 5`, it is applied, and the resulting `S` is pushed onto the parse stack, leaving the parser in `state 6`. Since this is the end of the input sentence and the grammar table indicates that an empty input stack in `state 6` can be accepted, the input is accepted in step 10, and the parse is completed.

The parse stack of the LR parser contains two types of elements, *state* nodes and *symbol* nodes. At the bottom of the stack is a node which corresponds to the initial state. When a *shift* action is performed, first a symbol node is pushed corresponding to the input token, and then a state node corresponding to the new state. The state at the top of the stack is known as the *active* state. This is the current state which is used along with the current input token to determine which action to perform. For example, at the end of step 3 in Figure A.3, the active state is 3. When v is shifted onto the top of the stack, the parser moves into state 4 since the entry in state 3 for shifting a v indicates that the new state should be 4. Thus, in step 4 in Figure A.3, first a symbol node for v is shifted and then a state node for state 4 is shifted.

When a reduction is performed, for each symbol on the right hand side of the rule, both a state node and a symbol node are popped. After this, the state node which is exposed is the state in which the first symbol on the right hand side of the rule was pushed. This makes it possible to compute what state to move to after the reduction has been completed and the symbol from the left hand side of the rule is pushed. For example, an *NP* reduction takes place in step 7 in Figure A.3. At the end of step 6, the parser is in state 2. The compiled grammar table in Figure A.2 indicates that reduction 2 should be performed in this state. In Figure A.1 we see that there are two symbols on the right hand side of reduction 2. So the state 2 state node, the n symbol node, the state 1 state node, and the *det* symbol node would all be popped off the top of the stack. This would expose the state 4 state node. This is the state the parser was in when the *det* symbol node was originally pushed. Now an *NP* symbol node is pushed in its place. The grammar table in Figure A.2 indicates that when an *NP* symbol node is pushed in state 4, the new state is 7.

So a new state node corresponding to state 7 is pushed on the top of the state and becomes the active state.

Because each state in the compiled grammar represents a set of possible completions of the portion of input processed so far, pushing a symbol node of a particular category in a particular state in the LR approach is the same as matching an element in the right hand side of a set of rules in a chart parser. Once that element has been matched, the state is changed to one corresponding to a new set of possible completions. For a more in-depth discussion of LR parsing see (Aho and Ulman, 1972).

A.3 Tomita's GLR: the foundation for GLR*

The same actions are employed in the GLR framework as in the LR framework, but an extension is made in order to make it possible to handle ambiguity, which is inherent in human languages. Where in the original LR framework exactly one of the four actions listed above was performed, totally determined by the state and the current input token, in the GLR framework, multiple actions are possible in order to allow for alternative analyses for the same input sequence. Because separate actions may leave the parser in different states, it must be possible to have more than one top node on the parse stack, one for each current state. In order to keep track of these alternative states, each of which corresponds to a path through the compiled grammar table for the input processed so far, the parse stack is implemented as a graph structured stack rather than as a simple stack. Because each state encodes a set of possible completions of the input string given the portion processed already, each path through the graph structured stack similarly corresponds to an alternative set of possible completions of the input stream given that portion already processed. See (Tomita, 1991; Tomita, 1986a) for a more in-depth discussion of GLR parsing.

A.4 Extensions in GLR* for Skipping

The idea behind Lavie's approach is to identify and parse the maximal subset of the input string which is found to be grammatical according to the parsing grammar. Where the original GLR parser only shifted onto the active state nodes at the top of the graph structured stack, in the GLR* parser skipping is performed by shifting onto previous states in the graph structured stack, thus skipping over the words in between the corresponding previous word and the current word.

The GLR* parser also involves three other significant extensions over the original GLR algorithm: namely, using a beam to limit the size of the frontier on the GSS, statistical

disambiguation based on probabilities assigned to individual parser actions, and a number of parse selector heuristics. These are all discussed in depth in (Lavie, 1995).

Appendix B

Interlingua Specification

This appendix contains a description of the meaning representation specification that is used both by IRLH and ROSE. It is called an Interlingua Representation because it is language independent. What distinguishes both IRLH and ROSE from other two stage repair approaches (Hobbs et al., 1991; Ehrlich and Hanrieder, 1996; Danieli and Gerbino, 1995) is that neither of them rely on any hand crafted repair rules. Instead they have the ability to search for an acceptable combination of partial analyses by making reference to the meaning representation specification which describes the meaning representation language. This ability makes both IRLH and ROSE completely portable. IRLH and ROSE can both be used in any system where the meaning representation language can be described in the format specified here.

Though this meaning representation specification is knowledge which must be encoded by hand, it is knowledge which can be used by all aspects of the system, not only the repair module as is the case with repair rules.

B.1 Abstract Description

The meaning representation is assumed to describe a language of frame based feature structures. Each frame encodes a concept in the domain. The set of frames in the meaning representation are arranged into subsets which are assigned a particular type. Each frame is associated with a set of slots. The slots represent relationships between feature structures. Each slot is associated with a type which determines the set of possible fillers for that slot.

Figure B.1 contains an example of a feature structure in the Enthusiast interlingua representation. This feature represents the meaning of the sentence "I'm busy all next week." The top level frame represents what the main idea of the sentence is. In the example, the top level frame `*busy` indicates that the main idea expressed in the sentence

```
((speech-act (*multiple*
                *state-constraint *reject))
 (sentence-type *state)
 (frame *busy)
 (who ((frame *i)))
 (when
     ((frame *special-time)
      (next week)
      (specifier (*multiple* all-range
                             next)))))
```

Figure B.1: **Sample interlingua representation returned by the parser for "I'm busy all next week."**

is about being busy. Slots indicate a semantic relationship between the concept represented by the frame and that of the feature structure which is a filler of the slot, which can be either a frame or an atomic value. In the example, the `who` slot indicates that the feature structure inserted in it represents the one who is busy. In this case, the frame `*i` indicates that the speaker of the sentence is the one who is busy. The `when` slot indicates that the feature structure inserted in it represents the time during which the state of being busy is true. Thus, in this representation meanings are built up compositionally.

In the top level frame, additional slots are added by the parser for the sentence type and the speech act. Sentence type roughly corresponds to mood, i.e. `*state` is assigned to declarative sentences, `*query-if` is assigned to yes/no questions, and so on. The speech act indicates what function the utterance performs in the discourse context. In the example in Figure B.1, the parser could not determine out of context which of two potential speech acts was most appropriate for the sentence. So multiple possibilities were assigned. In every other slot, however, `*multiple*` indicates a list of fillers rather than a set of possibilities.

The interlingua specification determines the set of possible interlingua structures. It is composed of BNF-like rules which specify subsumption relationships between types of feature-structures, as in Figure B.2, or between types of feature-structures and feature-structure specifications, as in B.3.

A feature structure specification is a feature structure with slots which are filled in with types rather than with atomic values or feature structures. Feature structure specifications are the leaves of the subsumption hierarchy of interlingua specification types. The

$$
\begin{aligned}
(< TEMPORAL > = & < SIMPLE-TIME > \\
& < INTERVAL > \\
& < SPECIAL-TIME > \\
& < RELATIVE-TIME > \\
& < EVENT-TIME > \\
& < TIME-LIST >)
\end{aligned}
$$

Figure B.2: **Sample interlingua specification rule for expressing a subsumption relationship between type $< TEMPORAL >$ and more specific temporal types.**

$$
\begin{aligned}
(< BUSY > = & ((\text{frame *busy}) \\
& (\text{topic} < FRAME >) \\
& (\text{who} < FRAME >) \\
& (\text{why} < FRAME >) \\
& (\text{when} < TEMPORAL >) \\
& (\text{how-long} < LENGTH >) \\
& (\text{degree [DEGREE]})))
\end{aligned}
$$

Figure B.3: **Sample interlingua specification rule for expressing a subsumption relationship between the type $< BUSY >$ and the feature-structure specification for the frame *busy.**

type associated with a slot is the most general type of structure which can be inserted into that slot. Thus, any feature structure subsumed by the type associated with a slot can be a filler for that slot. Notice in Figure B.3 that some types are enclosed in angled brackets whereas others are enclosed in square brackets. The convention is that types enclosed in square brackets denote atomic fillers whereas types enclosed in angled brackets denote frames.

Since every feature structure has a type, these rules can be used to determine whether a particular feature structure can "fit" into a particular slot in another feature structure. Because the interlingua representation is defined independently of the repair module both for ROSE and for IRLH, the same implementation can be used with any other meaning representation defined in the same format. And with a slight modification to

the code which parses the meaning representation specification, it can be used with other feature structure based meaning representations as well.

B.2 Portions of the Interlingua Representation

This section contains some portions of the interlingua specification. In the next section are some interlingua representation structures for some additional sentences.

```
(<TOP-LEVEL-FRAME> =
<*FIXED-EXPRESSION>
<*PREDICATE>
<FRAGMENT>)

(<*FIXED-EXPRESSION> =
<ADDRESS-PERSON>
<APOLOGIZE>
<GREET>
<RESPOND>
<THANK>
<YOU+RE-WELCOME>
<INTERJECT>)

(<*PREDICATE> =
<ABANDON>
<AGREE>
<ARRIVE>
<ASK>
<AVOID>
<AWAY>
<BAD>
<BOOKED>
<BRING>
<BUSY>
<CALL>
<CAUSE>
```

<CHECK>

<CHOOSE>

<CLARIFY>

<COME>

<CONSIDERING>

<CONTINUE>

<CORRECT>

<DESIRED>

<DIFFICULT>

<DISCUSS>

<DO>

<EASY>

<EAT>

<ELLIPSIS>

<END>

<ENOUGH>

<EXIST>

<EXTEND>

<FIND>

<FRAME-LIST>

<FREE>

<GIVE>

<GO>

<GOOD>

<HOW>

<HUNGRY>

<IN-TOWN>

<INFORM>

<INTERESTING

<KNOW>

<LATE>

<LEAVE>

<LET>

<LISTEN>

<MARK>

<MEET>
<MENTION>
<MISS>
<NEAR>
<NEEDED>
<OFFER>
<OPPOSITE>
<OUT-OF-TOWN>
<REASONABLE>
<RECEIVE>
<RETURN>
<RUSH>
<SAME>
<SCHEDULE>
<SEND>
<SETTLED>
<START>
<STRANGE>
<SUGGEST>
<TIRED>
<UNDERSTAND>
<UTILIZE>
<VISIT>
<WAIT>
<WORK>
<WORRY>
<UNDESIRED>
<WRONG>
<RELAX>
<SETTLE>
<SHOP>
<SOLVE>
<SWIM>)

(<*ACTION> =

```
<BRING>
<CALL>
<CHECK>
<CHOOSE>
<COME>
<CONTINUE>
<DISCUSS>
<EAT>
<FIND>
<GIVE>
<LEAVE>
<MEET>
<SCHEDULE>
<SEND>
<RELAX>
<SETTLE>
<SHOP>
<SOLVE>
<SWIM>)

(<BUSY> =
((frame *busy)
 (tense [TENSE])
 (aspect [ASPECT])
 (negative [VALUE+])
 (degree [DEGREE])
 (when-0 <*WHEN>)
 (who <*WHO>)
 (when <*WHEN>)
 (why <*EVENT>)
 (purpose <*EVENT>)
 (how-long <LENGTH>)))

(<FREE> =
((frame *free)
```

```
 (tense [TENSE])
 (aspect [ASPECT])
 (negative [VALUE+])
 (degree [DEGREE])
 (when-0 <*WHEN>)
 (who <*WHO>)
 (when <*WHEN>)
 (why <*EVENT>)
 (purpose <*EVENT>)
 (how-long <LENGTH>)))

(<CONTINUE> =
((frame *continue)
 (when-0 <*WHEN>)
 (tense [TENSE])
 (aspect [ASPECT])
 (negative [VALUE+])
 (who <*WHO>)
 (when <*WHEN>)
 (action <*action>)
 (where <*WHERE>)
 (what <*WHAT>)
 (how-long <LENGTH>)))

(<SCHEDULE> =
((frame *schedule)
 (tense [TENSE])
 (aspect [ASPECT])
 (attitude [ATTITUDE])
 (when-0 <*WHEN>)
 (negative [VALUE+])
 (who <*WHO>)
 (with-whom <*WHO>)
 (when <*WHEN>)
 (where <*WHERE>)
```

```
 (what <*WHAT>)
 (how-long <LENGTH>)))

(<GOOD> =
((frame *good)
 (tense [TENSE])
 (aspect [ASPECT])
 (negative [VALUE+])
 (degree [DEGREE])
 (purpose <*EVENT>)
 (for-whom <*WHO>)
 (what <*WHAT>)
 (when-0 <*WHEN>)
 (when <*WHEN>)
 (who <*WHO>)
 (how-long <LENGTH>)
 (where <*WHERE>)))

(<OUT-OF-TOWN> =
((frame *out-of-town)
 (tense [TENSE])
 (aspect [ASPECT])
 (when-0 <*WHEN>)
 (who <*WHO>)
 (when <*WHEN>)
 (where <*WHERE>)
 (why <*EVENT>)
 (negative [VALUE+])
 (how-long <LENGTH>)))

(<DO> =
((frame *do)
 (tense [TENSE])
 (aspect [ASPECT])
 (attitude [ATTITUDE])
```

```
 (negative [VALUE+])
 (who <*WHO>)
 (what <*WHAT>)
 (when <*WHEN>)
 (with-whom <*WHO>)
 (where <*WHERE>)))

([TENSE] =
past
present)

([ASPECT] =
progressive
perfect)

([ATTITUDE] =
*desired
*how-about
*impossible
*let-s
*needed
*possible
*should
*should-not
*supposed
*undesired
*unneeded
*unwilling
*willing
*shall)

(<*WHO> =
<I>
<CLIENT>
<DOCTOR>
```

<EACH-OTHER>

<SPOUSE>

<FRIEND>

<GIRLFRIEND>

<BOYFRIEND>

<PERSON>

<PERSON-NAME>

<PROFESSOR>

<STUDENT>

<THEY>

<HE>

<SHE>

<WE>

<YOU>

<YOU-KNOW-WHO>)

(<*WHAT> =

<ADDRESS>

<ANYTHING>

<APPOINTMENT>

<BREAKFAST>

<BRUNCH>

<BUSINESS>

<CALENDAR>

<CHOICE>

<COFFEE>

<DINNER>

<DRINK>

<EVENT-LIST>

<IT>

<LETTER>

<LUNCH>

<MAIL>

<MEETING>

<SOMETHING>

<EVERYTHING>
<PART>
<PHONE-NUMBER>
<PLAN>
<PRO>
<PROJECT>
<QUESTION>
<TEA>
<THIS>
<THAT>
<THEY>
<THING>
<TIME>
<TIME-SLOT>
<YOU-KNOW-WHAT>
<WHAT>
<WHICH>
<*ACTIVITIES>
<*TRANSPORTATIONS>
<*WHERE>
<*WHEN>
<*WHO>
<WHAT-LIST>)

(<*WHEN> =
<SIMPLE-TIME>
<SPECIAL-TIME>
<RELATIVE-TIME>
<INTERVAL>
<EVENT-TIME>
<TIME-LIST>
<THIS>
<THAT>)

(<SIMPLE-TIME> =

```
((frame *simple-time)
 (minute [NUMBER-VALUE])
 (hour [NUMBER-VALUE])
 (day [NUMBER-VALUE])
 (month [NUMBER-VALUE])
 (day-of-week [DAY-OF-WEEK])
 (time-of-day [TIME-OF-DAY])
 (am-pm [AM-PM])
 (specifier [SPECIFIER])))

(<*WHERE> =
<ADDRESS>
<CAFETERIA>
<BUILDING>
<CITY>
<CITY-NAME>
<CONFERENCE-ROOM>
<COUNTRY-NAME>
<DOOR>
<FLOOR>
<HERE>
<SPECIAL-PLACE>
<MALL>
<LIBRARY>
<LOUNGE>
<OFFICE>
<PLACE>
<PLACE-LIST>
<RESTAURANT>
<RESTAURANT-NAME>
<STATE-NAME>
<THERE>
<UNIVERSITY>
<WHERE>
<YOU-KNOW-WHERE>)
```

```
(<CAFETERIA> =
((frame *cafeteria)
 (relation [PLACE-RELATION])
 (whose <*WHO/*WHAT>)
 (modifier <*MODIFIER>)
 (specifier [SPECIFIER])))

(<LIBRARY> =
((frame *library)
 (relation [PLACE-RELATION])
 (whose <*WHO/*WHAT>)
 (modifier <*MODIFIER>)
 (specifier [SPECIFIER])))

(<*HOW-LONG> =
<LENGTH>
<LENGTH-LIST>)

(<LENGTH> =
((frame *length)
 (specifier [SPECIFIER])
 (quantity [QUANTITY])
 (unit [UNIT])))

(<*EVENT> =
<*PREDICATE>
<*WHAT>)
```

B.3 Examples

Example 1:

I'd like to continue this meeting at a later date.

```
(((FRAME *CONTINUE)
  (SENTENCE-TYPE *STATE)
  (WHEN
    ((SPECIFIER
       (*MULTIPLE* INDEFINITE LATER))
     (NAME DATE)
     (FRAME *SPECIAL-TIME)))
  (WHAT
    ((SPECIFIER THIS)
     (FRAME *MEETING)))
  (ATTITUDE *DESIRED)
  (WHO ((FRAME *I)))))
```

Example 2:

If we could schedule a meeting for two hours in the next two weeks then that would be good.

```
(((CONJUNCTION IF)
  (SENTENCE-TYPE *STATE)
  (FRAME *SCHEDULE)
  (HOW-LONG
    ((QUANTITY 2)
     (UNIT HOUR)
     (FRAME *LENGTH)))
  (WHEN
    ((SPECIFIER
       (*MULTIPLE*
         DEFINITE
         NEXT
         2
         PLURAL))
     (FRAME *SPECIAL-TIME)
     (NAME WEEK)))
```

```
  (WHAT
    ((SPECIFIER INDEFINITE)
     (FRAME *MEETING)))
  (ATTITUDE *POSSIBLE)
  (WHO ((FRAME *WE))))
 ((CONJUNCTION THEN)
  (FRAME *GOOD)
  (WHAT ((FRAME *THAT)))
  (SENTENCE-TYPE *STATE)))
```

Example 3:

Let me know what times would be good for you.

```
(((FRAME *GOOD)
  (FOR-WHOM
    ((FRAME *YOU)))
  (SENTENCE-TYPE *QUERY-REF)
  (WHAT
    ((SPECIFIER
       (*MULTIPLE* WHAT PLURAL))
     (WH +)
     (FRAME *TIME-SLOT)))))
```

Example 4:

But I'm not free for two hours on Monday the eighth.

```
(((CONJUNCTION BUT)
  (FRAME *FREE)
  (SENTENCE-TYPE *STATE)
  (HOW-LONG
    ((QUANTITY 2)
     (UNIT HOUR)
     (FRAME *LENGTH)))
```

```
  (WHEN
    ((FRAME *SIMPLE-TIME)
     (DAY 8)
     (DAY-OF-WEEK MONDAY)))
  (WHO ((FRAME *I)))
  (NEGATIVE +)))
```

Example 5:

However, on the ninth I'm free after twelve o'clock.

```
(((WHEN
    ((FRAME *INTERVAL)
     (START
       ((FRAME *SIMPLE-TIME) (HOUR 12)))
     (INCL-EXCL EXCLUSIVE)))
   (WHEN-0
     ((FRAME *SIMPLE-TIME) (DAY 9)))
   (FRAME *FREE)
   (SENTENCE-TYPE *STATE)
   (WHO ((FRAME *I)))))
```

Example 6:

I will be out of town until Thursday.

```
(((FRAME *OUT-OF-TOWN)
  (SENTENCE-TYPE *STATE)
  (WHEN
    ((END
       ((FRAME *SIMPLE-TIME)
        (DAY-OF-WEEK THURSDAY)))
     (INCL-EXCL INCLUSIVE)
     (FRAME *INTERVAL)))
  (WHO ((FRAME *I)))))
```

Example 7:

I am free on Friday the twelfth from eleven to one.

```
(((FRAME *FREE)
  (SENTENCE-TYPE *STATE)
  (WHEN
    ((FRAME *TIME-LIST)
     (ITEMS
     (*MULTIPLE*
        ((FRAME *SIMPLE-TIME)
         (DAY 12)
         (DAY-OF-WEEK FRIDAY))
        ((FRAME *INTERVAL)
         (END ((FRAME *SIMPLE-TIME) (HOUR 1)))
         (START ((FRAME *SIMPLE-TIME) (HOUR 11)))
         (INCL-EXCL INCLUSIVE))))
         (CONNECTIVE -)))
   (WHO ((FRAME *I)))))
```

Example 8:

Because I couldn't do it until Tuesday.

```
(((CONJUNCTION BECAUSE)
  (FRAME *DO)
  (SENTENCE-TYPE *STATE)
  (WHEN
    ((END
       ((FRAME *SIMPLE-TIME)
        (DAY-OF-WEEK TUESDAY)))
     (INCL-EXCL INCLUSIVE)
     (FRAME *INTERVAL)))
  (WHAT ((FRAME *IT)))
```

```
(ATTITUDE *IMPOSSIBLE)
(WHO ((FRAME *I)))))
```

References

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

Aho, A. and J. Ulman. 1972. *The Theory of Parsing, Translation, and Compiling Vol 1: Parsing.* Pretence-Hall, Englewood Cliffs, New Jersey.

Allen, J. 1995. *Natural Language Understanding.* The Benjamin/Cummings Publishing Company, Inc.

Allen, J. F. and L. K. Schubert. 1991. *The Trains Project.* Ph.D. thesis, University of Rochester, School of Computer Science. Technical Report 382.

Beale, S., S. Nirenburg, and K. Mahesh. 1996. Hunter-gatherer: Three search techniques integrated for natural language semantics. In *Proceedings of the 13th National Conference on Aritifical Intelligence (AAAI 96)*, Portland, Oregon.

Carbonell, J. G. and P. J. Hayes. 1984. Recovery strategies for parsing extragrammatical language. Technical Report 84-107, School of Computer Science, Carnegie Mellon University.

Carbonell, J. G. and M. Tomita. 1987. Knowledge-based machine translation, the cmu approach. In S. Nirenburg, editor, *Machine Translation: Theoretical and methodological issues.* Cambridge University Press.

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

Clark, H. H. and E. F. Schaefer. 1989. Contributing to discourse. *Cognitive Science*, 13:259–294.

Clark, H. H. and D. Wilkes-Gibbs. 1986. Referring as a collaborative process. *Cognition*, 22:1–39.

Danieli, M. and E. Gerbino. 1995. Metrics for evaluating dialogue strategies in a spoken language system. In *Working Notes of the AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation.*

Darwin, C. 1859. *On the Origin of Species by Means of Natural Selection.* John Murray.

Early, J. 1970. An efficient context-fre parsing algorithm. *Communications of the ACM*, 13(2).

Ehrlich, U. and G. Hanrieder. 1996. Robust speech parsing. In *Proceedings of the Eight European Summer School In Logic, Language and Information, Prague, Czech Republic.*

Federici, S., S. Montemagni, and V. Pirrelli. 1996. Shallow parsing and text chunking: a view on underspecification in syntax. In *Proceedings of the Eight European Summer School In Logic, Language and Information, Prague, Czech Republic.*

Fox, B. 1987. *Discourse Structure and Anaphora: Written and Conversational English.* Cambridge University Press.

Garfinkel, H. 1964. Studies in the routine grounds of everyday activities. In H. Garfinkel, editor, *Studies in Ethnomethodology*. Prentice Hall, Inc.

Gertner, A. N. and A. L. Gorin. 1993. Adaptive language acquisition for an airline information subsystem. In *Neural Networks for Speech and Vision Applications*. R. Mammime, ed.

Grosz, B. and C. Sidner. 1986. Attention, intentions, and the structure of discourse. *Journal of Computational Linguistics*, 12:175–204.

Haas, N. and G. G. Hendrix. 1983. Learning by being told: Acquiring knowledge for information management. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann Publishers, pages 405 – 421.

Hatch, E. 1983. Simplified input and second language acquisition. In R. Andersen, editor, *Pidginization and Creolization as Language Acquisition*. Newbury House Publishers.

Heeman, P. and J. Allen. 1994. Detecting and correcting speech repairs. In *The 32nd Meeting of the Association for Computational Linguistics*.

Hinkelman, E. A. 1990. *Linguistic and Pragmatic Constraints on Utterance Interpretation*. Ph.D. thesis, University of Rochester, School of Computer Science. Technical Report 288.

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

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

Holland, J. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

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

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

Jensen, K., G. Heidorn, L. Miller, and Y. Ravin. 1984. Parse fitting and prose fixing: Getting a hold of ill-formedness. *American Journal of Computational Linguistics*, 9(3-4):147–60.

Johnson, M. 1991. The computational complexity of glr parsing. In M. Tomita, editor, *Generalized LR Parsing*. Kluwer Academic Publishers.

Just, M. and P. Carpenter. 1987. *The Psychology of Reading and Language Comprehension*. Allyn and Baker, Inc.

Kay, M. 1980. Algorithm schemata and data structures in syntactic processing. Technical report, Xerox PARC, CSL-80-12.

Kipps, J. R. 1991. Glr parsing in time $o(n^3)$. In M. Tomita, editor, *Generalized LR Parsing.* Kluwer Academic Publishers.

Klavans, J. and P. Resnik. 1997. *The Balancing Act.* The MIT Press.

Koza, J. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press.

Koza, J. 1994. *Genetic Programming II.* MIT Press.

Lambert, L. 1993. *Recognizing Complex Discourse Acts: A Tripartite Plan-Based Model of Dialogue.* Ph.D. thesis, Department of Computer Science, University of Delaware.

Lambert, L. and S. Carberry. 1992. Modeling negotiation subdialogues. In *Proceedings of the ACL.*

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

Lavie, A., D. Gates, M. Gavalda, L. Mayfield, and A. Waibel L. Levin. 1996. Multi-lingual translation of spontaneously spoken language in a limited domain. In *Proceedings of COLING 96, Kopenhagen.*

Lavie, A. and M. Tomita. 1993. GLR* - an efficient noise-skipping parsing algorithm for context free grammars. In *Proceedings of the Third International Workshop on Parsing Technologies.*

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

Lehman, J. F. and J. G. Carbonell. 1989. Learning the user's language: A step towards automated creation of user models. In A. Kobsa and W. Wahster, editors, *User Models in Dialogue Systems.* Springer-Verlag.

Levin, L., O. Glickman, Y. Qu, D. Gates, A. Lavie, C. P. Rosé, C Van Ess-Dykema, and A. Waibel. 1995. Using context in machine translation of spoken language. In *Theoretical and Methodological Issues in Machine Translation.*

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

Mann, W.C̃. and S.Ã. Thompson. 1986. Relational propositions in discourse. Technical Report RR-83-115, Information Sciences Institute, Marina del Rey, CA.

Mayfield, L., M. Gavaldà, Y-H. Seo, B. Suhm, W. Ward, and A. Waibel. 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., M. Gavaldà, W. Ward, and A. Waible. 1995b. 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.*

Michalewicz, Z. 1994. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York.

Miller, L. G. and A. L. Gorin. 1992. Structured networks for adaptive language acquisition. In *The International Journal of Pattern Recognition and Artificial Intelligence.*

Pierce, R. and L. Beekman. 1985. Effects of linguistic and extralinguistic context on semantic and syntactic processing in aphasia. *Journal of Speech and Hearing Research*, 28:250–254.

Polanyi, L.˙ 1988. A formal model of the structure of discourse. *Journal of Pragmatics*, 12:601–638.

Polzin, T. S. 1994. Parsing spontaneous speech: A hybrid approach. In *Workshop on Combining Connectionist and Symbolic Processing, ECAI-94*, Amsterdam, The Netherlands.

Ramshaw, L. A. 1994. Correcting real-world spelling errors using a model of the problem-solving context. *Computational Intelligence*, 10(2).

Reithinger, N. and E. Maier. 1995. Utilizing statistical dialogue act processing in verbmobil. In *Proceedings of the ACL.*

Rosé, C. P. 1995. The structure of multiple-headed negotiations. Technical Report CMU-PHIL-65, Carnegie Mellon University.

Rosé, C. P. 1995b. Plan-based discourse processor for negotiation dialogues. Unpublished manuscript.

Rosé, C. P., B. Di Eugenio, L. S. Levin, and C. Van Ess-Dykema. 1995. Discourse processing of dialogues with multiple threads. In *Proceedings of the ACL.*

Rosé, C. P. and A. Waibel. 1994. Recovering from parser failures: A hybrid statistical/symbolic approach. In *Proceedings of The Balancing Act: Combining Symbolic and Statistical Approaches to Language workshop at the 32nd Annual Meeting of the ACL.*

Rosenbeck, J., L. LaPointe, and R. Wertz. 1989. *APHASIA: A Clinical Approach.* PRO-ED, Inc.

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

Scha, R. and L. Polanyi. 1988. An augmented context free grammar for discourse. In *Proceedings of the 12th International Conference on Computational Linguistics.*

Schiffrin, D. 1987. *Discourse Markers.* Cambridge University Press.

Slator, B., M. Anderson, and W. Conley. 1986. Pygmalion at the interface. *Communications of the ACM*, 29(7).

Smith, R. 1992. *A Computational Model of Expectation-Driven Mixed-Initiative Dialog Processing.* Ph.D. thesis, CS Dept., Duke University.

Smith, R. 1997. An evaluation of strategies for selective utterance verification for spoken natural language dialog. In *Proceedings of the Fifth Conference on Applied Natural Language Processing.*

Smith, R. W., D. R. Hipp, and A. W. Biermann. 1995. An architecture for voice dialogue systems based on prolog-style theorem proving. *Computational Linguistics*, 21(3):218–320.

Srinivas, B., C. Doran, B. Hockey, and A. Joshi. 1996. An approach to robust partial parsing and evaluation metrics. In *Proceedings of the Eight European Summer School In Logic, Language and Information, Prague, Czech Republic.*

Suhm, B., L. Levin, N. Coccaro, J. Carbonell, K. Horiguchi, R. Isotani, A. Lavie, L. Mayfield, C. P. Rosé, C. Van-Ess Dykema, and A. Waibel. 1994. Speech-language integration in a multi-lingual speech translation system. In *Proceedings of the AAAI Workshop on Integration of Natural Language and Speech Processing.*

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

Tomita, M. 1986b. Sentence disambiguation. *Computers and Translation*, 1.

Tomita, M. 1991. *Generalized LR Parsing.* Kluwer Academic Publishers.

Van Noord, G. 1996. Robust parsing with the head-corner parser. In *Proceedings of the Eight European Summer School In Logic, Language and Information, Prague, Czech Republic.*

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

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

Young, S. 1990. Use of dialogue, pragmatics and semantics to enhance speech recognition. *Speech Communication*, 9(5), December.

Young, S. 1993. Dialog structure and plan recognition in spontaneous spoken dialogue. Technical report, School of Computer Science, Carnegie Mellon University.

Young, S. and W. Ward. 1993. Semantic and pragmatically based re-recognition of spontaneous speech. In *Proceedings of the European Conference on Speech Communication and Technology.*