

**Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596**

**STRATEGIES IN LANGUAGE ACQUISITIONS:
LEARNING PHRASES FROM EXAMPLES IN CONTEXT**

U. Zernik

**January 1987
CSD-870099**

**Strategies in Language Acquisitions:
Learning Phrases from Examples in Context**

Uri Zernik

January 1987

Technical Report UCLA-AI-87-1

Forwarding Address: General Electric
Research & Development Center
Schenectady, N.Y. 12301

UNIVERSITY OF CALIFORNIA

Los Angeles

Strategies in Language Acquisition:

Learning Phrases in Context

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in Computer Science

by

Uri Zernik

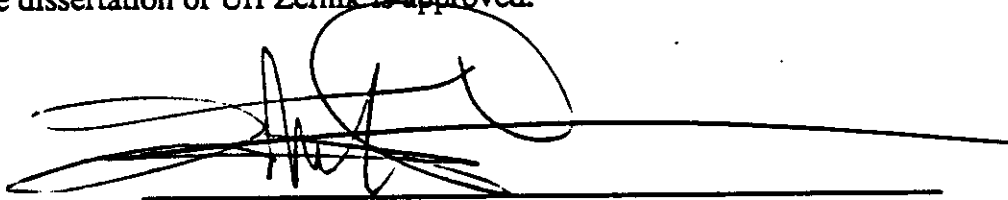
1987

© Copyright by

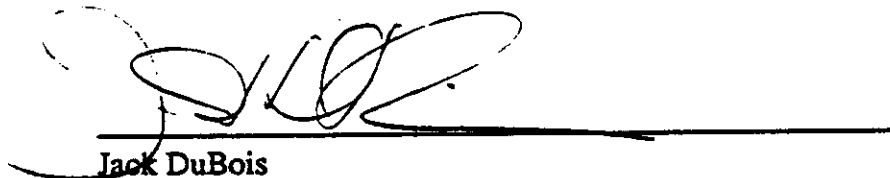
Uri Zernik

1987

The dissertation of Uri Zernik is approved.



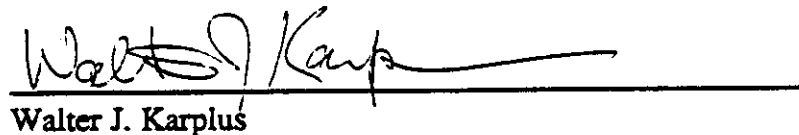
Edward L. Keenan



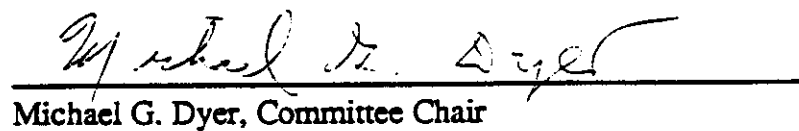
Jack DuBois



Judea Pearl



Walter J. Karplus



Michael G. Dyer, Committee Chair

University of California, Los Angeles

1987

To My Grandfather David

Table of Contents

	page
PREFACE	1
1. LEARNING LANGUAGE IN A COMPUTER PROGRAM	2
1.1 LEARNING A NEW PHRASE	3
1.2 ISSUES IN LANGUAGE ACQUISITION	4
1.3 THE APPROACH	6
1.4 THE COMPUTER PROGRAM <i>R/NA</i>	9
1.5 PREVIOUS COMPUTATIONAL MODELS	14
1.5.1 Language Processing	14
1.5.2 Language Learning	16
1.6 OUTLINE OF THE FOLLOWING CHAPTERS	18
PART I: DHPL: A Dynamic Hierarchical Phrasal Lexicon	20
2. ISSUES IN LEXICAL REPRESENTATION	21
2.1 INTRODUCTION	21
2.1.1 The Linguistic Behavior	23
2.1.2 Issues in Language Acquisition	25
2.2 ACCOUNTING FOR IDIOMACY IN THE LEXICON	27
2.2.1 Idioms as Equal Citizens	27
2.2.2 Productive vs. Non-Productive Phrases	28
2.2.3 Fixed vs. Variable Phrases	29
2.2.4 Overspecification and Underspecification	31
2.3 LEXICAL REPRESENTATION: PREVIOUS WORK	32
2.3.1 Lexical Presupposition	33
2.3.2 Language as a Knowledge-Based System	35
2.3.3 LFG and Language Acquisition	36
2.4 REPRESENTING THE CONTEXT	38
2.5 CONCLUSIONS	41
3. ORGANIZING THE LEXICON	42
3.1 THE LEXICON: CONTENTS AND STRUCTURE	43
3.1.1 Basic Phrases	43
3.1.2 The Global Structure	46
3.2 REPRESENTING THE INFINITIVE	47
3.2.1 Phrase Interaction	50
3.2.2 Parsing an Unknown	51
3.2.3 Overgeneralization and Recovery	52
3.2.4 Error Recovery	53
3.3 HANDLING WORD SENSES	53
3.3.1 Assigning Meanings to Particles	54
3.3.2 Resolving Word-Sense Ambiguity	54
3.3.3 Determining Level of Generality	55

3.3.4 Analyzing a New Production	58
3.3.5 Learning from Examples	58
3.4 INHERITING CASE ORDER	59
3.5 CONCLUSIONS	61
4. UNDERLYING KNOWLEDGE	63
4.1 INTRODUCTION	63
4.1.1 The Task Domain	64
4.1.2 The Issues	65
4.2 THE PROGRAM	67
4.2.1 Phrasal Parser	67
4.2.2 Phrasal Lexicon	68
4.2.3 Phrase Acquisition through Generalization and Refinement	69
4.3 CONCEPTUAL REPRESENTATION	69
4.3.1 Scripts	70
4.3.2 Specific Plans and Goals	70
4.3.3 Relationships	71
4.3.4 Abstract Planning Situations	73
4.3.5 Emotions and Attitudes	74
4.4 LEARNING PHRASE MEANINGS	75
4.5 CONCLUSIONS	76
 PART II: Learning Phrases in Context	 78
5. LEARNING A HIERARCHY OF PHRASES	79
5.1 INTRODUCTION	80
5.1.1 The Linguistic Phenomenon	80
5.1.2 The Issues	82
5.1.3 The Approach	83
5.2 REPRESENTING THE CONTEXT	85
5.3 THE LEXICON	87
5.4 THE ALGORITHM	89
5.4.1 Extracting a Phrase from a Single Example	90
(1) Forming the Pattern	91
(2) Forming the Meaning	92
5.4.2 Generalizing from Two Examples	92
5.4.3 Specializing a General Phrase	93
5.5 PREVIOUS WORK IN LANGUAGE ACQUISITION	93
5.6 CONCLUSIONS	96
6. LEARNING IDIOMS – WITH AND WITHOUT EXPLANATION	97
6.1 INTRODUCTION	98
6.1.1 The Passive-Voice Anomaly	98
6.1.2 The Program Behavior	100
6.1.3 Issues in Idiom Acquisition	101
6.2 THREE MACHINE-LEARNING PARADIGMS	101
6.3 KNOWLEDGE REPRESENTATION	103

6.4 THE ALGORITHM	105
6.5 THE PROCESS MODEL	106
6.6 LEARNING WITHOUT EXPLANATION	109
6.7 CONCLUSIONS	110
 PART III: Parsing for Learning	 112
7. PERFORMING IN THE PRESENCE OF INCOMPLETE LEXICAL KNOWLEDGE	113
7.1. INTRODUCTION	113
7.1.1 The Linguistic Behavior	114
7.1.2 The Issues	115
Overgeneralization:	116
7.1.3 The Approach: A Unified Parsing Mechanism	116
7.2 SEMANTIC REPRESENTATION OF ACTS	117
7.3 A HIERARCHICAL PHRASAL LEXICON	118
7.3.1 Single Phrasal Entries	119
7.3.2 Generalized Features	120
7.3.3 The Hierarchical Structure	121
7.4 PHRASE INTERACTION	122
7.4.1 Specific Interaction	123
7.4.2 General Interaction	124
7.5. GENERATION: A PROCESS MODEL	125
7.6 CONCLUSIONS	126
7.7 LIMITATIONS	127
 8. PARSING IS MONITORED	 128
8.1 INTRODUCTION	129
8.1.1 The Linguistic Phenomenon	129
8.1.2 The Theoretical Issues	130
8.1.3 Theoretical Approaches	132
8.1.4 The Program	133
8.2 THE PHRASAL LEXICON	134
8.2.1 A Verb Phrase	134
8.2.2 Modifiers	135
8.2.3 References	136
8.3 THE CONTEXT	136
8.4 PHRASE INTERACTION	137
8.4.1 Levels of Interpretation	137
8.4.2 Types of Interactions	138
8.5 DISCREPANCY DETECTION	141
8.6 ERROR ANALYSIS	142
8.7 CONCLUSIONS	143
8.8 LIMITATIONS	144
 9. BOOTSTRAPPING SYNTACTIC PATTERNS FROM SEMANTIC CONCEPTS	 145
9.1 INTRODUCTION	145
9.1.1 The Linguistic Phenomenon	146

9.1.2 The Issues	147
9.1.3 Lexical Representation	150
9.1.4 Phrase Acquisition	151
9.2 PARSING A COMPLEX CLAUSE	151
9.3 LEARNING A NEW INTERACTION	154
9.4 HANDLING AN OVERGENERALIZATION	156
9.5 CONCLUSIONS	157
PART IV: Design and Implementation	158
10. IMPLEMENTING A SELF-EXTENDING PARSER	159
10.1 INTRODUCTION	160
10.1.1 The Structure of the Program	160
10.1.2 The Functionality of the Program	162
10.1.3 Design Goals	163
10.2 CASE-LEVEL PARSING	164
10.2.1 Matching Syntactic Patterns	165
10.2.2 Constructing a Case Frame	165
10.2.3 Looking Up the Lexicon	166
10.2.4 Retrieving Referents	167
10.2.5 The Case-Frame-Stream	167
10.3 PHRASE-LEVEL PARSING	170
10.3.1 Triggering a Phrase	170
10.3.2 Unifying Case Frames	172
10.3.3 Inheriting Case Properties	173
10.3.4 Proving the Presupposition	175
10.3.5 Instantiating the Concept	176
10.3.6 Juxtaposing Phrases	178
10.4 CONCLUSIONS	181
11. STRATEGIES IN LEARNING	183
11.1 FIGURATIVE PHRASE ACQUISITION: A PROCESS MODEL	183
11.1.1 Literal Interpretation	184
11.1.2 Learning by Feature Extraction	185
11.1.3 Forming the Pattern	185
11.1.4 Forming the Concept	186
11.1.5 Phrase Generalization	187
11.2 THE STRATEGIES	188
11.2.1 Extracting a New Pattern	189
11.2.2 A Phrase-Modification Cycle	192
11.2.3 Modifying Class Specifications	196
11.2.4 Learning General Phrases	199
11.2.5 Extracting Phrase Concepts	202
11.3 CONCLUSIONS	205
12. SUMMARY, CONCLUSIONS, AND FUTURE WORK	206
12.1 CONCLUSIONS	206

12.2 STATUS OF IMPLEMENTATION	208
12.3 LIMITATIONS	208
12.4 IMPORTING ARTIFICIAL INTELLIGENCE METHODS INTO LINGUISTICS	210
12.5 THE LEARNING ALGORITHM	210
12.6 FUTURE RESEARCH	211
Was It Worth it?	215
References	216
A. PROGRAM TRACE	227
A.1 PARSING WITH A COMPLETE LEXICON	227
A.2 LEARNING A NEW PHRASE	239
B. MCRINA	257
B.1 Getting Started	258
B.2 Adding Lexical Entries	258
B.3 Sample Session	259
B.4 A Sample Lexicon	264
B.5 Single-Word Definitions	267
B.6 Verb Inflections	272
B.7 Interfacing with GATE	273
B.8 The Code	275
References	309

ACKNOWLEDGEMENTS

I wish to thank my adviser Michael Dyer who taught me to distinguish between the interesting and the mundane. Michael has single handedly created an environment, the Artificial Intelligence Laboratory at UCLA, which enabled us all to pursue research challenges, that otherwise would have remained only as dreams. Within the lab, I acknowledge Mike Gasser's impact on my work. It is interesting how two people can investigate the same phenomenon, have a complete agreement about it—and still come up with two different views. I acknowledge the constructive criticism I continuously received from the people in the lab: Sergio Alvarado, Charlie Dolan, Stephanie August, Ric Feieffer, Anna Gibbons, Seth Goldman, Jack Hodges, Erik Mueller, Mike Pazzani, Alex Quilici, Walter Read, John Reeves, Eve Schooler, Ron Sumida, and Scott Turner. I thank professors Evelyn Hatch and Margot Flowers, and my committee members Judea Pearl, Walter Karplus, Jack DuBois, and Ed Keenan, for helpful comments. In general, people who gave me a hard time about my research helped me the most.

I wish to thank my parents Rivka and Rafael. I don't want to underestimate the contribution by theirs and my grandmother's, Sara, in weekly nudging from Israel, "Noo, when are you going to finish your PhD?" Linda, my wife, and David, my son, helped me in ways that are hard to explain. Linda also read and rewrote parts of the dissertation, and she would be the one to tell me: "Well, this sen-

tence really supports your theory, but it does not make sense!" Yehuda Afek supported me through the years with his friendship.

My getting started in a new country was, and still is, a difficult task. Academically, I was always supported by Walter Karplus' advice. Personally, I was supported by my family, Henry and Miriam Picard, and Shlomo and Chana Givon. Hopefully, in the future, I will be instrumental in helping out other students who are fresh off the boat—whether in America or in Israel.

This work has been supported throughout the years by a grant from the Initial Teaching Alphabet (ITA) foundation. I thank in particular Betty Thompson, the executive director of the foundation, for her faith in our work, and for her personal interest. Finally, I acknowledge numerous second language speakers, who inadvertently contributed errors which stimulated this work.

ABSTRACT OF THE DISSERTATION

Strategies in Language Acquisition

Learning Phrases in Context

by

Uri Zernik

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1987

Michael G. Dyer, Chair

How is language acquired by people, and how can we make computers simulate language acquisition? Although current linguistic models have investigated extensively parsing and generation, so far, there has been no model of learning new lexical phrases from examples in context.

We have identified four issues in language acquisition. (a) How can a phrase be extracted from a single example? (b) How can phrases be refined as further examples are provided? (c) How can the context be incorporated as part of a new phrase? (d) How can acquired phrases be used in parsing and in generation?

In solving these problems, we have established three theoretical points. (a) We have shown how a dynamic lexicon is structured as a phrasal hierarchy. (b) We have constructed strategies for learning phrases. (c) We have constructed a parsing mechanism which can operate even in the presence of lexical gaps.

The program RINA has incorporated these elements in modeling a second-language speaker who augments her lexical knowledge by being exposed to examples in context.

PREFACE

Here is the basic artificial intelligence paradox: On the one hand, computer systems can solve problems which are very difficult for humans. One can sit at home and type a query regarding his income tax into a personal computer, and in spite of the tremendous complexity of the problem, the computer will come back in no time with the answer.

On the other hand, computers fare poorly on tasks which people perform every day. Understanding even simple sentences expressed in human language, a task we can perform without any mental effort, is considered a tough task for computers. In fact, the simpler the utterance is, the harder it is for a computer to analyze. How can a computer understand a simple word such as *this*? In the computer's notion, what is *this*? Clearly, talking, and communicating in general, is just a manifestation of deeper thinking processes. This is what makes linguistics research so difficult.

We have learned to live with the basic stupidity of computers as an inevitable evil. Chief among these evils is computers' inability to learn from experience. How many times have you, as a computer user, asked yourself: "Why can't some systems programmer write a piece of code so that I won't have to type in the same sequence of commands each time? If only I had the time, I would write this simple program that learns common sequences of commands by experience, but I can't right now, my hands are full with writing this proposal..." (and so it goes). Learning by incorporating experiences turns out to be one of the most exciting areas of research in artificial intelligence. For one thing, it is so useful, even in the simple tasks outlined above. For another, this mode of learning is the key for turning computers into smart machines. The ability to collapse individual episodes into applicable knowledge is a keystone for intelligent behavior—which again seems to be one of those very difficult tasks.

If both language and learning by experience are difficult, then why did I decide to take on the task which is their cross section, namely to investigate language acquisition? My answer is personal. As a second-language speaker I find myself constantly occupied with guessing what people mean when they say certain things. My adviser would introduce me to an audience by saying: *Uri, take it away!*, and I would be left wondering, "take *what* away?" For an American this is a "dead idiom", a known. But for me, figuring out such phrases is a very lively problem, in which I am consciously engaged. Consequently, I decided to investigate in depth the issues involved in language acquisition.

Chapter 1: Learning Language in a Computer Program

How can a computer program augment its own lexicon? Current language processing programs have assumed the existence of a fixed lexicon, which is given to the program at the outset. However, is this assumption realistic, or must programs acquire language dynamically? First, due to the huge size of the human vocabulary, especially when considering phrases and idioms, it is impossible to manually code the entire lexicon, and it is necessary to automate the encoding process. Second, human language is not a static entity. Since language is dynamically evolving, a complete, fixed lexicon cannot be supplied to a program at the outset. Consider the phrases *to debug a program*, *to reboot a node*, *to bring down an operating system*, *to kill a process*, in the world of computer jargon. Whether or not these are new phrases or they are new interpretations to existing phrases, for both human or a computer reader to be robust, these elements must be encoded dynamically as they are encountered.

Finally, there is also a methodological problem to consider: what other alternative exists for encoding *systematically* a large body of knowledge? How can each individual entry be justified as to its validity? Acquisition presents the method for encoding a lexicon in a principled way: a uniform set of learning strategies is applied throughout. Thus, the task is shifted from the justification of many individual entries to the justification of a relatively small number of learning strategies.

However, although many acquisition models [Reeker76, Anderson77, Bates82, Langley82, Berwick85, MacWhinney87] have learned grammar rules, which are relatively easy to encode manually, only a few language acquisition models [Granger77, Selfridge80] have aimed at the automatic augmentation of the lexicon itself. Why has this aspect been overlooked? If, indeed, this is a difficult task, what are the problems involved?

1.1 LEARNING A NEW PHRASE

Second language acquisition has presented the paradigm for our model. This learning mode was chosen—and not child learning acquisition—in order to factor out aspects of world-knowledge acquisition. Subsequently, world knowledge is assumed to be a given. Consider, for example, the following dialog between a second language speaker and a native speaker:

Native: Remember the story of David and Goliath?

David took on Goliath.

Learner: David took him somewhere?

Native: No. David took on Goliath.

Learner: David won the fight. He took on him?

Native: No. David took him on. he decided to fight.

Learner: David accepted the challenge. He took him on?

Native: Ok.

later on:

Native: I took on the hardest question in the exam.

Learner: You tried to solve a hard problem.

In this dialog the learner acquires a new English phrase, *to take on*, relative to the given context, the biblical story of David and Goliath. The learner is familiar with the single words *take* and *on*. However, he/she does not know the entire phrase *to take on*. Surprisingly—for a native speaker—the learner does not immediately zero in on the appropriate meaning for the phrase. Learning proceeds in three steps.

Literal interpretation: First, the learner passed over the new phrase altogether, assuming that *take* was used in its literal meaning: "to move an object from point A to

point B". In processing a new word combination, the learner incorrectly applied existing vocabulary. However, since this interpretation contradicts the learner's notion of the biblical story (David did not take Goliath anywhere in that story), the learner realized that this literal interpretation was inappropriate.

Hypothesis Formation: Second, when the native repeated his original sentence, the learner identified the existence of a new phrase and tried to form its meaning: "to take on means to win a fight". This incorrect guess is interesting: how did the learner come up with this hypothesis? Moreover, the learner also acquired incorrectly the syntax of the new phrase, thus producing: he took on him.

Hypothesis Correction: Finally, by the third input sentence, the learner figured out the appropriate meaning of the phrase: "he accepted a challenge". However, can the new phrase be used in understanding other examples? By confronting an example in a different domain, which involves a different kind of a challenge—solving a question rather than fighting an enemy, the learner demonstrated his/her ability to generalize the new phrase.

This process, in which a human learner (a) encounters a new linguistic concept, (b) acquires it through a process of hypothesis formation and error correction, and (c) generalizes the original concept, is the subject of this research.

1.2 ISSUES IN LANGUAGE ACQUISITION

We have identified three issues in language acquisition.

Learning from Examples: How can a learner extract a general linguistic concept from examples? Is it simply by detecting similarities among the set of given examples? Assume that the phrase *to take on* is given to the learner in three different episodes:

- (1) David took on his enemy.
- (2) Mary took on her elder brother.
- (3) I decided to take on the school establishment.

By extracting features shared by all the given episodes, the learner could have hypothesized that the phrase includes the words *take* and *on*, and that it depicts a

conflict between a person and a stronger agent. However, there are two differences between the behavior of this model and the behavior of a human learner.

- (a) This model might acquire spurious features, and it might hypothesize that *take on* must appear always in the past tense, whereas a person realizes that the past tense need not be taken as a mandatory feature.
- (b) This model required multiple examples. A person, as shown in the dialog above (Section 1.1), is able to extract a linguistic concept even from a single example.

The yet-unresolved issue is: how can the appropriate linguistic concepts be extracted from single examples?

Generalization and Specialization: Phrases in the lexicon must be further developed as additional examples are provided. Assume that the learner is given the following *sequence* of examples:

- (4) The Lakers took on the Celtics.
- (5) I finally took on the hardest question in the exam.
- (6) I took on a new job.
- (7) John took on the character of his professor.
- (8) We took on a new systems programmer.

These examples illustrate why learning is not a simple process of generalization. Indeed sentences (4) and (5) only extend the meaning acquired in the original dialog: by sentence (4) (the Lakers took on the Celtics) the characters, X and Y, are generalized from *persons* into *agents*. Similarly, by sentence (5) (to take on a hard question) the meaning itself is generalized from a *fight* into a *challenge*. However consider sentence (6) (to take on a new job) Is this a manifestation of the same phrase, or is it a specific *idiom*? Furthermore, in regard to sentence (7) (to take on a character), the entire concept constructed so far might diminish by trying to find a general sense that accounts also for this new case, since , at the conceptual level, there is nothing in common between sentences (4), (5), and (6) on the one hand, and (7) on the other hand. Similarly, sentence (8) (to take on a system's programmer) is a manifestation of a separate phrase altogether.

Thus, the dilemma at each new encounter is whether to generalize the existing phrase or whether to create a new phrase with a separate meaning.

Coping with Knowledge Gaps: In learning a new linguistic concept, the first problem a learner faces is processing the text in the presence of that unknown element. Here are two examples:

- (9) Goliath goggled David to fight him.
- (10) David took on Goliath.

In sentence (9) the learner is presented with a new word *goggled*, which should be acquired. However, how can the sentence be processed in the first place, in absence of knowledge of that word?

In contrast, in sentence (10) the learner is presented with a new combination of words *take on*. However, here since the single words *take* and *on* are known to the learner, how does the learner identify the existence of an unknown?

Thus, alternatively, the issues regard: (a) processing sentences which include unknown words, and (b) identifying the existence of an *unknown* when it is a combination of *known* words.

1.3 THE APPROACH

In addressing the issues above we pursued an approach which is defined by five aspects: (1) second language acquisition, (2) semantic knowledge representation, (3) lexical representation, (4) learning in a conceptual hierarchy, and (5) computer modeling.

Modeling Second Language Acquisition: Learning in general, and specifically learning linguistic concepts, is an ongoing process. Two groups in particular are extensively engaged in language learning: children learning native language and adults learning a second language [Richards74, Ulm75, Hatch83, Gasser85]. Adults, as opposed to children, may augment their linguistic knowledge while, to a large extent, maintaining otherwise unchanging world knowledge. Three aspects of second language acquisition are investigated in our research:

- (1) The various types of errors committed: acquiring incorrect concepts, acquiring incorrect linguistic patterns and performing incorrectly while having the correct linguistic knowledge.

- (2) The processes underlying these errors (observing errors is the only way to expose these processes).
- (3) Strategies for error-recovery based on failure-analysis.

The implications of this study, however, are not confined only to second language speakers. Rather, observing second language speakers may reveal general learning processes which are used more frequently by second language speakers.

Representing the Context: What is the representation of the context? For example, how can the events involved in the biblical story of David and Goliath be represented in a computer system? Following Schank [Schank77], we use a set of primitive acts (CD's) for representing events in the context. For example, the following sentence :

David threw a stone at Goliath.

is represented using the primitive act, *propel*, as shown in the following *slot-filler* notation:

act	propel
actor	david
object	stone
direction	goliath

However, primitive acts such as *propel*, *mtrans* (mentally transfer) and *ptrans* (physically transfer) can hardly depict complex notions such as the ones underlying that biblical story. Thus, higher level concepts, which pertain to the character's motivations—*why did David throw the stone at Goliath*—are given by a system of plans, goals and interpersonal relationships [Wilensky83, Dyer83]. There are two basic relations between the two characters in the story: (a) a goal conflict exists between D (David) and G (Goliath), and (b) the physical-power relation, by which G is superior to D. The goal conflict leads to other potential plans, one of which is to fight, and one possible move in a fight is to throw a stone. Thus, the low level act of propelling a stone is connected through this representation to higher-level motivations of the characters. This representation is used in two ways: (a) in parsing text, words are converted into such conceptual representation, and (b) in learning, this representation provides clues for meaning formation.

Regularity and Idiomaticity in Phrases: Idiomatic behavior of phrases is difficult to

capture in the lexicon. For example, read the next two sentences:

- (12) Peace was struck between Israel and Egypt.
The hatchet was buried.
- (13) Finally, death prevailed. The bucket was kicked.

As opposed to the second phrase *kick the bucket*, the first phrase *bury the hatchet* can take the passive voice and still maintain its figurative meaning. What is the reason for this difference, and how can it be predicted for idiomatic phrases in general? Fillmore, Kay and O'Connor [Fillmore87] address the problematic behavior of idiomatic phrases, classifying them into categories according to the knowledge required for the understanding of each idiom. Our contention is that the only way to predict phrase behavior is by modeling the learning process.

Conceptual Hierarchy: Humans perceive objects in conceptual hierarchies [Rosch78, Fahlman79]. This is best illustrated by an example from peoples's communication. Consider the question: what is Jack? The answer Jack is a cat is satisfactory, provided the listener knows that a cat is a mammal and a mammal is an animate. The listener need not be provided with more general facts about Jack (e.g., Jack has four legs and a tail), since such information can be accessed by *inheritance* from the general classes subsuming a cat. In fact, for a person who does not know that cats are mammals, an adequate description of Jack should be more extensive.

Hierarchical organization is essential in dynamic representation systems for three reasons:

- o **Economy:** Redundancy is avoided if shared features are not repeated in each instance, but are given by a single generality.
- o **Learnability:** As shown by Mitchell [Mitchell82], through a hierarchy, learning can be reduced to a search process. When one acquires a new zoological term, for example *feline*, one can traverse the conceptual hierarchy, by *generalizing* and *specializing*, until the appropriate location is found for *feline* in the hierarchy—above a number of specific species, and below the general *mammal*.
- o **Prediction:** Hierarchy accounts for *predictive power*, which allows learning

models to form intelligent hypotheses. When first observing a leopard and by assuming it is a feline, a learner, who has not been exposed to prior information about leopards, may hypothesize that this new animal feeds, breeds, and hunts in certain ways—based on his/her knowledge of *felines* in general.

While it is clear how hierarchy should be used in representing zoological concepts, it is not clear how it applies in representing linguistic concepts. Can linguistic systems too benefit from a hierarchical organization? We show how elements in a lexicon can be organized in a hierarchy and thus facilitate a dynamic linguistic behavior.

A Propositional Computer Program: What level of analysis is appropriate in modeling language acquisition? In contrast to *connectionistic* models, for example, in which a computer program basically organizes a set of connections, we suggest modeling language acquisition by a *propositional* computer program. In this program, all aspects of cognition are represented as symbolic propositions, and learning processes are simulated as logical interaction of rules. This level of analysis is appropriate for two reasons. First, such a program could be used as part of a larger natural language processing system. Acquired linguistic knowledge would be stored in the computer memory and could facilitate an interface between a knowledge-based system and natural language text. Furthermore, by tracing such a program, we could ask qualitative questions, and obtain answers about the learning process itself.

At this level of analysis we are concerned only with *external observed* behavior. No correspondence is claimed between *logical* structures in the program with *physiological* structures in a human. Rather, the purpose of the model is to satisfy Turing's criterion [Turing50] for intelligent behavior, by ultimately making the external behavior of the proposed program indistinguishable from expected human behavior.

1.4 THE COMPUTER PROGRAM RINA

Accordingly, we have constructed a computer program, called RINA, as a model of a second language speaker. The computer paradigm is simple:

- (1) All knowledge of language in the program resides in a data structure called a *lexicon*.
- (2) The program receives input sentences from a *user*. By *parsing*, using the lexicon, text is converted into a conceptual representation.

- (3) By *learning*, new phrases in the input text are added on to the lexicon.

The figure below illustrates schematically the operation of the program.

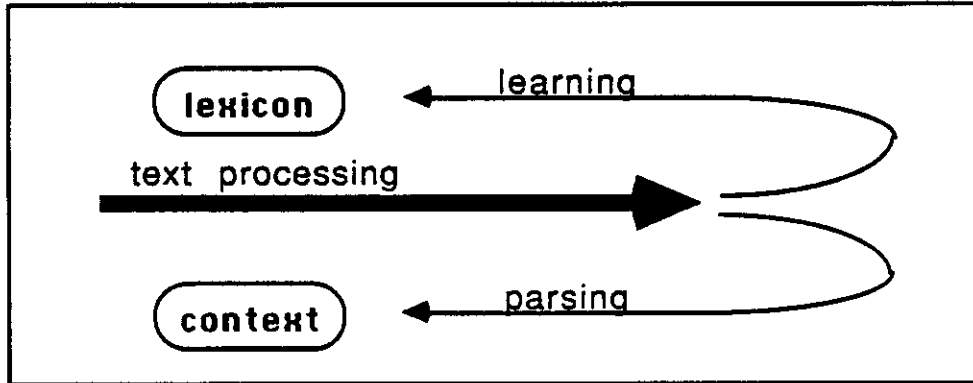


Figure 1.1: RINA: Basic Operational Scheme

Language processing, namely parsing as well as learning, is relative to (a) a given context, and (b) a given lexicon. *By parsing*, new concepts are added to the context and, *by learning* the lexicon itself is augmented with new phrases encountered in the text. At each stage, the program conveys back to the user the hypothesis regarding the input sentence. The three main components of the program are: lexicon, parser, and learner*.

A Lexicon of Phrases: The proposed Dynamic Hierarchical Phrasal Lexicon (DHPL), has three important features:

- (1) Lexical entries consist of *entire phrases*, and not of single words as in general dictionaries. This provision enables us to represent uniformly a large variety of phrases, including idioms.
- (2) The lexicon is organized in a *hierarchy* by generality, and not as a "flat" list of entries. This enables us to acquire phrases in a process of generalization and specialization.
- (3) *Lexical presuppositions* provide the semantic conditions for phrase selection, by incorporating elements of the context.

* Generation in RINA has not been developed. The sentences generated as output are produced not by a full-fledged generator, but by simple phrase instantiation.

The lexicon is described by (a) the form of a single entry, and (b) its global structure.

(a) **A Single Entry:** Consider, for example, the entry for take on:

phrase

pattern :	Person1 take on Person2
concept :	Person1 decide to fight Person2
presupposition :	Person2 is stronger than Person1

This entry is given as a *triple*: the pattern defines the syntax of the phrase; the *concept* is the meaning of the phrase; and the *presupposition* is the context in which the phrase may be applied.

(b) **The Global Structure:** The lexicon is structured as a hierarchy of phrases. All phrases in the hierarchy are given uniformly as triples, (as the one above)—at various levels of generality.

Learning in a Hierarchy: The hierarchy is instrumental in learning. For example, in encountering either a new word (e.g., John goggled Mary to come over), or a new word combination (e.g., John described it away in court), learning is not from scratch, but it is influenced by other existing words and phrases. The general learning step involves propagation of phrases in the hierarchy.

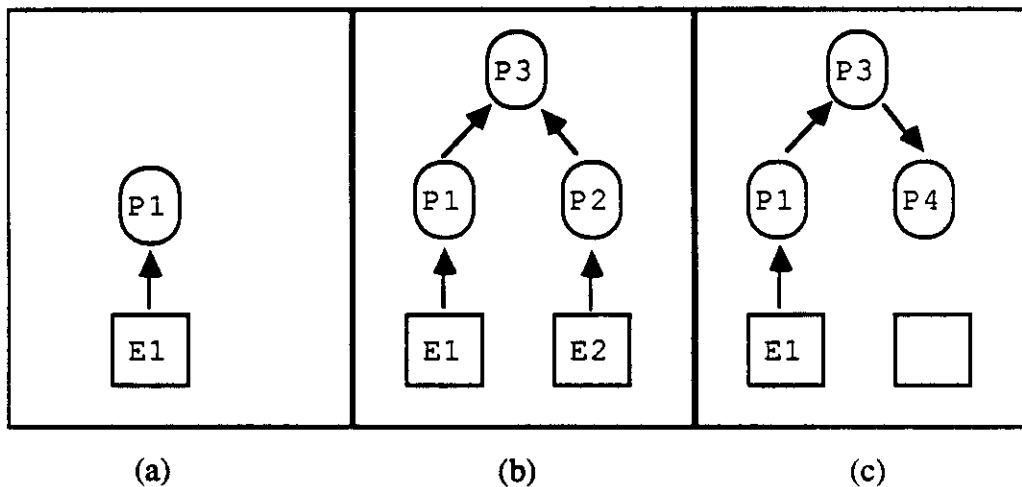


Figure 1.2: Propagating Phrases in a Hierarchy

This scheme, in which a *box* stands for an example, and an *oval* stands for a lexical phrase, illustrates how lexical phrases are propagated: (a) phrase P1 is extracted from

a single example E1, (b) phrase P3 is generalized from two existing phrases P1 and P2, and (c) phrase P4 is formed as a specialization of an existing generalized phrase P3. The last step, (c) presents the *predictive* power of the program: the meaning of a new phrase is hypothesized through existing *similar* phrases.

Step (a) in the scheme above is problematic, since it involves extraction of a phrase from a single example. The process in that case is given below:

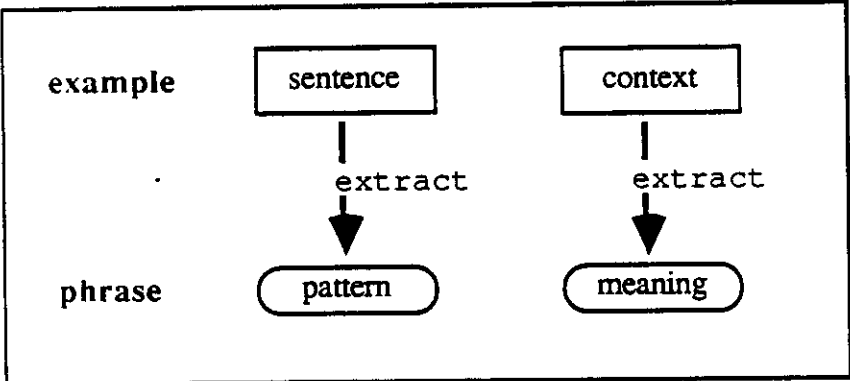


Figure 1.3: Learning from an Example

For example, consider learning the phrase *to take on*, as it is represented in Section (1) above. The pattern is extracted from the sentence *David took on Goliath*, and the concept is extracted from the context given by the biblical story. Since the context contains many aspects, there are heuristics for concept extraction. Acquisition is a process of hypothesis formation and error correction, by strategies associated with parsing discrepancies.

The purpose of learning in RINA is to augment the program's lexicon, so that the model's *performance* capabilities are improved.

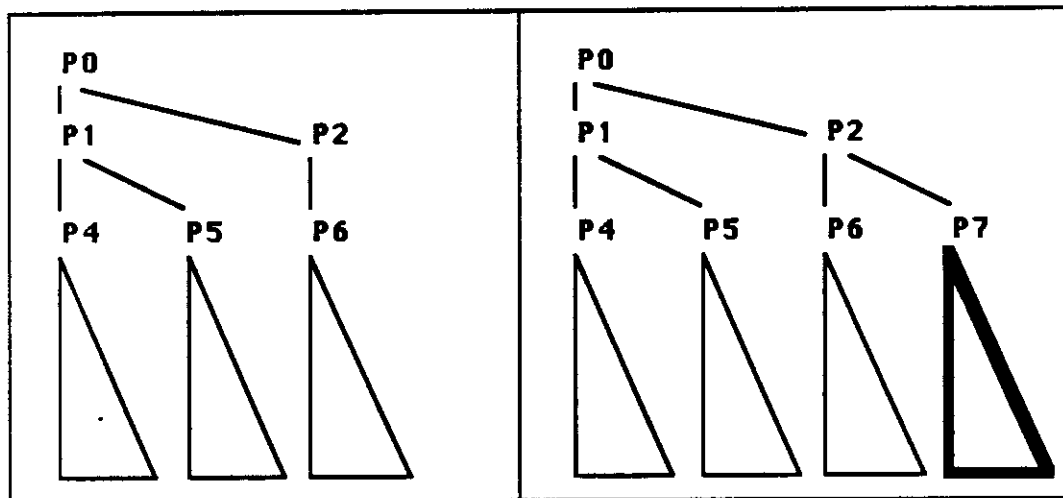


Figure 1.4: The Hierarchy: Before and After Learning

This figure illustrates how the lexicon is augmented by learning. An entire subtree (the marked triangle) is acquired in an incremental process. As a result, the model can parse *after* learning sentences it could not parse *before* learning.

Phrasal Parsing: The acquired lexicon is a functional unit which accounts for parsing text. RINA's parser introduces three features:

- (1) Parsing is relative to the context. Semantic disambiguation is supported by lexical presupposition.
- (2) The parser does not stall in the presence of an unknown. If a certain word or phrase is yet unknown, then a more general phrase is applied instead.
- (3) Parsing is monitored. Consequently, discrepancies in parsing are detected and are used to motivate learning.

These features have been introduced to support learning.

In conclusion, RINA has two functions: in the first place, RINA *performs* a task, namely, analyzing natural language text. Second, RINA is engaged in *learning*. By learning, the program improves its original performance capabilities.

1.5 PREVIOUS COMPUTATIONAL MODELS

In language processing, we have investigated (a) language *processing* programs, and (b) language *learning* programs.

1.5.1 Language Processing

In many applications, where it is advantageous to use natural language input, text must first be converted into an internal representation. It would be desirable to have a natural language front-end independent of any given task domain. However, language comprehension relies heavily on world knowledge, which varies according to the differing assumptions and underlying semantics of each task domain. Thus a diversity of language processing paradigms continue to exist, usually distinguished by the nature of the application:

In **database** applications, (e.g., LADDER and DCG [Hendrix77, Pereira80]) input sentences express queries and modifications to the database. A typical input query might be: How many ships are currently loading which weigh above 3000 tons? For such retrieval tasks, natural language queries are often parsed into logical expressions, where the major semantic elements consist of quantifiers, variables, and predicates over these variables.

In **knowledge-based** systems (e.g., MYCIN [Shortliffe76]), the task is to find the cause of a situation, or to project a consequence, such as: what could be the worst result of prescribing penicillin to my patient? Answering such questions requires identifying and activating cause/effect relations, often represented in the form of if/then rules.

A **planner-adviser** such as UC [Wilensky84] is intended to support the user by suggesting plans. It must figure out the goals and intended plans of the user. A typical input would be: I need more space. How do I remove some files? where the program must dynamically update the goal-plan situation from the input. Here, understanding the text may require maintaining a model of the user's current knowledge state and using this to guide comprehension.

Information-gathering and analysis systems (e.g., BORIS [Dyer83]) are required

to read documents and relate the facts into an intelligible picture. Such systems require *deep-analysis* of the input, where conceptual representations of goals, affects, beliefs, plans, justifications, reasoning, and abstract themes must interact. For instance, affect analysis is essential in understanding: The third launching attempt was rather *frustrating* since the weather changed again. Here we are not told that the attempt was aborted. This is inferred from knowledge of the affect *frustrating* combined with knowledge that good weather enables launches.

In all these applications, input text is converted into an internal representation geared to the semantics of the task domain. Parsing by a program is a difficult task, which is contrasted by the apparent ease with which people communicate in natural language. In designing algorithms for language processing, we discover the complexity of the cognitive tasks involved. Tasks in language processing can be ordered by increasing complexity:

Disambiguation: Ambiguity can appear at all levels of speech, and it must be resolved in the parsing process. Current language processing programs have focussed on disambiguation both at the syntactic [Marcus80] and at the semantic [Dyer83, Hirst86, Wilensky80] level. However, all these programs have assumed (a) *well-formed* input, and (b) *complete* lexical knowledge.

Lenient parsing accounts for ill-formed input. Input text might not always abide by text-book grammar, for three reasons: (a) a noisy channel might cause problems in reception, (b) operational conditions might cause people to generate terse sentences in which non-content words are omitted, and (c) in colloquial speech people tend to use sentences which are not well formed. Programs such as [Carbonell84, Granger83] are designed, by *lenient parsing*, to cope with ill-formed text. However, while these programs can cope with gaps in lexical knowledge they cannot construct new knowledge to fill in these gaps.

In **acquisition**, the program's knowledge is augmented. In contrast to *lenient parsing*, where the objective is to parse *in spite of* lexical unknowns, in *acquisition*, the objective is to identify discrepancies and to process them appropriately. So far there has been no general model for lexical acquisition by parsing.

In order to communicate effectively in natural language, a parser must first address

the ambiguity problem. Moreover, to appear robust and to face new situations, parsers must learn new lexical items from experience.

1.5.2 Language Learning

Past work in language learning emphasized either (a) learning of linguistic patterns or (b) learning of conceptual representations. There are three models for learning *linguistic patterns*:

General Problem Solving: PST [Reeker76] operated by GPS principles [Newell57] and similarly used a table of *difference-action* pairs. PST learned grammar by acting upon differences between the input sentence and an internally generated sentence. Six types of differences were classified and the detection of a difference which belonged to a class caused the associated alteration of the grammar. Although in our model we address learning of entire lexical entries (and not only syntactic patterns), we also use similar *discrepancy-driven* strategies.

General Production Systems: LAS [Anderson77] learned ATNs (Augmented Transition Networks) [Woods70] from sample sentence/meaning pairs. LAS presented one element in a larger cognitive model which accounted for general human inference and memory access [Anderson84].

Langley [Langley82] pursued Anderson's effort in his program AMBER which learned use of basic function words. The learning process was directed by mismatches between input sentences and sentences generated by the program. Learning involved recovery from both *errors of omission* (omitting function words such as *is* and *the* in *daddy bouncing ball*) and *errors of commission* (producing *daddy is liking dinner*). Like LAS, AMBER's main thrust was to apply general learning principles in language learning. These two projects were intended to demonstrate that language learning could be modeled using general production-system principles. In our model the generality of the learning algorithm is obtained by pursuing the knowledge-based approach [Wilensky81, Kay79].

Deterministic Syntactic Parsing: Berwick's program [Berwick85] performed in the context of PARSIFAL, a deterministic syntactic parser [Marcus80]. Berwick's program demonstrated how parsing rules can be acquired by actually applying the pars-

ing framework itself. In contrast to our approach, where learning is relative to the context, in Berwick's approach, the semantic context did not present a legitimate clue in the learning process, as stated by the following "psychological plausibility criterion" which dictated ([Berwick85] Chapter 1, page 41):

Minimal use of extrasyntactic information. Nearly every grammatical theory posits something like thematic role information. The acquisition procedure uses this information, but only for simple sentences, and only to fix a few syntactic parameters early in acquisition.

In other words, Berwick's program not only acquires language by ignoring the semantic context, Berwick asserts that using the context in language acquisition in general is not valid psychologically.

On the other hand, two computational models emphasized *learning semantic representations*:

Script Application: FOUL-UP [Granger77] learned meanings of single unknown words from context. The meaning was extracted from the script [Schank77] which provided the context. A typical learning situation was The car was driving on Hwy 66, when it swiveled off the road. The unknown verb was guessed from the *\$accident* script. FOUL-UP introduced three important elements which have been applied also in our model: (1) Learning was invoked by parsing failures. However, there was only one possible failure—the absence of a word in the lexicon—thus failure analysis was not required. (2) Word meanings were figured out from currently active scripts. (3) Linguistic clues, such as preposition senses, took part in forming meanings.

Minimalistic Parsing: CHILD [Selfridge80] modeled a one-year old child learning native language. At that age, concepts rather than language word-order conventions account for comprehension. A sentence such as Joshua, put the ball in the box is understood from the conceptual relationships and conceptual clues. Thus CHILD was able to learn basic word meanings starting only with a minimal linguistic knowledge. CHILD introduced heuristics to identify the unknown word in a sentence and to identify the intended concept in a context. Learning was accomplished by associating the new word with that concept.

The integration of these two approaches (learning syntax and learning semantics) is at the focus of our approach, in which:

- The model learns both syntactic patterns and semantic concepts. In fact, learning only one of these aspects in isolation is analogous to learning only one side of a mathematical equation.
- Semantic concepts are acquired by using linguistic clues, such as meanings of single words.
- Syntactic patterns themselves are acquired by using semantic clues which are derived from the representation of the context.

Thus learning syntax and learning semantics cannot be viewed as two isolated processes.

1.6 OUTLINE OF THE FOLLOWING CHAPTERS

Chapter 2 describes the issues in the design of a self-extending lexicon. Why can current linguistic theories not account for learning?

Chapter 3 presents a Dynamic Hierarchical Phrasal Lexicon (DHPL), which facilitates both language analysis and language acquisition.

Chapter 4 explains how figurative phrases can be represented using semantic structures such as scripts, goals, relations and emotions.

Chapter 5 presents the learning *scheme*: how phrases are acquired by propagation in a hierarchy. General as well as specific nodes in the hierarchy must be acquired from input which is restricted to specific examples.

Chapter 6 focuses on the basic learning step: (a) how a phrase can be extracted from a single example, and (b) what the significance is of the metaphor in the learning process.

Chapter 7 describes how parsing is performed in spite of missing lexical knowledge, and how application of general knowledge accounts for performance errors.

Chapter 8 turns to the basic parsing mechanism. In order to facilitate learning, parsing itself must be monitored, and discrepancies must be detected.

Chapter 9 introduces the duality between syntax and semantics as it is manifested in learning and in parsing. Syntactic derivation is presented as a short cut for a lengthy semantic derivation.

Chapter 10 details the implementation of the program RINA itself.

Chapter 11 lists the discrepancy-driven strategies applied in the learning schemes above.

Chapter 12 presents an evaluation of the project, and ideas for future work.

Appendix A describes sessions with the program RINA.

Appendix B introduces a "micro" version of RINA, called MCRINA, which can be used as a basic phrasal parser.

Part I

DHPL: A Dynamic Hierarchical Phrasal Lexicon

Chapter 2:

Issues in Lexical Representation

Part I of the dissertation focuses on the lexicon. First, in Chapter 2 we explain the issues involved in designing a lexicon which is inherently dynamic. We identify unresolved problems with existing, contemporary linguistic systems. Second, in Chapter 3 we present our design of a Dynamic Hierarchical Phrasal Lexicon (DHPL). We show how Finally, in Chapter 4, we describe how we handle the semantic aspect of lexical representation.

2.1 INTRODUCTION

Examination of the language acquisition task sheds light on the nature of the lexicon, illuminating issues which have been ignored by existing linguistic systems [Wilks75, Kay79, Bresnan82a, Gazdar85]. Current systems restrict their account to analysis and generation of text, by making the assumption that a fixed, complete lexicon exists at the outset. However, computational linguistic models are required to learn lexical items in context, the way people learn new words and phrases.

Learning commonly occurs when the learner detects a gap in his or her knowledge. In analysis, such a discrepancy can be detected when a new word or phrase is encountered. Learning involves three issues: (a) detecting the discrepancy in the first place, (b) forming an initial hypothesis about the new phrase, and (c) refining and generalizing this hypothesis through a process of error correction [Granger77, Langley82, Selfridge82, Zernik85a]. These three issues impose new requirements on the lexicon, regarding (a) its contents—the way individual entries are encoded, and (b) its

structure—the way entries are organized.

The need to detect discrepancies affects the **contents** of the lexicon. Both semantic and syntactic discrepancies must be detected, and correction strategies must be associated with various types of errors. Thus, lexical entries should not be underspecified, lest they will allow discrepancies to slip by unnoticed.

The need to generalize affects the **structure** of the lexicon. In order to make an initial hypothesis about a new element, it is important to glean from the text as much information as possible. This requirement is problematic: the text cannot be analyzed since an element is unknown; but on the other hand, for the element to be acquired, the text must be analyzed. The solution for this bootstrapping problem is to employ a lexical hierarchy by generality. When a specific pattern does not exist for a precise matching against the new element, one can apply a more general pattern, which albeit being less informative, does match the new element.

Thus, we propose employing a Dynamic Hierarchical Phrasal Lexicon (DHPL) which has three features: (a) lexical entries are given as entire phrases and not as single words, (b) phrases are organized in a hierarchy by generality, and (c) there is not separate grammar; grammar is encoded in general lexical phrases. The program RINA [Zernik86a] employs DHPL in modeling language acquisition. In particular, the program models second language acquisition of English phrases and idioms. The linguistic concepts being acquired are complex enough, so that neither a human learner, nor a computer program can acquire their complete behavior through a single example. Thus the initial hypothesis might be incorrect. Capturing incorrect hypotheses generated by humans, and simulating them by the computer program is essential for practical and theoretical reasons. First, the human user of the program will relate to the human-like errors generated by the program. Consequently he may present the program with constructive counterexamples. Second, human errors, such as errors of overgeneralization, reveal otherwise inaccessible cognitive processes and internal structures. Thus, errors made by human learners play a central role in constructing a cognitive model of acquisition. Subsequently, observed human behavior is analyzed in terms of the its corresponding computer program RINA.

2.1.1 The Linguistic Behavior

RINA receives examples from a user who teaches her new phrases. When RINA encounters a new phrase, she creates a hypothesis about its behavior, and accordingly she generates an example to demonstrate her state of knowledge. Communication between the program and the user is only through a sequence of examples – there is no way to discuss syntax and semantics explicitly.

Learning New Phrases: In the following dialog, RINA encounters an unknown phrase, *throw the book at somebody*.

User: Al Capone went on trial. The judge threw the book at him.
RINA: He threw a book at him?
User: No. The judge threw the book at him.
RINA: He punished him severely?

RINA is familiar with the single words *throw* and *book*. However, the entire figurative phrase is not in her lexicon. RINA first attempts a literal interpretation using a phrase existing in the lexicon (*throw an object*). When this interpretation fails she realizes the existence of an unknown, and tries to form the meaning of the new phrase by using (a) the context, and (b) the single words in the phrase.

Processing an Unknown: In the next dialog, RINA encounters a new word, *goggled*.

User: Jenny goggled John to come over.
RINA: Jenny told John that he must/can/will come to her.

RINA manages to extract useful information from the sentence in spite of the missing element. In particular, RINA's hypothesis includes three points:

- (a) Jenny's unknown act is a kind of *mtrans** (an act of communication).
- (b) The actor of the communicated event (coming over) is John.

* Conceptual classes such as *mtrans*, *mbuild*, *select-plan*, are based on semantic representation. Several of these elements are taken from Schank's [Schank77] system of primitive acts, goals and plans. *Mtrans* for example represents the transfer of mental information, and *mbuild* represents the construction of new information in memory. The particular scheme chosen is not so important as the fact that syntactic classes (such as verbs) are organized phrasally in terms of conceptual categories.

- (c) The communicated event is a conditional plan for the future (in contrast, for example, to the case: Jenny goggled John that he came over).

Normally, properties of an embedded phrase (e.g., to come over) are determined by the definition of the embedding verb. RINA manages to make an initial hypothesis even though the embedding verb (goggle) is unknown, by using generalized knowledge of phrase interaction. (The structure of a sentence Person1 goggled Person2 to do Act3 implies *mtrans* such as ask, tell and instruct, in contrast to Person1 goggled to do Act2 which implies an *mbuild*, such as decide). The hypothesis must be abstract, since RINA cannot determine at this point whether this *mtrans* act comes in the sense of allow (can come over), or instruct (must come over). Yet, even this hypothesis may turn out to be incorrect. For example, goggle could mean seduce, or influence in some other way. In either, it is important to come up with a hypothesis which provides a basis for further modification.

Resolving an Ambiguity: As with human listeners, computer parsers must also be able to interpret text successfully only when supplied the appropriate context [Zernik86b]. Consider the following sentence:

User: She took it up with her dad.

RINA: ?

Imagine a person hearing a fragment of a conversation between two unknown people, or alternatively, a computer program being given this sentence in isolation. Clearly, in the absence of a context, this sentence does not make complete sense. The pronouns, she, and it cannot be resolved in absence of referents which have been introduced in the discourse. In addition, the same phrase will mean different things in different contexts. Consider these two examples.

User: Jenny wanted to buy a new car.

She took it up with her dad.

RINA: She discussed the issue with her dad.

User: Jenny started jogging.

She took it up with her dad.

RINA: She started an activity with him.

Since the same sentence can be interpreted in two ways in two different contexts, a

question is raised regarding disambiguation. What is the impact of the context on phrase selection?

2.1.2 Issues in Language Acquisition

Three lexical representation issues must be addressed in modeling language acquisition.

Using Generalizations: As shown in the sentence below,

Jenny goggled John to come over.

the system must cope with unknown elements. Parts of the text must be examined to some extent, in spite of the presence of the unknown. Ideally, each element in the text is matched by a lexical phrase. Since no such phrase exists for a precise matching of the unknown element, a generalized phrase must be used to recover at least partial information. However, by the nature of generalization, the more generalized the matching phrase, the less informative it is.

Typical errors of overgeneralization were generated in a version of this paper by the first author, who is a second language speaker:

- o The third phrase requires to generalize the initial notion. (Section 3.3.1)
- o Wilensky suggested to represent knowledge as a database of rules. (Section 2.3.2)

In both cases, the learner applied the wrong generalized phrase, which accounts for verbs such as `decide` and `plan` (John decided to go home). This behavior does not capture verbs such as `suggest`, `require` or `tell` (John told to go sounds incorrect). The speaker faced a generation task in presence of incomplete lexical knowledge about `suggest` and `require`, and he resorted to using generalized knowledge. Using such knowledge, an idea could be communicated, albeit grammatically incorrectly.

Therefore, the lexicon must maintain phrases at various level of generality, to cope with different degrees of partial knowledge.

Using Linguistic Clues: Meaning representation is extracted from the context. For example, given the text below,

Al Capone went on trial.
The judge threw the book at him.

RINA guessed that throw the book at somebody means to punish that person severely. However, the context might consist of many concepts, some appropriate and some inappropriate (e.g.: did the judge acquit Al or did he punish him?). Thus, a basic task is feature extraction. In extracting features, the system must utilize clues provided by single words. For example, what is the significance of the particle at? How does it contribute to the construction of the meaning? An experiment with second language speakers reveals, predictably, that using a different preposition leads to a different learning result. When the given text is:

Al Capone went on trial.
The judge threw the book to him.

language learners formed the hypothesis that the judge actually acquitted the defendant. Thus, the lexicon must maintain senses for single words such as at and to that could be used as linguistic clues in feature extraction.

Using Semantic Clues: The system must hypothesize the scope and variability of the new phrases. Which one of the phrases below best captures the new phrase: the judge threw the book at him?

He threw **something** at him.
He threw **a book** at him.
He threw **the book** at him.

Each one of these patterns could be the specification of the new phrase. In determining degree of specificity the system must consult semantic clues extracted during parsing. For example, since no actual book exists in the context, then the reference the book is assumed to be a fixed literal. In contrast, consider the context below:

The judge was holding the third volume of tax law.
He threw the book at Al.

In this context, an instance of a book is found in the context (i.e., the third volume), and a different hypothesis is made about the the generality of the new pattern. Thus,

semantic discrepancies in parsing must be utilized in determining both scope and generality of syntactic patterns.

2.2 ACCOUNTING FOR IDIOMACITY IN THE LEXICON

What are the contents of the lexicon to be acquired? Traditionally, the lexicon has been viewed as a list of words, specifying syntactic and semantic properties for each entry. However, since in our theory, the lexicon provides the sole linguistic database, it must include a variety of linguistic knowledge types, not just properties of single words. Here the lexicon is extended in two ways: towards the specific by bringing in idioms, and towards the general by including grammar also.

2.2.1 Idioms as Equal Citizens

Are idioms such as *throw the book at a class apart*, to be distinguished from "normal" phrases which abide by grammar rules? The first to proclaim "equal rights" for idioms was Becker [Becker75], who called for a systematic treatment for the variety of phrases in the language. Consider these phrases:

We will be looking forward to seeing you guys.
He is cheap. He will not spend \$5, let alone \$8.
So much for superficial solutions.
Productive as well as non-productive phrases
should reside in the lexicon.

These phrases defy traditional text-book grammar analysis, however, they possess their own grammar. For example, it sounds odd to say *he is cheap. He will not spend 8\$, let alone \$5* [Fillmore87]. (Is the behavior of *as well as* analogous to the behavior of *let alone*?) Such linguistic phenomena cannot be ignored merely by tagging it as idiomatic, since idioms turn out to be ubiquitous in people's speech. Hardly can a sentence be found which behaves according to textbook grammar. There is a need therefore for a systematic treatment of idiosyncrasy [Fillmore87]. Furthermore, linguistic knowledge cannot be strictly divided into grammar rules and lexical items. Rather, there is an entire range of items: some very specific, in the sense that they pertain to a small number of instances, and some very general, pertaining to a large number of instances. The former have been called "lexical items", and the latter "grammar rules". However, it is not possible to define a clear borderline between such two distinct groups, as elements could be found at all levels of generality, not just at

the two ends of the spectrum. On one end, the phrase *it is raining cats and dogs* is very idiomatic. On other end, the phrase in *John took the spoon from Mary* is an instance of a general verb, *to take*, which may appear in many other ways. However, consider the phrase *John took the issue up with his dad*. Is this an idiom, or is it just an instance of the general verb *to take*?

2.2.2 Productive vs. Non-Productive Phrases

In the *phrasal approach* [Wilensky84] rather than maintaining lexical entries for single words, the lexicon maintains entire phrases. For example, the lexicon will contain many phrases involving the word *throw*. Consider these phrases as they appear in the following sentences.

- (1) He threw her off by a single inaccurate clue.
- (2) He threw a wild party for her graduation.
- (3) He threw up his whole breakfast.
- (4) He threw his weight around.
- (5) He threw a temper tantrum.
- (6) He threw a stone at the kitchen window.
- (7) He threw out that old chapter of his dissertation.
- (8) He threw out the garbage.
- (9) He threw the banana peel away.
- (10) He threw in the towel.
- (11) He threw the book at his students.
- (12) He threw it. His answer was totally incorrect.

To a certain extent, all the phrases above derive their meanings from the meaning of the verb *to throw*. However, the issue here is whether a single generic lexical entry for *throw* can suffice to produce the meanings of all those sentences. In example (6) (*he threw a stone*), the phrase for *throw* is used in its generic form and meaning: *to throw a physical object* means to propel that object through the air. Sentence (9) (*he threw away a banana peel*) too can be interpreted using the generic phrase. In sentence (8) (*he threw out the garbage*), on the other hand, the derivation of the meaning using the generic phrase is less direct, as it requires analysis at the level of plans and goals. Throwing an object *causes* the object to become inaccessible. Thus throwing out the garbage does not necessarily mean throwing it in the air as much as getting rid of it.

The meanings of the other sentences are even more detached from the generic meaning. The meaning of *throw the book at* is not a mere composition of the meanings of the single words, but requires extraneous knowledge from the trial situations. Neither a person, nor a computer program can produce the meaning of the phrase if the context is not given. Sentence (4) (*he threw his weight around*) introduces a metaphor [Lakoff80] in which a person's authority is compared to a weight, being used in a careless way. Sentence (2) (*he threw a party*) as well as sentence (5) (*he threw a temper tantrum*), use a different meaning of *throw* (to throw an event) which can hardly be related to its original meaning. Finally, sentence (12) (*he threw it*) represents a novel, yet still understandable, use of the word *throw* (as in *he blew it*).

Non-productive phrases are those in which the meaning of the entire phrase cannot be produced from the meanings of its constituents. Such phrases should be maintained in the lexicon as distinct entries. In fact, even *productive* phrases, such as *to throw out the garbage*, should be maintained as distinct entries. Even if the meaning can be produced each time from the single words, an objective of an *efficient* system is to compile knowledge whenever possible, and to minimize unnecessary derivations. Thus, phrases in the lexicon can be viewed as *linguistic episodes* indexed and compiled for further use. Such knowledge is redundant in regard to language parsing (the meaning could be derived from the constituents again and again). However, this is not the case in language generation, where unless the phrase is stored, it is unlikely to be generated again by the system. Thus, both productive and non-productive phrases must be stored in the lexicon.

2.2.3 Fixed vs. Variable Phrases

As another example of lexical phrases, consider phrases involving the word *at*:

- (13) John left school at noon.
- (14) He actually stayed at school for an hour.
- (15) He dabbled at the piano for a while.
- (16) John aimed the ball at Mary.
- (17) The criminal is still at large.
- (18) Mary did not feel at ease in the presence of her dad.
- (19) This is what I am trying to get at.
- (20) Did you understand anything at all?
- (21) Please come at once!

- (22) John looked at Mary.
- (23) Fred lives at New-York. (produced by a second language speaker.)

Certain phrases are *fixed*, in the sense that they do not take any variation. For example, at large, at all, or at once are such fixed phrases. One cannot say, for example, at twice. However, other phrases might be mutated and still maintain their basic meaning. For example, at noon, at midnight, at the hour, etc. convey a meaning of sharp timing. Another meaning shared among a set of phrases is described by the following sentences:

- (15) He dabbled at the piano for a while.
- (24) He nibbled at the corn.
- (25) He is playing at AI programming.

The use of the proposition at here implies an aimless, unfocused activity (This is the difference between playing the piano and playing at the piano). Similarly, the set of sentences:

- (22) John looked at Mary.
- (26) Spot sniffed at Mary.
- (27) Mary glanced at John.

share the implication that the sensory act was directed at the object.

Which ones of these phrases should be maintained in the lexicon? Fixed, idiosyncratic phrases such as at large, at once, and at all must be maintained in the lexicon. Otherwise they cannot be predicted by the system. However, the dilemma arises regarding variable phrases, such as in (22), (26) and (27). The question is whether to maintain all instances of a certain variable phrase or to maintain a single generalized entry which encompasses them all. We argue that *both* must be maintained. Specific phrases must be maintained as compiled, easy to access knowledge, while general phrases, which can derive many specific phrases, must be maintained too so that the system has a *predictive* power. Using such generalized phrases, the system can handle instances which have not been previously encountered.

In fact, specific "canned" phrases could not account for the following generation task, concerning the selection of appropriate prepositions in the following sentences:

- (28) {in on at} our school, there is one teacher I really like.

(29) I stayed late {in on at} school.

Notice that since both sentences involve the word *school*, it could not be used as a discriminator. Unless the lexicon maintains general predicates for the use of *in*, *at*, and *on*, the generator cannot select the appropriate preposition in each case. Clearly, it is difficult to capture the intuition of a native speaker in forming the general senses of these prepositions. An approximation of this intuition can be captured by modeling a second-language speaker who might "incorrectly" generate a sentence such as (23) above:

(23) Fred lives at New York.

Although it does not sound right to an English speaker, this sentence reflects the notion of that particular speaker.

2.2.4 Overspecification and Underspecification

Lexical entries should not be either underspecified or overspecified. Unless the lexical phrases are fully specified, they cannot serve in disambiguation. On the other hand, overspecification should also be avoided. Indeed, in encoding lexicons there is a temptation to overspecify. Consider the following pairs of examples in regard to lexical constraints:

He kicked the bucket.	The bucket was kicked.
Mary was taken by the car dealer.	The car dealer took her.
He put his foot down.	He put down his foot.
She laid down the law.	She laid the law down.
He took on Goliath.	He took on him.

There is a tendency to incorporate in the lexicon syntactic restrictions which will prevent the instances on the right. For example, *kick the bucket* would be marked as *active-voice-only*. This is in contrast to the phrase *bury the hatchet* which maintain its figurative flavor also in the passive voice: *the hatchet was buried by Israel and Egypt*.

We believe that this behavior is not dictated by an arbitrary, ad hoc syntactic restriction, rather it reflects the conceptual representation of the phrase as it has been shaped in the acquisition process [Zernik87]. The acquisition of the phrase *bury the hatchet* was based on a metaphor, and generalized from single-word meanings.

Bury was generalized into *disenable-use*, and the referent *the hatchet* was generalized to a tool, the availability of which is a precondition for an active conflict. Therefore, the reference to *the hatchet* stands for a certain generalized object. On the other hand, *kick the bucket* was learned as a whole chunk, since the underlying metaphor remained unresolved. Thus, the referent *the bucket* is maintained as a literal not associated with any concept. Due to this difference, there may arise a discourse function for passivizing *bury the hatchet*. However, since there is no referent for *the bucket*, there will never occur the need to passivize that phrase. Therefore, marking the phrase pattern as *active-voice-only* is redundant (albeit correct).

Another issue is verb-modifier separation, i.e.: *David took on Goliath* vs. *He took him on*. How can the lexicon account for this separation phenomenon? A grossly overspecified rule claims that pronouns (and only pronouns) separate such two-word verbs. However, there are counterexamples such as:

He took that ugly giant on.

(where the separation is by a lengthy reference). Therefore the rule must be revised to relate the phenomenon to *given* and *new* references. A given, or an already resolved reference, can separate, while a new reference cannot be placed between the verb and its modifier. We believe that this behavior should not be specified by the lexicon, rather the generation decision is according to discourse functions.

Overspecified lexical entries can always be contradicted by instances in context. In order to avoid the such contradictions we take the approach of maintaining syntactic specifications of lexical entries at appropriate levels, and use conceptual representation to account for apparently syntactic restrictions.

2.3 LEXICAL REPRESENTATION: PREVIOUS WORK

DHPL is a continuation of efforts in three distinct areas. First, in integrating the underlying situation as part of the lexical entry, we extend previous work on lexical presupposition. Second, we modify Wilensky's method of lexical representation for use in language acquisition. Third, we examine Bresnan's system of linguistic representation, which proves problematic in light of the acquisition task, and compare it to DHPL's representation.

2.3.1 Lexical Presupposition

A message might be conveyed by an utterance beyond its straightforward illocution. That message, called the *presupposition* of the utterance, is described by Keenan (1971) as follows*:

The presuppositions of a sentence are those conditions that the world must meet in order for the sentence to make literal sense. Thus if some such condition is not met, for some sentence S, then either S makes no sense at all or else it is understood in some nonliteral way, for example as a joke or metaphor.

Despite this definition of presupposition as a *condition* for application of lexical knowledge, presupposition has been studied as a means for generation and propagation of inferences, reversing its role as a condition. In [Gazdar79, Karttunen79, Keenan71] the goal has been to compute the part of the sentence which is already *given*, by applying "backward" reasoning, i.e.: from the sentence *the king of France is bald* determine if indeed there is a king in France, or from the sentence *it was not John who broke the glass*, determine whether somebody indeed broke the glass. Rather than using presuppositions to develop further inferences, we investigate how presuppositions are actually applied according to Keenan's definition above, namely, in determining appropriate utterance interpretations.

Fillmore [Fillmore78] introduced lexical presupposition to describe situations in which lexical items may appear. He described the meanings of judgement words such as *accuse*, *criticize*, *blame*, and *praise*, by separating the entire meaning into (a) a statement (the *illocutionary act*), and (b) a presupposition. We illustrate this distinction by comparing the meanings of *criticize* and *accuse* in the following sentences:

(30) John criticized Mary for adjourning the meeting.

(31) John accused Mary of adjourning the meeting.

In both sentences, John referred to a hypothetical act, namely adjourning the meeting. In (30), it is **presupposed** that Mary committed the act (a test for determining presupposition is invariance under negation: John did not criticize Mary of ad-

* (See also [Grice75], and [Fauconnier85] Ch. 3)

journing the meeting still implies that Mary committed the act), while it is **stated** that the act is judged negatively. In (31), on the other hand, it is **stated** that Mary committed the act, while it is **presupposed** that the act is negative.

We believe Fillmore's approach is suitable also for the task of language acquisition, since learning involves factoring out the statement of a phrase from the entire surrounding context. We have further pursued Fillmore's notion in utilizing lexical presupposition in specific tasks such as disambiguation, indexing, and accounting for communicative goals [Gasser86].

Presupposition must be distinguished from precondition. Consider the following text.

John ran into a pedestrian on a red light.
He managed to **explain it away** in court.

The lexical phrase under consideration is *explain away*. The presupposition for the *application of the phrase* is the entire situation in which the phrase typically appears. A person is attempting to justify a certain planning failure. The *precondition* for the *enablement of the act*, on the other hand, is a planning element from the domain itself. One precondition in the story above could be the judge's permission for John to stand up in court and defend his own case. Another trivial example is the sentence below.

John threw a rock at Mary.

There is no presupposition for the generic phrase *person throw phys-obj*. This phrase may appear in almost any context. However, from a planning point of view, for a person to throw a rock she must first grasp the rock in her hand. In contrast to presupposition, such planning information should not reside in the lexicon. In fact, any information which could be derived by means of general world knowledge does *not* belong in the lexicon.

Dyer [Dyer83] has described text comprehension as an integrated cognitive process. Parsing, he claimed, cannot be separated from other cognitive tasks such as memory update and retrieval. Accordingly, *search demons* were introduced in lexical entries to perform memory retrieval. For example, consider the difference between the two sentences.

(32) John made up his mind.

(33) He decided to go swimming.

In parsing sentence (33) the selected plan, namely going swimming, is mentioned explicitly. However, in sentence (32) neither the plan nor the problem to be resolved are mentioned explicitly. Therefore, a search demon associated with the phrase *make up one's mind* is dispatched to retrieve from memory the problem under consideration by the actor of the phrase. One of the objectives of DHPL representation is to eliminate such procedural knowledge. Lexical presupposition serves the task of memory retrieval. The mechanisms we use are unification and variable binding.

2.3.2 Language as a Knowledge-Based System

Wilensky [Wilensky81] promoted the view of language processing as a knowledge-based task. Accordingly, he suggested representing linguistic knowledge as a database of rules given at various levels of generality. The basic representation element is called a *phrase*, given as a *pattern-concept* pair. For example, the phrase in the sentence:

John dropped out of police academy.

is given as the phrase

pattern: <?x> <drop> out of <?y:school>
concept: goal-pursue-education by ?x terminated unsuccessfully

Parsing is viewed as a process of rule (phrase) application. When more than one rule is applicable (ambiguity), selection is by *specificity*, namely, the most specific phrase is selected.

An additional layer was added to this work by Jacobs [Jacobs85a] who noticed the need for inheritance and hierarchy in the lexicon. Concepts in memory are organized in a hierarchy of categories, through which more specific concepts can inherit features from more general ones. Concepts in the lexicon, namely lexical items, should be organized through the same general discipline. This approach enjoys three advantages:

- o **Modularity:** Adding a new entry does not require any global modification.
- o **Declarativeness:** The representation is neutral with respect to parsing and generation. The representation does not reflect any programming style (beyond

basic slot-filler notation) and it does not reflect the mechanism of any particular parser.

- o **Uniformity:** Modifying the level of generality of a phrase does not require a change of the phrase beyond the single feature being updated (generalized or specified).

These properties make the system more amenable to modeling language processing [Kay79] and acquisition [Mitchell82].

2.3.3 LFG and Language Acquisition

Bresnan's [Bresnan82b] linguistic representation, *lexical functional grammar* (LFG), is a system with a "flat" lexicon, which does not define a hierarchy of generalizations. LFG is contrasted here with DHPL's hierarchical approach, and it is examined here in regard to learning [Pinker84]. In LFG there are two lexical entries representing the word *ask*, as it appears in the following sentences.

- (32) John asked to leave.
- (33) John asked Mary to leave

The corresponding lexical entries are given respectively below.

<code>ask: V:PRED = "ask(SUBJ, V-COMP) "</code>	
<code> SUBJ = V-COMP' s SUBJ</code>	(subject-equi)
<code>ask: V:PRED = "ask(SUBJ, OBJ, V-COMP) "</code>	
<code> OBJ = V-COMP' s SUBJ</code>	(object-equi)

Figure 2.1: LFG representation of ASK

The meaning of *ask* is given as the predicate *ask* which takes either two or three arguments. There is no general notion which captures the similarities in the behavior of the two specific entries. In the hierarchical approach, on the other hand, the behavior of *ask* is described in the broader context of the infinitive interaction between phrases.

The schematic hierarchy is given in Figure 2.2 below:

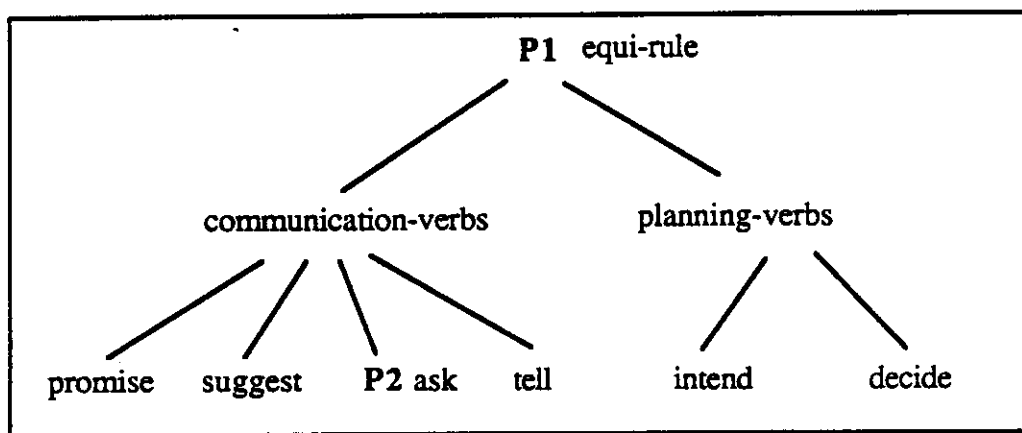


Figure 2.2: ASK as Part of a Broader Hierarchy

In this scheme, there is a single phrase for *ask* (P2). This phrase draws properties from a more general phrase (P1) which defines the general *equi rule* in complement-taking English verbs. In this representation, the behavior of *ask* is inherited from the general phrase P1 and there is no need to duplicate specific cases.

LFG current theory does not facilitate such hierarchies. In absence of hierarchy and inheritance, there is a need for duplication of the learning effort and can lead to serious flaws in modeling human behavior. For example, the word *promise* presents an exception to the general equi rule. Consider *John promised Mary to go*, in contrast to *John asked Mary to go*. The latter implies that John is the actor of the future act of going (John promised that he will go, but John asked that Mary go). In learning this behavior of *promise*, children make an error by hypothesizing the default equi rule, thus committing an error of *overgeneralization* (a child might say: *Dad promised Tommy to drive the big car alone* meaning "Tommy will drive the car"). In LFG it is impossible to model this behavior since generalizations do not exist. Indeed, Pinker [Pinker84] accounted for this error, but the equi rule he resorted to is not part of the LFG system itself. Moreover, through LFG it is impossible to *recover* from overgeneralization. Normally people recover from overgeneralizations by being given a counterexample (No. *Dad promised Tommy to take him to Disneyland*). However, since neither Bresnan nor Pinker attempt to represent meanings of words such as *take* and *drive* – the meanings are actually represented as the symbols *take*

and *drive* – it is impossible to make the necessary semantic inferences for error recovery. Thus, without the ability to generalize and without an appropriate representation of concepts, LFG as currently defined, cannot account for these behaviors in learning.

2.4 REPRESENTING THE CONTEXT

The semantics of entries in the lexicon draw from the various contexts in which they have been applied. Here we represent contexts using scripts, plans, goals, and relationships [Schank77, Dyer83, Dyer86]. Consider, for example, the context in reading the text:

```
Al Capone went on trial.  
The judge threw the book at him.
```

The underlying knowledge is the the *trial script*, which captures the basic events taking place in court.

-
- (a) The Prosecutor communicates (mtrans) his arguments.
 - (b) The Defendant communicates his arguments.
 - (c) The Judge decides (select-plan) either:
 - (1) Punish (thwart a goal of) Defendant.
 - (2) Do not punish him.
-

Figure 2.3: The Acts in \$Trial

This script, as shown in Figure 1, consists of a sequence of four events, in which the characters are the judge, the prosecutor, and a defendant. In addition, there is knowledge of the character's goals. The prosecutor is interested in thwarting a preservation goal – p-freedom, p-property of the defendant. The defendant attempts to block this goal thwart. Both parties advance their cases by trying to convince the judge. By this representation the meaning of the phrase to throw the book at somebody means *to punish him severely*, based on events (a) and (1) in the script.

Another situation, involving the same script, is presented in the following text.

```
John ran over a pedestrian.
```


He failed to **explain it away** in court,
and he went to jail.

In this case the phrase *explain away* pertains to the underlying goal-plan situation, given in Figure 2.4 below.

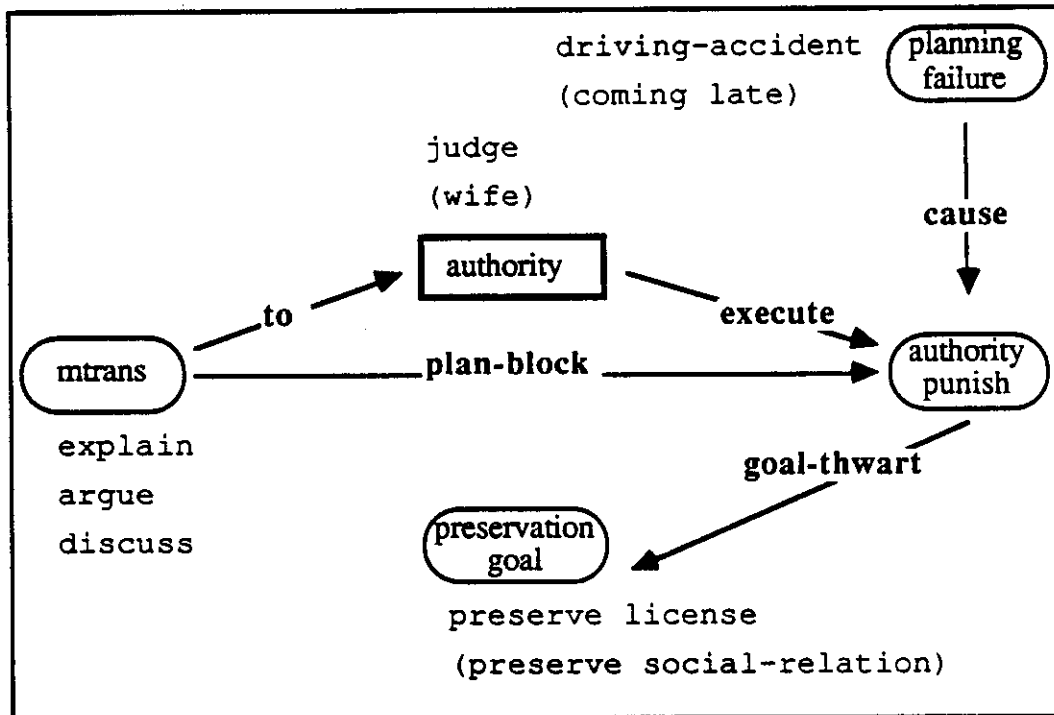


Figure 2.4: The Goal-Plan Structure for *explain away*

John experienced a planning-failure (failed plan of driving safely). John's preservation goal of freedom is threatened. A plan for preserving this goal is convincing the judge as to why John himself was not at fault. This second plan is executed and it fails also. Thus, his p-goal fails.

Notice that the same goal-plan schema exists also in the case of the next story:

Joe forgot to put away the dirty dishes.
When his wife came home, he **argued it away**
by telling her he had been working.

The phrase *argue away* also involves a prior plan failure, a thwarted p-goal (*p-social-relation*) and a recovery plan of convincing the other party. This underlying schema is a presupposition. It holds whether Joe fails to argue it away or whether he manages to argue it away. Since the same plan-goal schema underlies both phrases (up to the specific plan: argue vs. explain), they both can be viewed as instances of a more general phrase.

Many other phrases draw their meanings in terms of such general plan-goal structures. Consider the phrases in the next sentences:

This machine was **idling away** for hours.
 They stayed at home, and **argued away** for hours.
 The class was boring. John sat near the window **dreaming away**.

In all these sentences there is a similar underlying situation, shown in Figure 2.5 below.

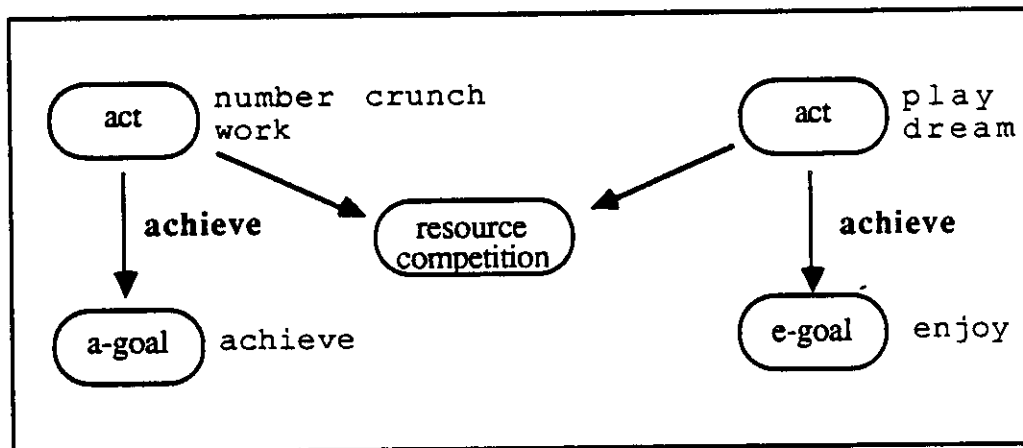


Figure 2.5: The Goal-Plan Structure for *idle away*

In this schema a resource competition (the resource is time) exists for an agent between two competing tasks, and that agent subordinates the important goal.

The fact that phrase representation can be elevated to a level of general plans and goals is very significant. It implies that *a relatively small number of structures can represent phrases whose instances can be used across many domains.*

2.5 CONCLUSIONS

We have shown the issues in lexical representation, in regard to language acquisition.

- The lexicon must first provide the conditions for disambiguation, based on the context.
- Lexical entries must be presented at various levels of generality, to account for various of cases of partial knowledge.
- Knowledge must be given in a way that is *declarative, uniform, and modular*, so that learning is not complicated.

We have shown how are approach draws from three previous lines of research:

- Lexical representation has been defined as the element which guarantees *felicity* of phrase application. However, in spite of this definition, previous systems have not used presupposition in the task of semantic disambiguation.
- The *knowledge-based* approach to language has promoted representation which is amenable to general computational mechanisms, such as unification. This approach is suitable to acquisition, in particular in light of Mitchell's [Mitchell82] theory of learning in a version space.
- Current linguistic theories such as LFG [Bresnan82a] have failed to take into account the two consideration above. LFG must be extended in two ways: (a) so that it maintains a hierarchy, and (b) so that it maintains contextual conditions.

Chapter 3:

Organizing the Lexicon

How can a lexicon account for a dynamically evolving language? In this chapter we describe a Dynamic Hierarchical Phrasal Lexicon (DHPL), in which:

- (1) Lexical entries are represented not as single words but as entire phrases.
- (2) Lexical entries are organized in a hierarchy by generality. Specific phrases reside at the bottom, and general grammar rules, also given as phrases, reside at the top. There is no separate grammar in this system.
- (3) At each stage, the lexicon presents the partial knowledge possessed by a learning model. The lexicon can be updated dynamically.

We show how this organization relates to the issues raised in the previous chapter:

- (1) The lexicon provides the means for semantic disambiguation, relative to the context.
- (2) The lexicon enables parsing even in conditions of partial knowledge.
- (3) The lexicon allows learning of phrases by propagation in the lexical hierarchy: phrases can be generalized and specialized incrementally.

3.1 THE LEXICON: CONTENTS AND STRUCTURE

We present here a dynamic lexical structure. Normally, linguistic systems account for spanning a static languages. In DHPL, a *Dynamic Hierarchical Phrasal Lexicon*, we propose a system which facilitates a *dynamic* linguistic behavior. The structure of this lexicon is specified by (a) the structure of a single lexical element, and (b) the global structure in which elements are interconnected.

3.1.1 Basic Phrases

We first describe the structure of a single lexical entry. Consider the marked clause in the following text.

For years they tried to prosecute Al Capone.
Finally, a judge threw the book at him for income-tax evasion.

This clause is derived from a lexical phrase which is given as the following simplified template:

phrase

pattern:	Person1 throw the book at Person2.
presupposition:	Person1 is an authority for Person2.
concept:	Person1 punishes person2 severely.

This lexical *phrase* is a triple associating a linguistic *pattern* with its semantic *concept* and *presupposition*. The *pattern* specifies the syntactic appearance in text. The *presupposition* specifies the surrounding context, while the *concept* specifies the meaning added by the phrase itself. Phrase *presupposition*, distinguished from phrase *concept*, is introduced in DHPL's representation since it solves three problems: (a) in *disambiguation* it provides a discrimination condition for phrase selection, (b) in *acquisition* it allows the incorporation of the context of the example as part of the phrase, and (c) in *generation* it provides an indexing scheme for phrase discrimination and triggering.

The role of the three slots in a phrase template may be better understood by the way they are applied in parsing the text above. The clause is parsed in four steps:

- (1) The pattern is matched successfully against the text. Consequently, Per-

- son1 and Person2 are bound to the judge and to Al Capone respectively (as the *person* class *restrictions* imposed by the pattern are satisfied).
- (2) The presupposition associated with the pattern is validated using the concepts in the context. Using knowledge of human relationships, it is inferred that the judge presents an authority to Capone.
 - (3) Since both (1) and (2) are successful, then the pattern itself is instantiated, adding to the context: *The judge punished Al Capone severely.*
 - (4) Steps (1)-(3) are repeated for each relevant lexical entry. If more than one entry is instantiated, then the concept with the best match is selected.

Actual Slot-Filler Notation: The actual representation of the phrase is implemented using GATE's [Mueller87] slot-filler language, as shown below. This representation of a phrase, which is a linguistic object, is not different from the representation of other objects in the database.

```

(comment (y THROW THE BOOK AT x))
(pattern ?x throw <the book> <at ?y>
(presupposition
  ((head authority)
   (class soc-relation)
   (high ?y)
   (low ?x)))
(concept
  ((head auth-punish)
   (class event)
   (from ?y)
   (obj ((head thwart-goal)
         (goal ((goal-of ?y)
                (class p-goal))))))
  (to ?x)))

```

Figure 3.1: The Phrase Notation

Notice that the phrase consists of three main parts: pattern, concept and presupposition (the comment is for reference only).

Case-Frame Representation: The pattern of the phrase above can be written as:

?x throw <the book> <at ?y>

This is an abbreviation which stands for the full notation given below.

```
(subject ((class person)
          (instance ?x)))
(verb    ((root throw)))
(object1 ((determiner the)
          (root book)))
(object2 ((marker at)
          (class person)
          (instance ?y)))
```

This full notation has three features:

- (1) The pattern is constructed of four case frames [Carbonell84].
- (2) Case frames are named. For example, *object2* is the name of the case frame given as:

```
(marker at)
(class person)
(instance ?y)
```

This case is referred to as the lexical subject to be distinguished from the *surface subject* (the element actually preceding the verb in the text).

- (3) Case frames are unordered, namely no order is imposed among the case frames. In no place in the case frame is it mentioned, for example, that the lexical subject should precede the verb or follow it (or not appear at all). Case ordering, thus, is inherited from general linguistic patterns, as shown later in this paper).
- (4) Case frames contain both semantic and syntactic properties. For example, *object1* defines the named constituents *the* and *book*, while *object2* defines the class *person*.

Since not all properties are given explicitly within the pattern itself, there is a need for

an *inheritance* scheme. Properties such as case order (e.g. active and passive voice), and word-order of the syntactic constituents within cases (e.g. the determiner the precedes the root book) are inherited from general linguistic patterns.

3.1.2 The Global Structure

While varying in generality, lexical entries are represented uniformly throughout. The lexicon can be viewed as a collection triples (Pattern-Concept-Presupposition), as shown in Figure 3.2, which are retrieved for parsing and for generation tasks, and become operational by unification.

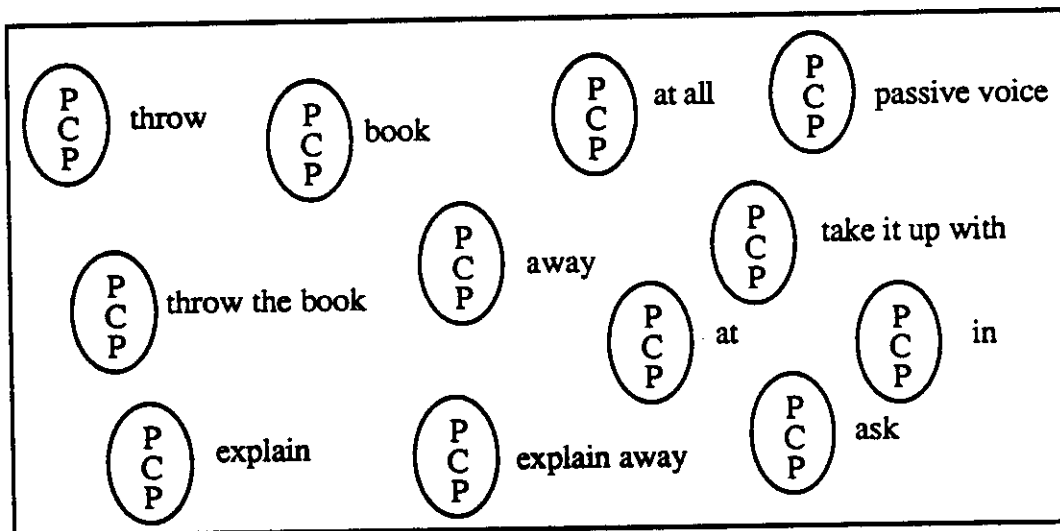


Figure 3.2: The Lexicon as a Collection of Triples

To facilitate learning, these triples are organized in hierarchies by generality. In a hierarchical scheme, the bottom nodes are very specific and idiomatic while the ones at the top are more general. Phrases may reside at and inherit from, more than one hierarchy. For example, the phrase to take on can inherit from the hierarchy of take as well as from the hierarchy of on (a hierarchy which defines properties of verb modifiers). Four operations, implemented as forms of unification, and are defined by this representation. They are: (a) *interaction* between two unrelated phrases, (b) *inheritance* between two related phrases (one more general than the other), (c) *generalization*, and (d) *discrimination* of a phrase, which both update its level of generality. Three hierarchy schemes are given in the following sections to demon-

strate three aspects of the system: (a) phrase interaction through the infinitive construction, (b) word-sense representation, and (c) case-order.

3.2 REPRESENTING THE INFINITIVE

Consider the following pair of clauses:

- (1) Judge Wilson threw the book at him.
- (2) Judge Wilson decided to throw the book at him.

Parsing the first sentence is carried out simply as a lexicon lookup: a phrase is found in the lexicon, and its concept is instantiated. Parsing the second sentence is more complex since no single lexical phrase is matched for *throw*. For one thing, the subject does not precede the verb *throw* as anticipated by the lexical pattern. Identifying the implicit subject involves knowledge of phrase interaction. Properties of phrase interaction (through the infinitive form [Kiparsky71]) are represented by a hierarchy below.

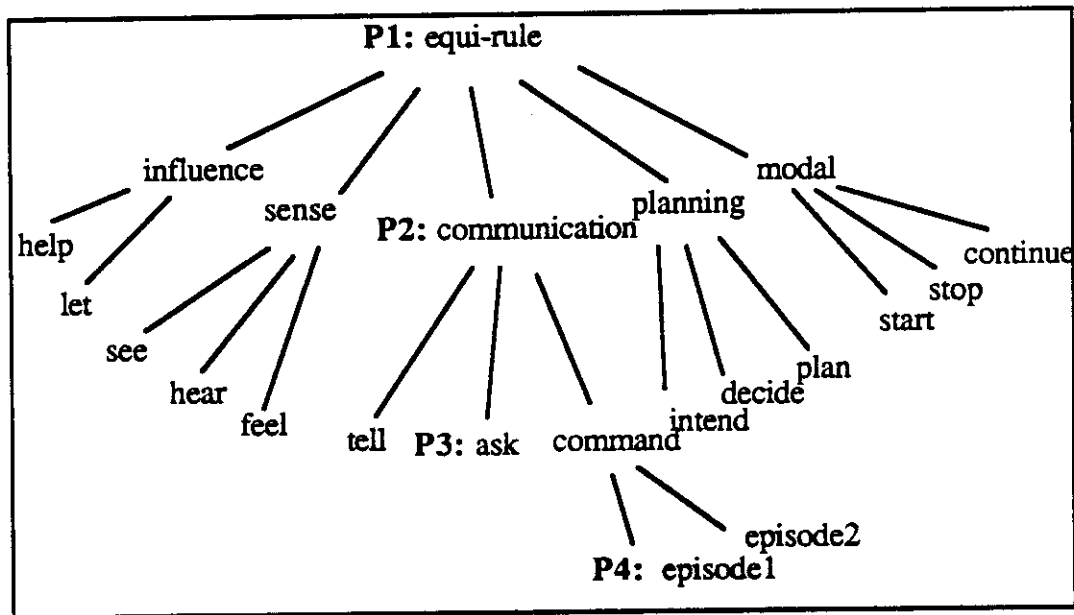


Figure 3.3: The Hierarchy for Phrase Interaction

The names of the individual nodes are mnemonic, and are used for reference only. Each such node is a full pattern-concept-presupposition triple (the presupposition may not appear). The nodes in Figure 3.3 are described as follows:

(a) The most general node (P1) denotes the basic *equi rule*, which stands for the following object:

```
(comment the general EQUI behavior)
(pattern ((subject ((instance ?x))
                  (verb ((root ?v))
                        (object1 ((instance ?y))
                                  (comp ((pattern
                                        ((subject ((instance (and ?y ?x)))
                                                  (verb ((form infinitive))
                                                        (concept ?z))))))))))
          (concept ((actor ?x)
                    (obj ?z)))
```

In this phrase, notice in particular the *complement* (*comp*), which defines the embedded phrase. The implicit subject of the embedded phrase is taken as either (1) the object of the embedding phrase, if that object exists, or (2) the subject of the embedding phrase, if the object does not exist.

(b) Middle-level nodes encompass classes of verbs. For example, P2 encompasses *communication* verbs such as ask, tell, instruct, etc., share certain features. It is represented as follows:

```
(comment communication verbs)
(pattern ((subject ((instance ?x))
                  (verb ((root ?v))
                        (object1 ((instance ?y))
                                  (comp
                                   ((pattern
                                    ((subject ((instance (and ?y ?x)))
                                              (verb ((form infinitive) (comp'er to))
                                                    (concept ?z))))))))))
          (concept ((head mtrans)
                    (actor ?x)
                    (to ?y)
                    (plan ?z)))
```

This phrase is similar to the phrase P1. However, it includes information specific to that class of verbs. It defines shared syntactic features: subject, verb, object, complement (where the *complementizer* is *to*). It also defines shared semantic properties: (a)

the equi-rule, (b) the concept of the complement, which is a hypothetical, future plan communicated by the actor.

(c) *Specific* nodes give the behavior of individual verbs, such as the phrases for decide (a *planning* verb) and command (a *communication* verb).

```
(comment X DECIDE TO Z)
(pattern ((subject ((instance ?x)))
          (verb    ((root decide)))
          (comp
            ((pattern
              ((subject ((instance ?x)))
                (verb    ((form infinitive) (comp'er to)))
                (concept ?z)))))))
(concept ((head plan-select)
          (actor ?x)
          (plan  ?z)))

(comment X COMMAND Y TO Z)
(pattern ((subject ((instance ?x)))
          (verb    ((root command)))
          (object1 ((instance ?y)))
          (comp
            ((pattern
              ((subject ((instance ?y)))
                (verb    ((form infinitive) (comp'er to)))
                (concept ?z)))))))
(presupposition
  (head authority)
  (high ?x)
  (low ?y))
(concept ((head mtrans)
          (actor ?x)
          (to ?y)
          (obj  ((active-goal ?z)
                (goal-of ?x))))))
```

Each one of these phrases adds on the information specific to the denoted verb. According to this representation ?x command ?y to ?z means that ?x who presents an authority to ?y, tells ?y that ?z is a goal of ?x.

(d) *Episodes* such as P4, which include specific instances of a phrase, are indexed to the phrase. For example, P4 is the situation in which God commands Moses to approach the Mountain. This episode contains the semantic ingredients constituting the meaning of the phrase.

The hierarchy of Figure 3.3 is used by four processing tasks.

3.2.1 Phrase Interaction

The analysis of sentence (2):

(2) Judge Wilson decided to throw the book at him.

involves the interaction of two specific phrases, as shown schematically in Figure 3.4.

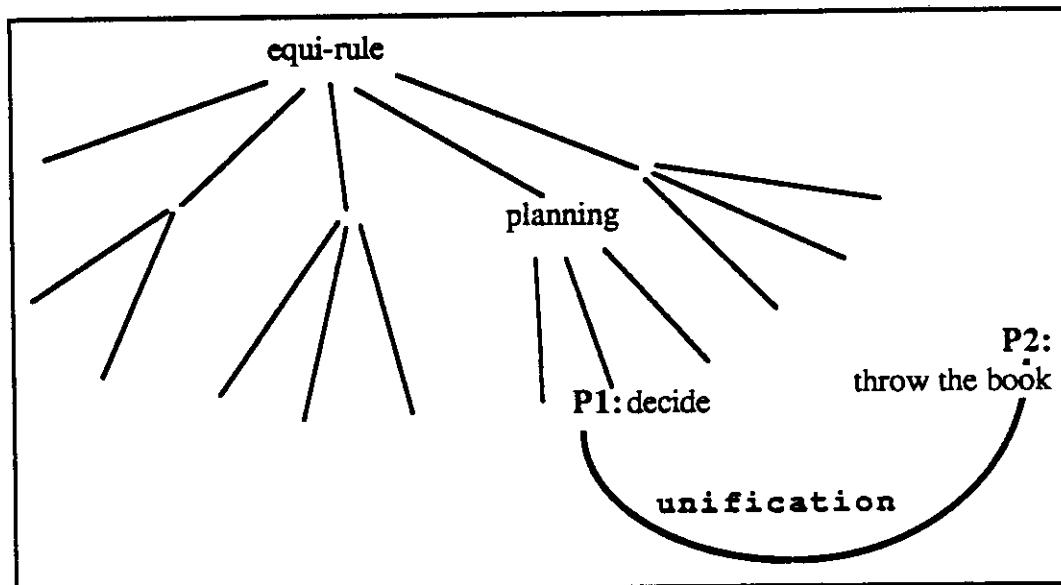


Figure 3.4: Interaction of Two Specific Phrases

The two specific lexical phrases involved are the entries for *decide* (the embedding phrase, P1, elaborated in item (c) at the beginning of Section 3.1 above) and for *throw the book* (the embedded phrase, P2, described in Figure 3.1 above). The *unification* of these two phrases guarantees that: (a) the subject of P1 is the subject of P2, and (b) the concept of the P2 (denoted by ?z) is plugged in the *plan* slot of P1. The interaction of these two phrases yields the compound concept:

```

(head plan-select)
(actor wilson.1)
(plan ((head auth-punish)
      (actor wilson.1)
      (to capone.2)))

```

This concept conveys the meaning of the entire sentence.

3.2.2 Parsing an Unknown

In contrast to the previous example, consider the analysis of a sentence in which an unknown word is included:

Mary goggled John to come over.

In analyzing this sentence, no lexical phrase is found to account for the word *goggle*. Therefore, the meaning of the entire sentence cannot be produced. Yet, even a partial meaning cannot be produced for the known clause, *to come over*, since it is intertwined with the unknown clause *Mary goggled John*. In order to overcome this obstacle, the interaction involves a more general phrase as shown in Figure 3.5.

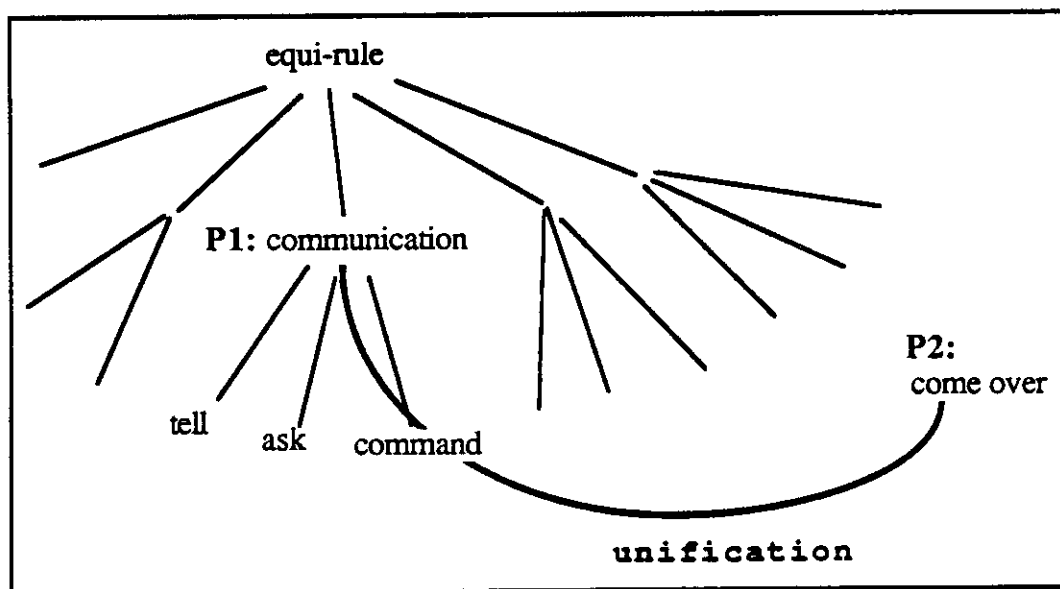


Figure 3.5: Interaction with a Generalized Phrase

In contrast to Figure 3.4, here no specific phrase could be found for *goggle*, and it was necessary to select the generalized phrase, P1, which encompasses *communica-*

tion verbs in general. For *come over*, on the other hand, there exists a specific entry in the lexicon, P2, thus a generalization is not sought for. The *partial* meaning constructed for the sentence, in absence of a phrase for *goggle* is:

```
(head mtrans)
(actor mary.1)
(to john.2)
(obj ((head ptrans)
      (actor john.2)
      (to mary.1)))
```

Thus, even when the particular phrase does not exist, the parser is able to construct an initial hypothesis, based on a generalization.

In fact, the selection of the generalized phrase is not unambiguous. The nature of the selected phrase is restricted by two schemes: (a) the hierarchy in Figure 3.3 above, and (b) the *persuade plan box* [Schank77] which provides the planning options available for a person in persuading another person to act (overpower, threaten, promise, steal, etc.). Accordingly, *goggle* could have as well conveyed meanings such as:

- (3) Mary pushed John to come over. (Influence verb)
- (4) Mary let John come over. (help verb)
- (5) Mary threatened John to come over. (promise verb)

Indeed option (5) is not available in English, however, since the phrase is yet unknown to the learner, this option must be given consideration.

3.2.3 Overgeneralization and Recovery

In the case that the word *promise* does not exist in the lexicon, the program behaves as follows:

User: John promised Mary to come over.

RINA: John told Mary that she must/can come to him.

In using the generalized phrase, RINA unified inappropriately the roles. This is an error of overgeneralization which is typical of children learning new vocabulary items.

3.2.4 Error Recovery

The user can correct the program by giving an explicit example.

User: No. John promised Mary to come to her place.

By using few inferences (e.g., person ?x does not come to the same person ?x), RINA figures out the confusion in the role-binding and corrects appropriately the phrase for `promise`, as given below:

```
(comment X PROMISE Y TO Z)
(pattern ((subject ((instance ?x)))
         (verb      ((root promise)))
         (object1  ((instance ?y)))
         (comp
          ((pattern
            ((subject ((instance ?x)))
             (verb      ((form infinitive) (comp'er to)))
             (concept ?z)))))))
(presupposition
 (head goal)
 (goal-of ?y))
(concept ((head plan-select)
         (actor ?x)
         (to    ?y)
         (plan ?z)))
```

Notice two interesting points regarding the semantics of `promise`: (a) ?x (the embedding subject) is always the subject of the embedded phrase, and (b) the act ?z is presupposed to be a goal of ?y. ?x is the subject of the embedded act, and the act ?z is presupposed to be a goal of ?y.

3.3 HANDLING WORD SENSES

By its nature, the phrasal approach is oriented towards the representation of entire groups of words. However, single words, such as `up`, `at`, and `away` must also be represented. Three issues are involved in representing such words.

3.3.1 Assigning Meanings to Particles

Compare the following two sentences:

- (6) John looked **up** at Mary.
- (7) John looked at Mary.

The meanings of the two sentences are given below*:

- | | | | |
|-----|------------------------|-----|---------------------------------------|
| (6) | (head attend) | (7) | (head attend) |
| | (obj eyes) | | (obj eyes) |
| | (actor john.3) | | (actor john.3) |
| | (to mary.4) | | (to mary.4) |
| | | | (direction vertical-positive) |

The contribution of the particle **up** is given as (**direction vertical-positive**). The role of the particle in the next sentence is less obvious.

- (8) John **flew away** from the scene of the crime.

What is the contribution of the word **away** to the meaning of sentence (8)? For instance, how is the meaning of sentence (8) different than the meaning of sentence (9) below?

- (9) John **flew** to Alaska.

3.3.2 Resolving Word-Sense Ambiguity

Is the contribution of **away** identical in all the sentences (10)-(13), or are there several meanings involved?

- (10) John **flew away** from the scene of the crime.
- (11) John did not **put away** the clean dishes.
- (12) He managed to **argue it away** with his wife.
- (13) This machine was **idling away** for hours.

For example, consider two appearances of the production **argue away** which involve two different senses of **away**:

* Another phrase, John looked up to Mary, in contrast to John looked up at Mary, is NOT processed as a simple production of the particles, since it involves the entire phrase "X look up to Y".

- (14) His lawyer can **argue away** any tax violation.
 (15) He is a bum. He can **argue away** for hours without
 convincing anybody.

The first sense implies success in deceiving the authorities (as in *get away with*), while the second sense implies a waste of time (as in *idle away*). If there is more than one sense for *away*, then how is the appropriate meaning selected in each instance?

In our lexicon, there are two phrases for *argue away*, which are disambiguated by matching their presuppositions with the context. The two phrases are:

pattern: ?x <argue away> ?y
presupposition:
 1. ?y is a planning failure by ?x.
 2. ?g is a goal of ?x thwarted by authority punishment ?a.
 3. ?v (argue) is a communication act by ?x for blocking act ?a.
concept: act ?v is successful, and the goal thwart is removed.

pattern: ?x <argue away>
presupposition:
 1. act ?v (arguing) serves no achievement goal of ?x.
 2. ?v is in a resource conflict over time with goal ?g
concept: act ?v is selected over a long period of time,
 causing ?g to be abandoned.

Figure 3.6: Two Different Senses for *argue away*

The appropriate phrase is selected in each context by matching the presupposition.

3.3.3 Determining Level of Generality

Which is the appropriate alternative for representing the phrase in sentence (16)?

- (16) He managed to **argue it away** with his wife.

- (a) Is it as "fixed" phrase as given below?

pattern: ?x <argue away> ?y <with ?z>

concept: ?x managed to explain event ?y
to person ?z by arguing.

(b) Or is it a "variable" phrase as given next:

pattern: ?x <?v away> ?y

concept: ?x managed to explain event ?y
to person ?z by act ?v.

Answers for these dilemmas are given by the hierarchy in Figure 3.7 below:

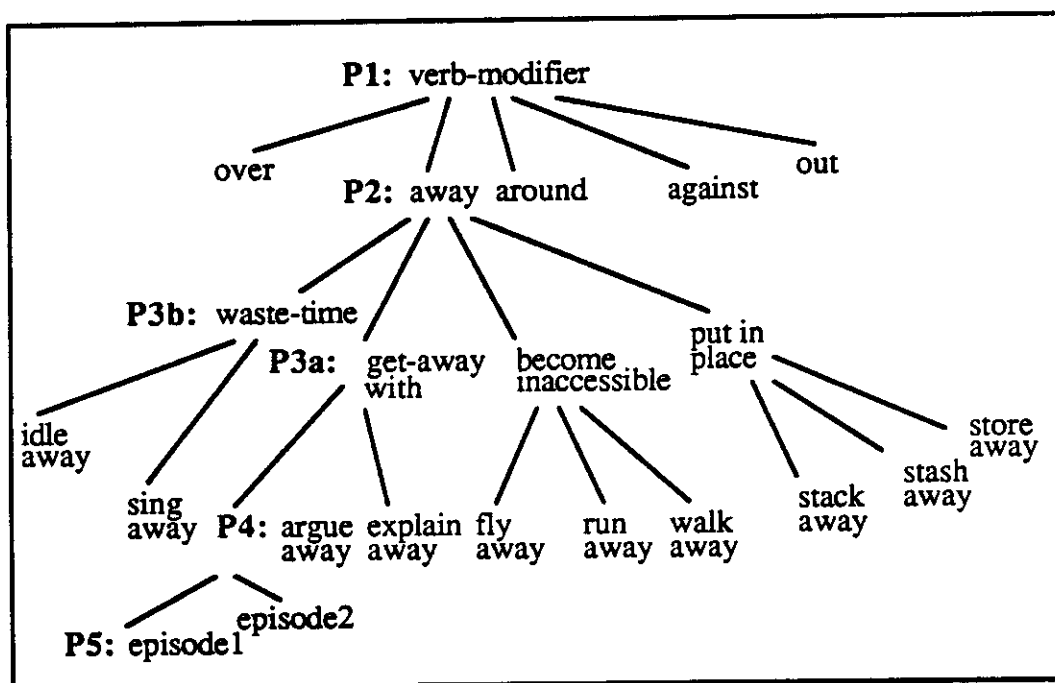


Figure 3.7: The Hierarchy for *away*

The most general phrase (P1) denotes the general properties of English verb modifiers. The modifier follows the verb, but separation is allowed (i.e.: he explained it away vs. he explained away his latest goof).

Second-level nodes represent some general meaning conveyed by words such as *away* (P2), up and down. The pattern for P2, for example is <?v away>, where ?v can be any verb.

Nodes at the third level convey word senses which encompass classes of specific phrases. For example, P3a (*convince*) conveys the meaning encompassing both explain it away and argue it away, while P3b (*waste time*) conveys the meaning encompassing both idle away and sing away. These two phrases (P3a and P3b) are elaborated here:

pattern: ?x <?v away> ?y

presupposition:

1. ?y is a planning failure by ?x.
2. ?g is a goal of ?x thwarted by authority punishment ?a.
3. ?v is a communication act by ?x for blocking act ?a.

concept: act ?v is successful, and the goal thwart is removed.

pattern: ?x <?v away>

presupposition:

1. act ?v serves no achievement goal of ?x.
2. ?v is in a resource conflict over time with goal ?g

concept: act ?v is selected over a long period of time, causing ?g to be abandoned.

Figure 3.8: General Phrases for *away*

These two phrases generalize respectively the phrases in Figure 3.6.

Nodes at the next level denote specific phrases, or *productions*, such as run away, argue away (P4), idle away, etc. Such phrases are given in Figure 3.6 for two cases of argue away.

Nodes at the bottom level describe *episodes* in which instances of phrases were encountered (e.g., the instances Al Capone argued it away in court (P5), John Smith argued it away with his wife are indexed to the phrase ?x argue ?y away).

On the face of it, it seems that levels (a) and (d) are sufficient for all parsing and generation purposes. What is the function of levels (b), (c), and (e)?

3.3.4 Analyzing a New Production

These intermediate levels of generalization facilitate the analysis of new productions such as:

(17) John tried to **describe it away** in court.

Sentence (17) introduces a new production to the reader of this paper. Yet, the reader should be able to resolve the new production by using the generalized linguistic pattern P3a in Figure 3.7.

3.3.5 Learning from Examples

In the previous example we have assumed an existing generalized phrase P3a, which was used in predicting a specific phrase. When such a *generality* does not exist, learning must be done by induction from *specific* examples. The following set of examples provide episodes from which RINA can hypothesize the meaning of the phrase to take on.

(18) David took on Goliath.

(19) The *Celtics* took on the *Lakers*.

(20) Finally, I took on the hardest question on the midterm.

So far we have shown two ways of deriving new phrases: First, a new phrase can be *generalized* from indexed episodes (which include instances in context). However, learning is easier when a generalized template already exists, in which case learning is by applying a generality [Zernik85b].

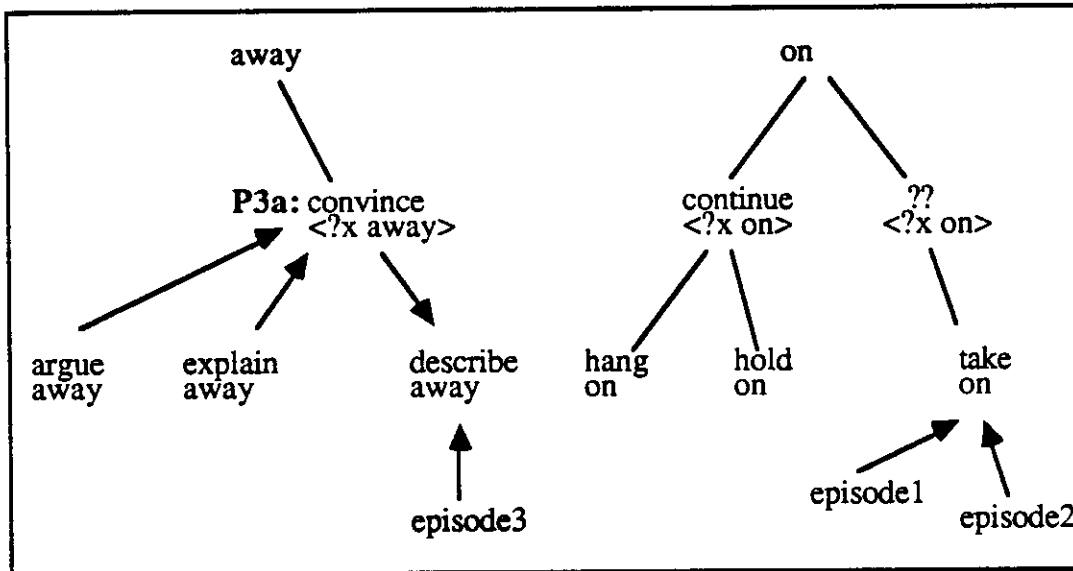


Figure 3.9: Top-Down vs. Bottom-Up Propagation

Figure 3.9 shows two learning processes: *describe it away* is deduced top-down from an existing general concept (P3a). On the other hand, *take on* is induced bottom-up from the set of specific episodes such as David and Goliath, the *Celtics* vs. the *Lakers*, and the midterm. There is no generalized concept which could serve as a short cut.

3.4 INHERITING CASE ORDER

Consider the lexical pattern given as a set of four unordered case-frames:

P0: ?y throw <the book> <at ?x>

Since ordering is not specified explicitly in pattern P0, then how can this pattern match sentences such as:

- (21) The judge threw the book at Al. (active voice)
- (22) The book was thrown at him. (passive voice)
- (23) Al he decided to throw the book at,
but John he gave a break. (left dislocation)
- (24) "Take it easy!" said the prosecutor. (right dislocation)

Under what condition does the *lexical subject* precede the *verb*, and when can the lexical subject be omitted altogether? This information is contained in a case-order hierarchy (Figure 3.10 below) in the lexicon.

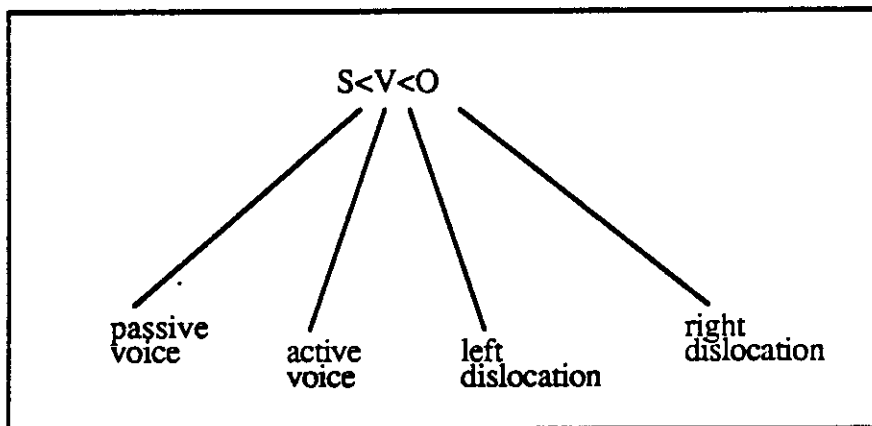


Figure 3.10: Case-Order Hierarchy

The patterns for the *passive* and the *active* voice, for example, are given in the figure below.

```

P2: (subject ((location bef) (marker none)))
     (verb ((location ref) (voice active)))
     (object1 ((location aft)))
     (object2 ((location aft)))
  
```

```

P3: (object1 ((location bef) (marker none)))
     (verb ((location ref) (voice passive)))
     (object2 ((location aft)))
     (subject ((location any)))
  
```

In matching sentences (21) and (22) above, the pattern P0 inherits case-order proper-

ties from these general linguistic patterns. For example, after inheriting the passive voice for matching sentence (22), the pattern augmented by inheritance from P3 would be:

```
P1:  (object1 ((location bef) (determiner the) (root book)
           (instance ?x)))
      (verb   ((location ref) (root throw) (voice passive)))
      (object2 ((location aft) (marker at) (class person)
                (instance ?y)))
      (subject ((location any) (marker by) (class person)
                (instance ?x)))
```

An even more general pattern exists which captures the basic SVO structure of the language. This phrase is given at the top of the hierarchy:

```
(pattern
  (subject ((location bef) (marker none) (instance ?x)))
  (verb   ((location ref)))
  (object1 ((location aft) (marker none) (instance ?y)))
  (object2 ((location aft) (marker ?m) (instance ?z)))
(concept
  (head act) (actor ?x) (patient ?y) (?m ?z))
```

What is the use of that general SVO phrase? This phrase is called for in absence of more specific knowledge. Children who have not yet mastered specific case-structure patterns resort to this pattern. For example, a 2-year-old child might incorrectly understand:

Mary was fed by John.

as if Mary actually fed John. Adults too, in case of missing knowledge, might resort to this generality in making sense out of sentences.

3.5 CONCLUSIONS

We have shown how the Dynamic Hierarchical Phrasal Lexicon (DHPL) supports language analysis, and language acquisition. We accounted for a dynamic language behavior by promoting four aspects of lexical representation:

Phrases: The lexicon contains entire phrases, accounting uniformly for an entire

range including productive as well as non-productive phrase.

Hierarchy: The lexicon organizes in a hierarchy, phrases ranging from specific "lexical entries" at the bottom, to general "grammar rules" at the top.

Lexical Presupposition: Contextual conditions are incorporated into the lexicon through lexical presuppositions. Presuppositions account for disambiguation in parsing, and for phrase selection in generation.

Integration of Syntax and Semantics: Phrases specify a *relation* (in the logical sense) between syntax and semantics. Thus, the question whether any lexical feature is syntax or whether it is semantics, becomes insignificant. For example, consider thematic roles for a phrase such as *promise* (Section 3.2.4). Are they syntactic or are they semantic? They can be viewed as either.

Chapter 4:

Underlying Knowledge

This chapter describes the structures used in semantic lexical representation. The semantics of entries in the lexicon are drawn from contexts in which those entries have been applied. The contexts we considered involved the domain of interpersonal relationships, and planning situations in that domain. It would have been possible to represent phrases in respect to any other domain. However, meanings of phrases in the language, even of specialized and technical phrases, are derived as extension of the interpersonal domain. Accordingly, even an application of a phrase in the computers domain, for example, to `kill a process` [Martin86], is interpreted by mapping it to the basic interpersonal act of one person terminating the life of a second person.

We have selected a set of primitives: acts, goals and plans, relations, and emotions. We show, for a large number of phrases, how they can be represented using these primitives.

4.1 INTRODUCTION

Since English words and phrases are applied in many domains, a question is raised regarding the ingredients of the representation: is the semantic representation given in terms of concepts drawn from each particular domain? In this case there would be a vast number of elements which take part in phrase representation. Alternatively, is it possible to represent phrases using a set of concepts which pertain to all domains? In this case, a relatively small number of elements could cover representation of many phrases. We will answer this question by analyzing for a variety of phrases, how they

are acquired, and how they are applied in text analysis.

4.1.1 The Task Domain

Consider the figurative phrases in the sentences below, as they are parsed by the program RINA.

S1: The Democrats in the house **carried the water**
 for Reagan's tax-reform bill.*

S2: The famous mobster evaded prosecution for years.
 Finally, they **threw the book** at him for tax evasion.

Depending on the contents of the given lexicon, the program may interpret these sentences in one of two ways. On the one hand, assuming that the meaning of a phrase exists in the lexicon, the program applies that meaning in the comprehension of the sentence. In S1, the program understands that the Democratic representatives did the "dirty" work in passing the bill for Reagan. On the other hand, if the figurative phrase does not exist in the lexicon, an additional task is performed: the program must figure out the meaning of the new phrase, using existing knowledge. First, the meanings given for the single words *carry* and *water* are processed literally. Second, the context which exists prior to the application of the phrase, provides a hypothesis for the formation of the phrase meaning. A dialog with RINA proceeds as follows:

RINA: They moved water?

User: No. The Democrats carried the water for Reagan.

RINA: They helped him pass the bill?

Thus, RINA detects the metaphor underlying the phrase, and using the context, it learns that *carry the water* means helping another person do a hard job. Consider encounters with three other phrases:

Jenny wanted to go punk but her father

S3: **laid down the law .**

S4: **put his foot down .**

S5: **read her the riot act .**

S6: **gave her a hard time .**

* This sentence was recorded of the ABC television program *Nightline*, December 12, 1985. At that point, by a peculiar turn of events, the Democrats pushed forwards Reagan's favorite legislation.

In all these cases, it is understood from the context that Jenny's father objected to her plan of going punk (aided by the word *but* which suggests that something went wrong with Jenny's goals). However, what is the meaning of each one of the phrases, and in particular do all these phrases convey identical concepts?

4.1.2 The Issues

In encoding meanings of figurative phrases, we must address the following issues.

Underlying Knowledge: What is the knowledge required in order to encode the phrase *throw the book*? Clearly, this knowledge includes the situation and the events that take place in court, namely the judge punishing the defendant.

The phrase *carry the water*, for example, requires two kinds of knowledge:

- (a) Knowledge about the act of carrying water which can support the analysis of the phrase metaphor.
- (b) Knowledge about general plans and goals, and the way one person agrees to serve as an agent in the execution of the plans of another person. This knowledge supports the analysis of the context.

While the phrases above could be denoted in terms of plans and goals, other phrases, i.e.: *rub one's nose in it*, *climb the walls*, and *have a chip on one's shoulder* require knowledge about emotions, such as *embarrassment* and *frustration*. Unless the program maintains knowledge about *resentment*, the phrase *have a chip on the shoulder*, for example, cannot be represented. Thus, a variety of knowledge structures take place in encoding figurative phrases.

Representing Phrase Meanings and Connotations: The appearance of each phrase carries certain implications. For example, *John put his foot down* implies that John refused a request, and on the other hand, *John read the riot act* implies that he reacted angrily about a certain event in the past. *John gave Mary a hard time* implies that he refused to cooperate, and argued with Mary since he was annoyed, while *John laid down the law* implies that John imposed his authority in a discussion. The representation of each phrase must account for such implications.

Four different phrases in sentences S3-S6 are applied in the same context. However,

not any phrase may be applied in every context. For example, consider the context established by the following paragraph:

S6: Usually, Mary put up with her husband's cooking, but when he served her cold potatoes for breakfast, she **put her foot down**.

Could the phrase in this sentence be replaced by the other two phrases: (a) lay down the law, or (b) read the riot act? While understandable, these two phrases are not appropriate in that context. The sentence she read him the riot act does not make sense in the context of debating food taste. The sentence she laid down the law does not make as much sense since there is no argument between individuals with non-equal authority. Thus, there are conditions for the applicability of each lexical phrase in various contexts. These conditions support phrase disambiguation, and must be included as part of a phrase meaning.

Phrase Acquisition: Phrase meanings are learned from examples given in context. Suppose the structure and meaning of put one's foot down is acquired through the analysis of the following sentences:

S6: Usually, Mary put up with her husband's cooking, but when he served her cold potatoes for breakfast, she **put her foot down**.

S7: Jenny was dating a new boyfriend and started to show up after midnight. When she came at 2am on a weekday, her father **put his foot down**: no more late dates.

S8: From time to time I took money from John, and I would forget to give it back to him. He **put his foot down** yesterday when I asked him for a quarter.

Since each example contains many concepts, both appropriate and inappropriate, the appropriate concepts must be identified and selected. Furthermore, although each example provides only a specific episode, the ultimate meaning must be generalized to encompass further episodes.

Literal Interpretation: Single-word senses (e.g.: the sense of the particle into in run into another car), as well as entire metaphoric actions (e.g.: carry the water in the Democratic representatives carried the water for Reagan's

tax-reform bill) take part in forming the meaning of unknown figurative phrases. Can the meaning of a phrase be acquired in spite of the fact that its original metaphor is unknown, as is the case with read the riot act (what act exactly?) or carry the water (carry what water)?

4.2 THE PROGRAM

The program RINA [Zernik85a] is designed to parse sentences which include figurative phrases. When the meaning of a phrase is given, that meaning is used in forming the concept of the sentence. However, when the phrase is unknown, the figurative phrase should be acquired from the context. The program consists of three components: phrasal parser, phrasal lexicon, and phrasal acquisition module.

4.2.1 Phrasal Parser

A lexical entry, a *phrase*, is a triple associating a linguistic *pattern* with its *concept* and a *presupposition*. A clause in the input text is parsed in three steps:

- (1) Matching the phrase *pattern* against the clause in the text.
- (2) Validating in the context the relations specified by the phrase *presupposition*.
- (3) If both (1) and (2) are successful then instantiating the phrase *concept* using variable bindings computed in (1) and (2).

For example, consider the sentence:

S9: Fred wanted to marry Sheila, but she ducked the issue
for years. Finally he put her on the spot.

The figurative phrase is parsed relative to the context established by the first sentence. Assume that the lexicon contains a single phrase, described informally as:

pattern: Person1 put Person2 on the spot
presupposition: Person2 avoids making a certain tough decision
concept: Person1 prompts Person2 to make that decision

The steps in parsing the clause using this phrase are:

- (1) The pattern is matched successfully against the text. Consequently, Per-

son1 and Person2 are bound to Fred and Sheila respectively.

- (2) The presupposition associated with the pattern is validated in the context. After reading the first phrase the context contains two concepts: (a) Fred wants to marry Sheila, and (b) she avoids a decision. The presupposition matches the input.
- (3) Since both (1) and (2) are successful, then the pattern itself is instantiated, adding to the context:

Fred prompted Sheila to make up her mind.

Phrase *presupposition*, distinguished from phrase *concept*, is introduced in our representation, since it helps solve three problems: (a) in *disambiguation* it provides a discrimination condition for phrase selection, (b) in *generation* it determines if the phrase is applicable, and (c) in *acquisition* it allows the incorporation of the input context as part of the phrase.

4.2.2 Phrasal Lexicon

RINA uses a declarative phrasal lexicon, with *unification* [Kay79] as the grammatic mechanism. Below are some sample phrasal patterns.

P1: ?x <lay down> <the law>
P2: ?x throw <the book> <at ?y>

These patterns actually stand for the slot fillers given below:

P1: (subject ?x (class person))
 (verb (root lay) (modifier down))
 (object (determiner the) (noun law))

P2: (subject ?x (class person))
 (verb (root throw))
 (object ?z (marker at) (class person))
 (object (determiner the) (noun book))

This notation is described in greater detail in the previous chapter.

4.2.3 Phrase Acquisition through Generalization and Refinement

Phrases are acquired in a process of hypothesis formation and error correction. The program generates and refines hypotheses about both the linguistic pattern, and the conceptual meaning of phrases. For example, in acquiring the phrase *carry the water*, RINA first uses the phrase already existing in the lexicon, but it is too general a pattern and does not make sense in the context.

```
?x carry:verb ?z:phys-obj <for ?y>
```

Clearly, such a syntactic error stems from a conceptual error. Once corrected, the hypothesis is:

```
?x carry:verb <the water> <for ?y>
```

The meaning of a phrase is constructed by identifying salient features in the context. Such features are given in terms of scripts, relationships, plan/goal situations and emotions. For example, *carry the water* is given in terms of *agency* goal situation (?x executes a plan for ?x) on the background of *rivalry* relationship (?x and ?y are opponents). Only by detecting these elements in the context can the program learn the meaning of the phrase.

4.3 CONCEPTUAL REPRESENTATION

The key for phrase acquisition is appropriate conceptual representation, which accounts for various aspects of phrase meanings. Consider the phrase *to throw the book* in the following paragraph:

```
S2:   The famous mobster avoided prosecution for years.  
       Finally they threw the book at him for tax evasion.
```

We analyze here the components in the representation of this phrase.

4.3.1 Scripts

Basically, the figurative phrase depicts the *trial script* (shown in Section 2.4), which is given below:

-
- (a) The prosecutor says his arguments to the judge
 - (b) The defendant says his arguments to the judge
 - (c) The judge determines the outcome, either:
 - (1) to punish the defendant
 - (2) not to punish the defendant
-

Figure 4.1: Four events in *\$trial*

This script involves a Judge, a Defendant, and a Prosecutor, and it describes a sequence of events. Within the script, the phrase points to a single event, the decision to punish the defendant. However, this event presents only a rough approximation of the real meaning which requires further refinement.

- (a) The phrase may be applied in situations that are more general than the trial script itself. For example:

S10: When they caught him cheating in an exam
for the third time, the dean of the school
decided to **throw the book** at him.

Although the context does not contain the specific *trial script*, the *social authority* which relates the judge and the defendant exists also between the dean and John.

- (b) The phrase in S2 asserts not only that the mobster was punished by the judge, but also that a certain prosecution strategy was applied against him.

4.3.2 Specific Plans and Goals

In order to accommodate such knowledge, scripts incorporate specific planning situations. For example, in prosecuting a person, there are three options, a basic rule and two deviations:

- (a) Basically, for each law violation, assign a penalty as prescribed in the book.
- (b) However, in order to loosen a prescribed penalty, mitigating circumstances may be taken into account.
- (c) And on the other hand, in order to toughen a prescribed penalty, additional violations may be thrown in.

In S2 the phrase conveys the concept that the mobster is punished for tax evasion since they cannot prosecute him for his more serious crimes. It is the selection of this particular prosecution plan which is depicted by the phrase. The phrase representation is given below,

```

pattern      ?x:person  throw:verb  <the book>  <at ?y:person>
presupposition ($trial  (prosecution ?x) (defendant ?y))
concept      (act      (select-plan
                          (actor      prosecution)
                          (plan       (ulterior-crime
                                       (crime ?c)
                                       (crime-of ?y))))))
              (result (thwart-goal
                       (goal ?g)
                       (goal-of ?y)))

```

Figure 4.2: Representing *throw the book at*

where *ulterior-crime* is the third prosecution plan above.

4.3.3 Relationships

The *authority* relationship [Schank78, Carbonell79] is pervasive in phrase meanings, appearing in many domains: judge-defendant, teacher-student, employer-employee, parent-child, etc. The existence of *authority* creates certain expectations; if X presents an *authority* for Y, then:

- (a) X issues rules which Y has to follow.
- (b) Y is expected to follow these rules.

- (c) Y is expected to support goals of X.
- (d) X may punish Y if Y violates the rules in (a).
- (e) X cannot dictate actions of Y; X can only appeal to Y to act in a certain way.
- (f) X can delegate his authority to Z which becomes an authority for Y.

In S10, the dean of the school presents an authority for John. John violated the rules of the school and is punished by the dean. More phrases involving authority are given by the following examples.

S11: I thought that parking ticket was unfair
so I **took it up with** the judge.

S12: My boss wanted us to stay in the office until 9pm
every evening to finish the project on time.
Everybody was upset, but nobody **stood up to** the boss.

S13: Jenny's father **laid down the law**: no more late dates.

The representation of the phrase *take it up with*, for example, is given below:

pattern	?x:person	<take:verb up>		
			?z:problem	<with ?y:person>
presupposition	(authority (high ?y) (low ?x))			
concept	(act	(auth-appeal		
		(actor ?x)	(to ?y)	(object ?z))
	(purpose (act			
		(auth-decree		
		(actor ?y)	(to ?x)	(object ?z)))
		(result	(support-plan	(plan-of ?x))))

Figure 4.3: Representing *take it up with*

The underlying presupposition is an *authority* relationship between X and Y. The phrase implies that X appeals to Y so that Y will act in favor of X.

4.3.4 Abstract Planning Situations

General planning situations, such as *agency*, *agreement*, *goal-conflict* and *goal-concordance* [Wilensky83] are addressed in the examples below.

S1: The Democrats in the house carried the water
for Reagan in his tax-reform bill.

The phrase in S1 is described using both *rivalry* and *agency*. In contrast to expectations stemming from *rivalry*, the actor serves as an agent in executing his opponent's plans. The representation of the phrase is given below:

pattern	?x:person	carry:verb		
			<the water ?z:plan>	<for ?y:person>
presupposition	(rivalry	(actor1 ?x)	(actor2 ?y))	
concept	(agency	(agent ?x)	(plan ?z)	(plan-of ?y))

Figure 4.4: Representing *carry the water*

Many other phrases describe situations at the abstract goal/plan level. Consider S14:

S14: I planned to do my CS20 project with Fred.
I backed out of it when I heard that
he had flunked CS20 twice in the past.

Back out of depicts an agreed plan which is cancelled by one party in contradiction to expectations stemming from the agreement.

S15: John's strongest feature in arguing is his ability
to fall back on his quick wit.

Fall back on introduces a *recovery* of a goal through an alternative plan, in spite of a *failure* of the originally selected plan.

S16: My standing in the tennis club deteriorated since
I was bogged down with CS20 assignment the whole summer.

In bog down, a goal competition over the actor's time exists between a major goal (tennis) and a minor goal (CS20). The major goal fails due to the efforts invested in the minor goal.

4.3.5 Emotions and Attitudes

In text comprehension, emotions [Dyer83, Mueller85] and attitudes are accounted for in two ways: (a) they are generated by goal/planning situations, such as goal failure and goal achievement, and (b) they generate goals, and influence plan selection. Some examples of phrases involving emotions are given below. *Humiliation* is experienced by a person when other people achieve a goal which he fails to achieve. The phrase in S17 depicts humiliation which is caused when John reminds the speaker of his goal situation:

S17: I failed my CS20 class. My friend John rubbed my nose in it
by telling me that he got an A+.

Resentment is experienced by a person when a certain goal of his is not being satisfied. This goal situation causes the execution of plans by that person to deteriorate. The phrase in S18 depicts such an attitude:

S18: Since clients started to complain about John, his boss
asked him if he had a chip on his shoulder.

Embarrassment is experienced by a person when his plan failure is revealed to other people. The phrase in S19, depicts embarrassment which is caused when a person is prompted to make up his mind between several bad options.

S19: Ted Koppel put his guest on the spot when he asked him if he
was ready to denounce apartheid in South Africa.

In all the examples above, it is not the emotion itself which is conveyed by the phrase. Rather, the concept conveys a certain goal situation which causes that emotion. For example, in S20 (rub one's nose) a person does something which causes the speaker to experience *humiliation*.

4.4 LEARNING PHRASE MEANINGS

Consider the situation when a new phrase is first encountered by the program:

User: The Democrats in the house carried the water
for Reagan's tax-reform bill.

RINA: They moved water?

User: No. They carried the water for him.

RINA: They helped him pass the bill.

Three sources take part in forming the new concept, (a) the linguistic clues, (b) the context, and (c) the metaphor.

The Context: The context prior to reading the phrase includes two concepts:

- (a) Reagan has a goal of passing a law.
- (b) The Democrats are Reagan's rivals—they are expected to thwart his goals, his legislation in particular.

These concepts provide the phrase *presupposition* which specifies the context required for the application of the phrase.

The literal interpretation of *carried the water* as "moved water" does not make sense given the goal/plan situation in the context. As a result, RINA generates the literal interpretation and awaits confirmation from the user. If the user repeats the utterance or generates a negation, then RINA generates a number of utterances, based on the current context, in hypothesizing a novel phrase interpretation.

Since the action of moving water does not make sense literally, it is examined at the level of plans and goals: Moving water from location A to B is a low-level plan which supports other high-level plans (i.e., using the water in location B). Thus, at the goal/plan level, the phrase is perceived as: "they executed a low-level plan as his agents" (the *agency* is suggested by the prepositional phrase: *for his tax-reform bill*; i.e., they did an act *for* his goal). This is taken as the phrase *concept*.

The Constructed Meaning: The new phrase contains three parts:

- (a) The phrase *pattern* is extracted from the example sentence:

?x carry:verb <the water> <for ?y>

- (b) The phrase *presupposition* is extracted from the underlying context:

(rivalry (actor1 ?x) (actor2 ?y))

- (c) The phrase *concept* is taken from the metaphor:

(plan-agency (actor ?x) (plan ?z) (plan-of ?y))

Thus, the phrase means that in a rivalry situation, an opponent served as an agent in carrying out a plan.

4.5 CONCLUSIONS

The *phrasal* approach elevates language processing from interaction among single words to interaction among entire phrases. Although it increases substantially the size of the lexicon, this *chunking* simplifies the complexity of parsing since clauses in the text include fewer modules which interact in fewer ways. The phrasal approach does reduce the power of the program in handling non-standard uses of phrases. For example, consider the situation where a mobster kidnaps a judge, points the gun at him, and says: No funny book you could throw at me now would do you any good!*. Our current parser would certainly fail in matching the syntactic pattern and inferring the ironic meaning. The analysis of such a sentence would require that the program *associate* the two existing phrases, the general *throw something* and the figurative *throw the book*, and make inferences about the pun meant by the mobster. Such examples show that it is difficult to capture human behavior through a single parsing paradigm.

Phrase acquisition from context raises questions regarding the volume of knowledge required for language processing. A phrase such as *throw the book* requires highly specialized knowledge involving sentencing strategies in court. Now, this is only one figurative phrase out of many. Thus, in order to handle figurative phrases in general, a program must ultimately have access to all the knowledge of a socially mature person. Fortunately, learning makes this problem more tractable. In the process of phrase acquisition, phrase meaning is elevated from the specific domain in which the phrase has originated to a level of abstract goal situations. For example, once *throw the book* is understood as the act of *authority-decree*, then knowledge of the trial si-

* This example is attributed to an anonymous referee.

tuation no longer needs to be accessed. The phrase is well comprehended in other domains: my boss threw the book at me, his parents threw the book at him, her teacher threw the book at her, etc. *At that level, a finite number of goal situations can support the application of figurative phrases across a very large number of domains.*

Part II

Learning Phrases in Context

Chapter 5:

Learning a Hierarchy of Phrases

The previous part of the dissertation presented DHPL, the dynamic lexical structure. This part presents the learning algorithm itself: Chapter 5 describes the entire learning scheme, and the various steps taken at each situation. Chapter 6 focusses on one step in that scheme: how a phrase is acquired from a single example.

Since the lexicon is organized as a hierarchy, and not as a flat structure, phrases cannot be simply placed in the lexicon: they must be interconnected with other phrases in the hierarchy at the appropriate level of generality. Furthermore, since input examples are given in terms of specific phrases only, phrases must be propagated up and down the hierarchy, starting at the bottom level.

In this chapter we describe a learning algorithm, which is based on two existing machine-learning models: learning in a *version space* [Mitchell82], and learning by accumulating specific *episodes* in a dynamic memory [Kolodner84, Schank82]. The input required by the algorithm is a sequence of specific episodes, or training examples, from which lexical entries at various levels in the hierarchy are *generalized* and *specialized*.

5.1 INTRODUCTION

Traditionally, the lexicon has been viewed as a *flat* list of lexical entries. Recently, however [Jacobs85b, Langacker86], *hierarchy* has emerged as a method for organizing lexical information. Through the hierarchical representation, linguistic knowledge ranging from general grammar rules, to specific lexical entries can be covered uniformly. A significant advantage of this approach is the ability to cope with *incomplete knowledge*—a problem faced by any running linguistic model, as well as by humans.

Two aspects of this problem must be investigated: performance—parsing and generation—and acquisition. In a later chapter we show how the hierarchical lexicon facilitates performance even in conditions of incomplete knowledge: through the hierarchy, general phrases can fill in for missing specific ones. However, how is the lexicon generated in the first place? In this chapter we show how the hierarchy itself is constructed dynamically, as a by-product of parsing.

5.1.1 The Linguistic Phenomenon

Four scenarios are distinguished here, according to two parameters: (a) the contents of the program's lexicon, and (b) the clues given by the user's example. The entire sequence illustrates how the program upgrades its parsing ability as experiences are being accumulated.

(1) Neither the Entry Exists in the Lexicon, Nor is the Context Provided by the User: Initially, the lexicon contains no entry to cope with the new phrase to push around.

User: Throughout their childhood,
Frank used to push Corinne around.
RINA: He moved her in circles?

Although the words *push* and *around* exist in the lexicon, the entire phrase *push somebody around* is yet unknown. Neither does the context, established by the text (*throughout their childhood* does not reveal the essence of the characters' relationship), provide any clue. Therefore, the program cannot cope with the new phrase, and it uses the literal interpretation of the single words: *he moved her in circles?*

(2) **No Entry Exists, but an Explanatory Context is Given:** While no entry exists in the lexicon for `push around`, this time the context provided by the user is explanatory:

User: Frank, Corinne's elder brother, **pushed** her **around**.
He asked her to make his bed against her will.
RINA: Frank made Corinne carry out a job on his behalf.

Here, due to the context provided by the user, RINA forms an initial hypothesis. However, based on a single example, the hypothesis is not general enough (doing a job on behalf of a second person is not the essence of the meaning).

(3) **A Similar Entry Exists, and a Context is Given:** Next, RINA hears a similar phrase, `to boss around`.

User: Mary was hired on the same day as John, but she knew
the manager. Therefore, she used to **boss around**
her colleague, John.
RINA: Mary forced him to act.

At this point, RINA could identify the similarities between the two situations. Basically, RINA detected a common surface feature (the word `around`), and common semantic features (one person uses his power to impose his will on a second person). She constructs a generalization regarding the association between these features, and accordingly, she conveys a more general hypothesis to the user. This hypothesis may help the learner predict the behavior of other phrases involving the word `around`. However, since there are phrases involving `around` which do not have this meaning (e.g., `stick around`, `get around to it`, etc.), the acquired generalized phrase must carry discriminating conditions.

(4) **No Context is Given, but a General Entry Exists:** Next, RINA hears a third related phrase, `to order around`:

User: Doug, Linda's younger brother,
told her to stop **ordering** him **around**.
RINA: Doug asked Linda to stop forcing him to act?

Like the first scenario, neither: (a), the phrase `order somebody around` is encoded explicitly in the lexicon, nor (b), any explanatory context is provided by the user.

However, in absence of specific knowledge for parsing the new phrase, RINA applies here the general phrase acquired previously. RINA's hypothesis is not guaranteed to be correct, yet it enables the formation of a hypothesis regarding the unknown phrase.

Notice how learning improved the performance of the program.

- *Before* learning, RINA was not able to parse `push around` (without an explanatory context).
- *After* learning, even in conditions of minimal context, RINA could parse sentences which include, not only the phrase itself, but an entire set of similar phrases.

5.1.2 The Issues

Five issues must be addressed in acquiring such a hierarchical lexicon.

Determining Scope and Variability: The syntactic pattern of a new phrase is extracted initially from a single sentence.

Frank used to push Corinne around.

From that sentence, the program must determine scope and the variability of the pattern. However, the unknown pattern could take on many forms.

```
Frank pushed Corinne
X:person  push:verb  Y:person
X:person  push:verb  Y:person  around
X:person  use:verb  to push:verb  Y:person  around
```

Which one of the forms above is appropriate as the phrase pattern?

Forming the Meaning: The meaning of the phrase is extracted from the context. However, the context includes many concepts, some appropriate, and some inappropriate for meaning formation. Thus, based on the given context of *Frank and Corinne*, what is the meaning of `push around`?

Generalization: From the first example, RINA extracted a narrow meaning:

RINA: Frank made Corinne carry out a job on his behalf.

By hearing a second example, RINA was able to generalize the meaning:

RINA: Mary forced John to act against his will.

This meaning pertains to general interpersonal relationships (*power, authority*) and can cover more situations in which the phrase may appear. (1) How can a generalization be formed across two apparently distinct situations (family feud vs. problems at the working place)? (2) How far should generalization be pursued without causing *overgeneralizing* (e.g., here is an overgeneralization: Frank did something negative to Corinne)?

Specialization: A new phrase order around is encountered without an explanatory context. Yet, although a specific lexical entry for that phrase does not exist, the phrase can be analyzed by a generalized phrase. How is generalized knowledge applied in coping with gaps in specific knowledge?

Phrase Disambiguation: Even when all the necessary information exist in the lexicon, parsing is not without problems. Consider the following pair of sentences:

Mary and John were ice skating in circles.

He was pushing her around.

John acted bossy with Mary. He was always pushing her around.

Since two phrases exist for push around, the program must select the appropriate one in each case. Individual lexical entries must provide the sufficient conditions for phrase discrimination.

5.1.3 The Approach

The *phrasal lexicon* [Becker75, Wilensky81, Fillmore87] contains not only single words (e.g., around), rather it contains entire phrasal entries (e.g., to get around to it, to stick around, around the clock, around 5pm). The phrasal approach has proved effective in parsing [Wilks75] and in generation [Jacobs85b]. Yet the basic dilemma regarding inclusion of phrases in the lexicon is unresolved. Which linguistic elements belong in the phrasal lexicon, and which elements do not? On the one hand, *productive* phrases such as John gave Mary the spoon, or Mary went to school, should not reside in the lexicon, lest the lexicon will get oversized. On the other hand, *non-productive* phrases such as the big apple, or to throw the book at somebody must reside in the lexicon. Their meanings cannot be derived

from their constituents. However, there is a gray area which includes phrases such as *stick around*, *hang around*, and *sit around*, for whom the inclusion question is unclear. These phrases are non-productive, since their meanings cannot be simply derived from the single words. Thus, they must be included. However, they all share semantic and syntactic features, thus they should not all be included as separate entries. Therefore in our model, such groups of phrases are clustered into *generalized* phrases, which represent their shared features. Such generalized phrases serve in predicting meanings of similar combinations even before they have been encountered.

The hierarchical representation facilitates learning as a continuous process of knowledge refinement. Two approaches have been integrated in our model. Using Mitchell's method [Mitchell82] we view the lexicon as a *version space* of rules. Since phrases are organized in a hierarchy by generalization, then both parsing and learning require determining location of phrases in that hierarchy. Kolodner [Kolodner84], on the other hand, has shown how learning is accomplished by organizing episodes in a dynamic memory. In our model, the lexical hierarchy itself is dynamic and nodes at all levels of generality are updated by receiving instance episodes at the bottom of the hierarchy.

5.2 REPRESENTING THE CONTEXT

The context is represented as a goal-plan situation at two levels: specific planning, and abstract planning.

Specific Planning Knowledge: Planning knowledge in each domain consists of goals—and acts which implement those goals. For example, consider the paragraph:

Frank needed to clean up his room.
Corinne made his bed for him.

The events underlying the sentence are given in the following *plan box* [Schank77].

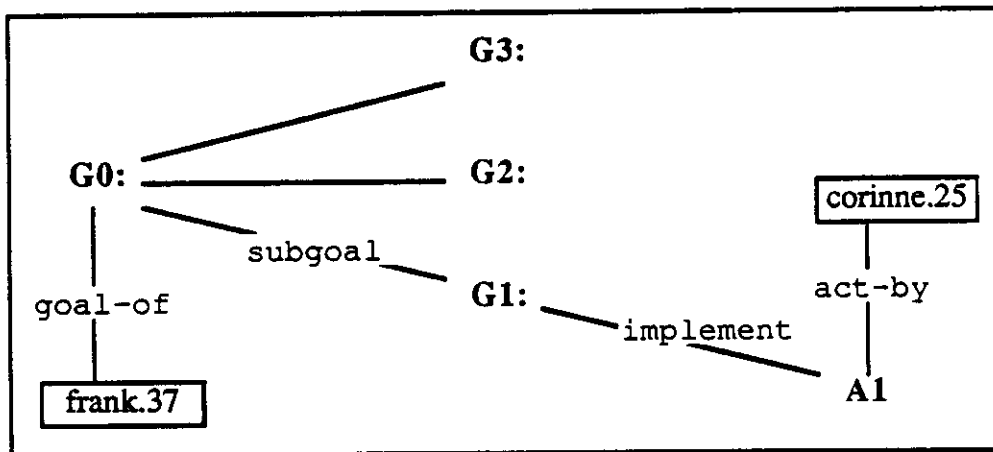


Figure 5.1: The Specific Plan Box

Making the bed (G1), wiping the floor (G2), clearing the desk (G3), are all *subgoals* for cleaning one's room (G0). In this case, although G0 is a goal of Frank, Act A1 which *implements* G1 is executed by Corinne.

Abstract Planning Knowledge: Many specific plan boxes are required in covering planning situations across domains. In contrast, a relatively small set of abstract planning structures [Wilensky83] (e.g., *goal-conflict*, *goal-competition*, *goal-concordance*, *plan agency*, etc.), and relations (*authority*, *power*, *friendship*) can cover situations across all domains. In our example, due to the *power* relationship between characters X and Y, X causes Y execute an act A on X's behalf. This is shown in figure 5.2 below.

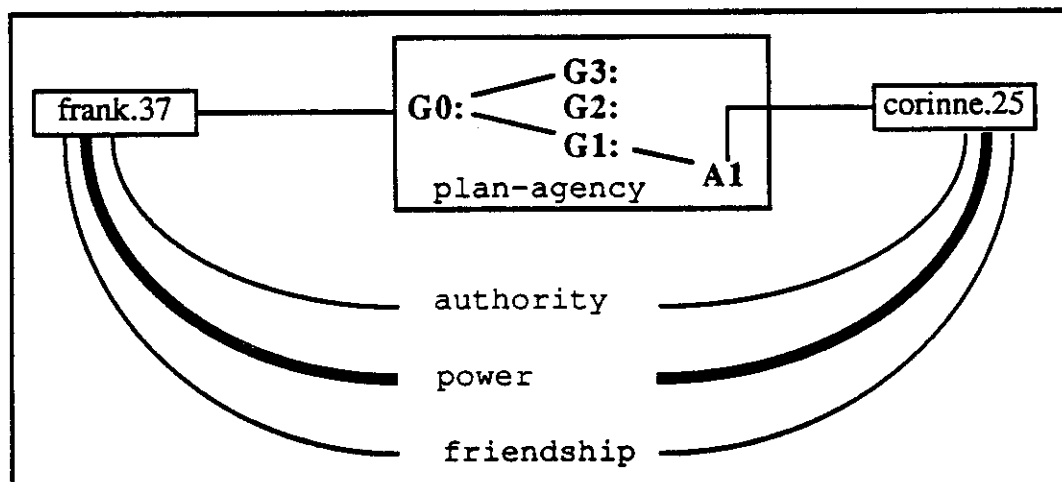


Figure 5.2: An Abstract Goal Situation

Corinne serves as a *plan agent* in executing a plan for Frank. Her motivation is neither (a) an *authority* relationship (Frank is not an authority for Corinne), nor (b) a friendship with Frank, but (c) the *power* relationship (due to his Frank's age). The abstract planning elements are used here to *explain* the planning situation. Planning knowledge at these two levels provide the semantics for lexical representation.

5.3 THE LEXICON

As explained in Chapter 3, the phrasal lexicon is specified in two ways: by the *contents* of single lexical entries, and by the *structure* of the entire lexicon.

Single-Phrase Representation: A lexical entry, or a *phrase*, is a triple associating a linguistic *pattern* with its *concept* and a *presupposition*. For example compare the following sentences:

- (1) John sat three days working on his thesis.
- (2) John sat around three days working on his thesis.

What is the difference between their meanings, and what is the value added by the modifier *around*? The phrase *to sit around* is represented as a triple:

pattern:	X:person sit around
presupposition:	X dedicates time to an act P
concept:	the execution of P is a waste of time

Sentence (2), then, is parsed in three steps using the lexical phrase:

- (1) The *pattern* is matched successfully against the text. Consequently, X is bound to the person called John.
- (2) The *presupposition* is unified with the current context. In this case the variable P is taken by the context as the Ph.D work.
- (3) Since both (1) and (2) are successful, then the pattern itself is instantiated, adding to the context: *working on the Ph.D wasted John's time* (which disabled execution of other acts by John).

Generalized Phrases: However, the representation given above for *hang around* is not satisfactory when considering two other similar phrases:

John used to **stick around** for hours.
John **hung around**, waiting for his girlfriend.

The lexicon cannot simply enumerate all word combinations. There must be a more general entry to account for all the phrases of the form "verb around" which convey this particular sense.

pattern:	X:person V:act around
-----------------	-----------------------

presupposition: V is an act of staying in location L
for a period of time
concept: staying in L disables execution of other acts by X

This entry captures an entire set of such phrases. The *presupposition* part is used to discriminate phrases such as hang around, and stick around from phrases such as push somebody around and boss somebody around, by detecting necessary conditions in the context.

The Global Hierarchy: A hierarchy is defined, to accommodate for phrases at all levels of generality.

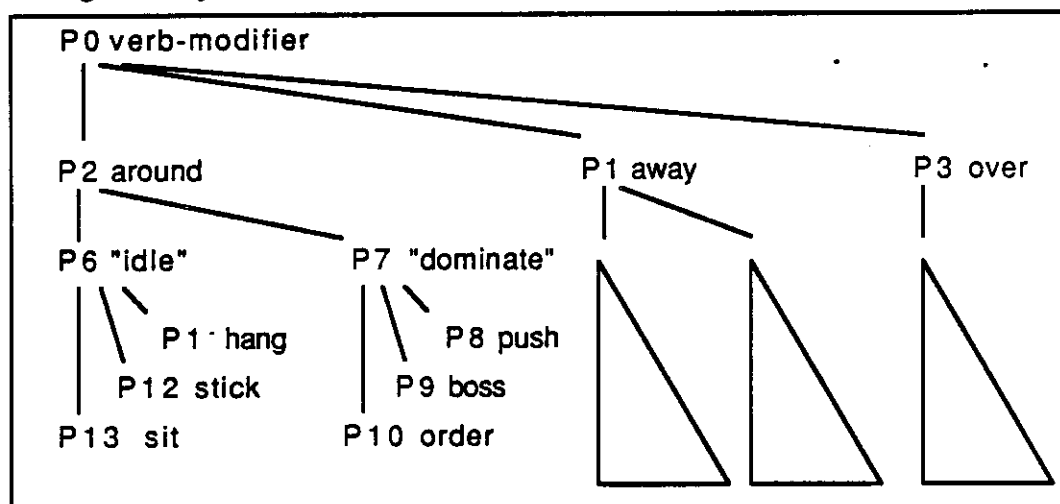


Figure 5.3: The Hierarchy for Verb-Modifiers

(The names of the nodes in this scheme are for reference only.) Phrases in this hierarchy are all given uniformly as *pattern-concept-presupposition* triples. For example, P13 (sit) is the phrase for sit around, and P6 ("idle") is the phrase encompassing a set of v around phrases. Specific phrases, traditionally called "lexical rules", reside near the bottom. General phrase, or "grammar rules", reside at the top. P2 (around), for example, maintains general knowledge about the behavior of that modifier.

5.4 THE ALGORITHM

The learning algorithm accepts as input a sequence of episodes, from which it generates a subtree of phrases. The hierarchy before and after learning is given schematically below, where subtree T4 is the element added to the lexicon by learning.

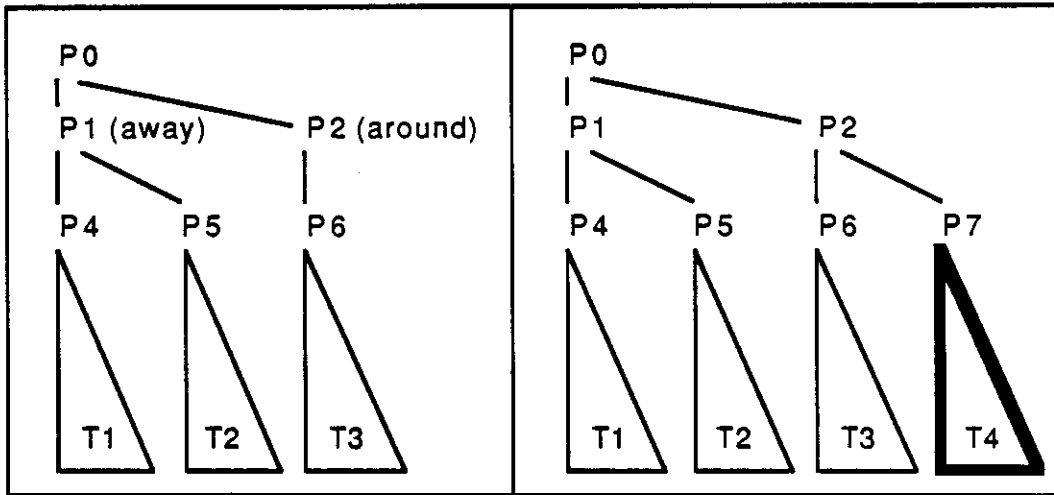


Figure 5.4: The Hierarchy Before and After

The process itself is shown schematically through a sequence of snapshots.

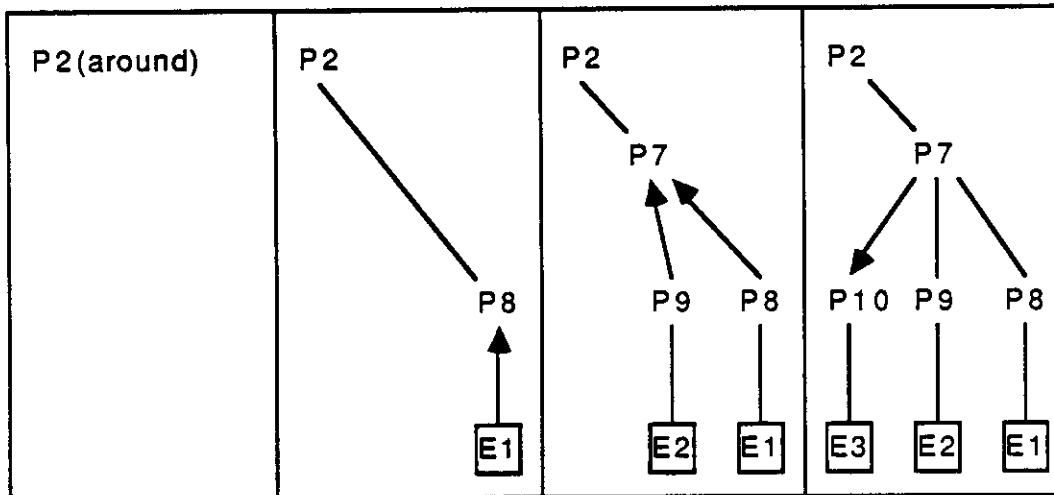


Figure 5.5: Four Snapshots in the Sequence

- (1) Initially, the subtree includes only a the general node P2 (which stands for the modifier *around*).
- (2) Node P8 is *extracted* from episode E1.
- (3) Node P9 is extracted from episode E2. However, P7 is *generalized* by detecting the similarities between E1 and E2,
- (4) E3 is not informative enough for extraction of a node. However, node P10 can be *specialized* from the general node P7.

Thus, a subtree is added under P2. The specifications of the algorithm are given below. The given information is:

- (1) An initial hierarchy. This hierarchy is not sufficiently developed, so parsing of certain sentences might fail.
- (2) A sequence of episodes. An episode is a paragraph embedding a new phrase in an explanatory context.
- (3) The semantics of the context. Semantic knowledge is given in terms of planning knowledge.

The objective is to refine the initial hierarchy. By adding on additional nodes, or by specializing and generalizing existing nodes, the new hierarchy should facilitate parsing of sentences on which the initial hierarchy has failed. The three basic steps of the algorithm are explained in regard to the sequence of scenarios in section 1.1.

5.4.1 Extracting a Phrase from a Single Example

Two simultaneous actions are involved in acquiring an unknown phrase from a training example (*Frank pushed his younger sister around*).

- (1) The pattern is extracted from the input sentence.
- (2) The meaning (both *concept* and *presupposition*) is extracted from the given context.

(1) Forming the Pattern

Two problems arise in pattern formation: determining the scope and determining the variability of the new pattern.

Determining the scope: Since the input sentence includes not only the bare phrase itself, but some additional elements, it is necessary to determine which elements in the sentence should be included in the pattern and which ones should be excluded. For example, the sentence under consideration contains two modifiers:

Throughout their childhood, Frank used to push Corinne around

While *throughout their childhood* is excluded, *around* is taken as a part of the new phrase. The reason is that *throughout their childhood* can be interpreted successfully by the parser in any context. It does not cause a parsing failure. However, *around* does cause a parsing failure: "Frank moved Corinne around something"? Thus, the pattern is scoped as follows:

X:person push Y:person around

Determining variability: In converting an instance sentence into a general pattern, each element in the new pattern either becomes a variable, or it remains as a literal. In the conversion above, both *Frank* and *Corinne* were taken as generalized references to persons, X and Y. Accordingly, it appears that the rule was:

References to persons and to objects are converted into variables.

However, this rule is refuted by examples such as:

- (1) Admiral Nelson went to Davey Jones' locker.
- (2) He kicked the bucket in Trafalgar.

In these idioms, the marked references are taken as literals and not as variables, in violation of the rule above. Thus, the rule must be refined:

References which can be resolved in the context are converted into variables. References which cannot be resolved in the context are kept as literals.

Further strategies for pattern formation are described elsewhere [Zernik85a].

(2) Forming the Meaning

Initially, the program focuses at the level of specific plans.

G (cleaning the room) is goal of X (Frank).

P (making the bed) is a plan for G.

P is carried out by Y (Corinne).

The meaning captured at this level is: X carried out a job on Y's behalf. Consequently, the acquired phrase is:

pattern:	X:person	<push around>	Y:person
presupposition	P is a plan for G		
concept:	Y is a plan agent in executing plan P for X		

P8: The Specific Phrase for *push around*

However, many situations in which *push around* may be applied, are not included in P8's definition. Thus, P8 must be generalized.

5.4.2 Generalizing from Two Examples

The second scenario, Mary bossed her colleague around, does not share the specific features with the first example. **Syntactically:** the verb is *boss* and not *push*. **Semantically:** John did not carry out a job in Mary's behalf. Thus both the pattern and the meaning must be generalized.

The pattern is generalized as the verb becomes a variable. The meaning is generalized when specific planning elements are replaced by *abstract planning* structures which are shared by both episodes: Mary used her power (and not an authority), and Frank used his power to make Corinne act against her will. Therefore, the resulting phrases is:

pattern:	X:person	<V:act around>	Y:person
presupposition:	X presents a power for Y		
concept:	X causes Y to act against X's will		

P7: The Generalized Phrase for "dominating"

Notice that although the general phrase P7 was generated, the specific phrases P8 and P9 have not been eliminated. Specific information is maintained even though it is subsumed by more general information.

5.4.3 Specializing a General Phrase

A specific phrase does not exist for `order around` in the fourth scenario. However, the sentence is parsed by using the generalized phrase P7. P7 is matched successfully in the episode E3, since:

- (1) The syntactic pattern of P7 matches the input sentence (the form is `x Verbed y around`).
- (2) The semantic presupposition matches the input context (`X presents a power to Y`).

The specific phrase P10 (for `order around`) is added on to the lexicon, although it is subsumed by P7. For one thing, specific phrases enrich the program's vocabulary. In text generation they enable production of sentences which include specific variants of generalized phrases.

5.5 PREVIOUS WORK IN LANGUAGE ACQUISITION

Two previous models of language acquisition are related to our work. Granger's program FOUL-UP [Granger77], which acquired word meanings, and Pinker's model [Pinker84] which acquired language syntax.

FOUL-UP was devised as a mechanism for extending SAM's [Cullingford78] lexicon while performing conceptual analysis. SAM constructed meanings for English sentences by a two-step cycle: (a) select a script, and (b) instantiate the script. For example the following text:

John's baby caught a cold. He called up his doctor,
and made an appointment for the next morning.
The nurse took his temperature, and it was 106 degrees.
She realized immediately that he needed to be ragged.

involves the *\$clinic* script, which stands for a chain of events associated with a visit to the family doctor. Thus, the combination of child-being-sick followed by call-up-a-doctor causes the selection of *\$clinic*. Once this script is selected, references and events can be readily resolved. Made an appointment for example is taken as a standard *\$clinic* event. The nurse is bound to a designated role-holder in the script, and take his temperature also matches an anticipated action while in the clinic. The reference his (in his temperature) is bound to John's baby (and not to John) since the baby holds the role of the *patient* in *\$clinic*. FOUL-UP is activated by SAM at the point that the word *ragged* is encountered, a word which does not appear in SAM's lexicon. By knowing the rest of the possible events and the possible outcomes of *\$clinic*, FOUL-UP can predict that to be *ragged* means to taken to the hospital. FOUL-UP learned lexical entries assuming that the new single word appeared in the context of a script, and that the meaning of the word could be drawn from that script. FOUL-UP did not handle syntax acquisition, nor multi-word phrases.

Pinker [Pinker84] modeled child language acquisition, covering phenomena ranging from basic phrase structure identification to mastering phrase interaction—the issue being addressed here. Phrase interaction, or *complementation*, is restricted by two substantive constraints:

- (a) The argument which might be missing in the complement is always the subject (e.g.: in *John persuaded Mary to leave*, the subject of the verb *leave* is not explicit in the text).
- (b) The missing argument is *equated* with either one of:
 - (1) an object of the controlling verb (*persuade* in the example above), if an object exists (which is the case in the example),
 - (2) The subject of the controlling verb if the object does not exist (as in *John asked to go*).

This pattern of behavior holds for virtually all complement-taking English verbs. Nonetheless there are some exceptions such as *promise* in *John promised Mary to go*, or *make* in *John made Mary a fine husband*. Such exceptions, although small in number, are significant since they could be used for testing the power of language-learning theories. If such exceptions did not exist, the learning of verb complementa-

tion would be trivial. Indeed this behavior (of *promise*) accounts for errors in children's language, as recorded by Carol Chomsky [Chomsky69] (as in Jenny's father promised her to go to Disneyland meaning he promised her that she would go there). At this point, an example in context (in which it is clear that the father is supposed to go) can correct such a hypothesis.

RINA draws from both these approaches. RINA's acquisition of lexical items concerns the mapping between syntactic patterns and their semantic concepts. However, Granger's model was restricted to single words and simple script-like contexts. RINA also addresses issues such as control and complementation. However, in our view, behavior of verbs reflects their semantics, and it is not just an arbitrary parameter which needs to be acquired to distinguish *promise* from *persuade*. Concepts representing such verbs must account for two facts. The verb means that (1) one party conveyed fact to a second party, and (2) each individual verb denotes a different speech act performed by conveying that fact. Thus learning the special syntax of *promise* is a by-product of understanding what *promise* means.

Pinker presupposes the existence of certain *innate* learning procedures. In his model, each language feature is accounted for by a custom-tailored procedure. For example, in learning the control aspect of complement-taking verbs, Pinker assumes this pre-existing rule (Rule C3):

Add to the lexical entry of the complement-taking predicate the equation X-COMP's SUBJ = (FUNCTION) where (FUNCTION) is the grammatical function annotated to the matrix argument that is coindexed with the missing complement subject in the contextually inferred semantic representation. ([Pinker84] pp. 213).

Pinker has chosen *Lexical-Functional Grammar* (LFG) [Bresnan82a] as his linguistic framework. While LFG intends to denote a variety of linguistic phenomena, the rule C3 talks *about* LFG notation, but it certainly is not expounded in LFG terms. Moreover, if a language-learning program requires such a specific *procedural* rule for each language feature, then the purpose of modeling learning in the first place is defeated. Learning must be handled by a general *declarative* mechanism. In fact, the same *unification* mechanism which accounts for parsing and generation, should account also for learning.

5.6 CONCLUSIONS

Although the lexicon includes both specific and general phrases, the algorithm which constructs the hierarchy requires as input only specific episodes. This reflects human learning behavior. When communicating, people do not exchange syntax and semantics in form of rules, rather they communicate in terms of specific examples in context.

Accordingly we have shown a learning algorithm which augments a hierarchical lexicon as follows:

- (1) A partial initial lexicon is **given**.
- (2) **The input** is a sequence of specific examples.
- (3) **The result** is an augmented lexicon.

The initial lexicon can be augmented by new specific phrases. It can also be enhanced by generalization and specialization of existing entries. As a result of learning, the system can parse sentences it initially could not parse.

Chapter 6: Learning Idioms – With and Without Explanation

The previous chapter presented the entire learning scheme, in which the basic task was learning phrases from examples. Two cases were described. On the one hand, when *multiple* episodes exist to exemplify a phrase, the phrase is generalized from the set of input examples by detecting structural similarities. On the other hand, when a phrase is first encountered, only a single example exists, the methods above fail. We must address alternative methods, which are not based on comparing similarities among multiple examples.

Modeling learning in any domain can be pursued in two directions: either by finding the domain-specific heuristics, or by applying general machine-learning methods. In the linguistic domain, programs such as FOULUP and CHILD [Granger77, Selfridge80] have used specific heuristics, regardless of the general learning methodology; other programs [Reeker76, Anderson77, Langley82, DeJong86] have focussed on the general learning methodology. In our particular task, learning idioms from examples, the evaluation of general learning models is very appealing, since such methods have been studied extensively, and may offer ready solutions.

Language is a special domain in regard to learning. Consider for example behavior of idioms. By definition, idiosyncratic properties are not systematic, and are not predictable. Thus, how do people learn such properties from examples? What is the general

learning model then, which accounts for idiom acquisition?

6.1 INTRODUCTION

Although idioms are pervasive in human communication, especially in spoken language, their irregular behavior has not been investigated systematically. Idioms are interesting since they provide singular points by which linguistic theories and machine learning theories should be evaluated. Here is a puzzling phenomenon [Katz63, Chafe68, Dong71, Jacobs85a], which has not been explained so far:

- (1) In 1977, Israel and Egypt resolved their long conflict.
The hatchet had been buried.
- (2) Finally the patient succumbed.
The bucket had been kicked.

These two phrases (kick the bucket and bury the hatchet) behave differently with respect to the passive voice. While the first paragraph is generally acceptable to native English speakers, the second one sounds awkward. This behavior is surprising (and unpredictable in computer-program terms) since this pair of phrases are structurally similar. We investigate this idiosyncratic property in terms of its acquisition. This behavior is significant because it sheds light on the otherwise hidden language-acquisition processes, and illuminate the role of *metaphor*.

Metaphors provide learners with the information necessary for explaining the nature of idioms. When encountering an idiomatic phrase, people seek for the clues which would make it less arbitrary. However, a metaphor might be obscure, at least from the view of a learner, and thus an explanation cannot always be constructed. Regarding this problem, the questions we consider are: (a) can people learn an idiom even when its metaphor is not accessible? (what is the metaphor underlying kick the bucket?) (b) what is the impact of the metaphor on the use of idioms?

6.1.1 The Passive-Voice Anomaly

Consider the two phrases introduced in sentences (1) and (2). Why can the phrase kick the bucket not take the passive voice? Are there other such phrases? In fact, there are: put one's foot down, for example, cannot appear in the passive (his foot was put down does not convey the meaning of the idiom). How do language

learners predict for each new phrase, its behavior, whether it takes the passive voice or not? There must be a rule for supporting this prediction.

Traditionally, linguistic systems [Fraser70] accounted for this phenomenon by including arbitrary syntactic restrictions in the lexicon. Accordingly, lexical entries included explicit clauses to inhibit the passive voice.

pattern: kick the bucket
passive: **not possible**

pattern: bury the hatchet
passive: **possible**

We argue against the inclusion of such arbitrary structural restrictions, and seek a more substantial solution, in which *metaphor* accounts for idiom behavior.

Compare the metaphors underlying our pair of idioms. On the one hand, in *bury the hatchet*, learners assume that *hatchet* stands for a generic war implement, whose disposal ends a war (similarly, the poet buried her pen could mean that she stopped writing poetry). Some learners can imagine a remote culture in which burying a hatchet is a ceremonial act in signing a peace treaty. On the other hand, no such metaphor could be found for *kick the bucket*. Learners cannot come up with an explanation as to why the words *kick* and *bucket* really mean "to die".

Discourse analysis concerns *why* people phrase utterances the way they do. Sandy Thompson [Thompson87] gives a detailed account of people's use of the passive voice. She concludes that the passive voice is not just an arbitrary feature, but it is a device for accomplishing certain communication goals, such as placing the discourse topic at the beginning of the sentence. In light of this fact, consider the semantic structure of our idioms. Although the reference *the hatchet* does not stand for any concrete knife, it symbolically refers to the enablement of a fight between two parties. In contrast, *the bucket* is merely a literal. It does not refer to any object in the context. In the lexicon this difference is denoted as follows:

(3) X:person bury:verb <Y:the hatchet>

(4) X:person kick:verb <the bucket>

The marked object in (4) is merely a literal. The marked object in (3) also specifies the surface appearance of the reference. But in addition, it has a variable which can be bound to a concept. Accordingly, a discourse structure can be conceived, in which Y,

the concept associated with the hatchet, is the topic (as I tried to show in example (1) above), and consequently a communication goal exists for a speaker to apply the passive voice. However, no context can possibly be conceived in which the bucket refers to a topic—since it is merely a literal.

In conclusion, it turns out that the question itself, "*Can a phrase take the passive voice?*" is wrong. The correct question should be: "*Is there any discourse structure, in which the phrase should appear in the passive?*". The phenomenon is interesting in terms of machine learning, since it enables us to examine learning concepts with and without explanation, and to show how the availability of explanation (or a metaphor) makes a difference in applying the acquired concepts.

6.1.2 The Program Behavior

The behavior above is explained in terms of the program RINA. Here RINA encounters the yet unknown phrase, *bury the hatchet*.

User: In 1977 Israel and Egypt buried the hatchet.
RINA: They placed a hatchet under the ground?

In this scenario, since the figurative phrase does not exist in the program's lexicon, RINA resorts to the use of the single words *bury* and *hatchet*, thus forming a literal interpretation. Next, the user provides the program with a *clarification*:

User: Israel and Egypt were involved in a long conflict.
 In 1977 they signed a peace agreement.
RINA: They buried the hatchet; they terminated the conflict.

By this clarification, RINA is able to form a hypothesis involving the elements in the situation. Later on, RINA encounters the same phrase in a different situation:

User: Doug buried the hatchet.
RINA: He terminated a dispute with his wife.

In this case, RINA was able to parse the sentence. Thus, by receiving examples in context, the program improved its parsing ability. *Before* learning, RINA failed in parsing the phrase; *after* learning, RINA managed to parse the phrase.

6.1.3 Issues in Idiom Acquisition

Three issues must be resolved in learning idioms from examples.

Extracting the Pattern from the Sentence: The syntactic pattern of the new phrase is extracted from the given sentence. However, since the sentence does not consist only of the phrase itself, the program must determine the scope and the variability of the new pattern. There are many ways the pattern can be extracted:

- | | |
|----------------------------|--------------------------------|
| X buried the hatchet | - not generalizing the verb |
| X bury Z:phys-object | - allowing any physical object |
| In 1977 X bury the hatchet | - requiring the complement |
| X and Y bury the hatchet | - requiring conjunction |

Which elements belong in the pattern, and to what extent should they be generalized?

Extracting the Meaning from the Context: The meaning of the new phrase is extracted from the context. However the context contains only a specific episode. Does bury the hatchet pertain only to "signing a treaty" or can it be applied in other situations (such as a family dispute) by generalizing the meaning?

Forming an Explanation: We have described independently acquisition of two separate entities: the pattern and the meaning. However, is their association arbitrary? Is there any significance to the words of the pattern in relation to the meaning? Would a different wording, for instance they buried the poodle, produce the same learning behavior?

6.2 THREE MACHINE-LEARNING PARADIGMS

What is the method by which a model can *predict* linguistic properties of idioms (do/do-not take the passive voice) as well as their meanings, by being given a small number of examples? Three machine learning paradigms are considered.

In **learning by rote**, an idiom is copied as a chunk. However, this treatment is unacceptable since: (a) idioms must be *generalized* semantically to be applicable in a variety of situations, different from the original one, (b) as shown by examples (1) and (2), idioms, possess their own internal grammar [Fillmore87], which must also be

acquired. Thus idioms cannot be acquired merely as "extended words".

In **similarity-based learning (SBL)**, by being given a sufficiently large ensemble of examples, the model acquires surface features which are shared by a number of instances. There is no justification as to *why* a feature is acquired. This approach raises three problems: (a) humans are able to acquire idioms from few, or even from a single example, (b) humans do not require negative examples (e.g., "the bucket was kicked is incorrect"), and (c) humans do not acquire spurious features. Imagine two coincidental instances such as In 1977, X buried the hatchet, and in 1977, Y buried the hatchet. Humans are not thrown off by this cooccurrence, while an SBL model would assume that In 1977 is a mandatory part of the phrase. Similarly, the appearance of a phrase in the past tense could be taken as a mandatory property by an SBL model.

Explanation-based learning (EBL) remedies these problems by acquiring only features whose appearance can be justified. Spurious similarities are thus ruled out, and significant features can be acquired from a single example. However, this method requires existence of perfect explanatory information, whereas humans are required to learn certain phenomena even without explanatory information. For example, from the learner's point of view, there is no explanation (*metaphor*) for the idiom kick the bucket. What is then the reason that the sequence of words kick, the, and bucket means "to die"? Unfortunately—for language learners—many such linguistic phenomena appear arbitrary.

In our model, explanation is a by-product of hypothesis formation, and accordingly, the existence of a metaphor determines the quality of the explanation. Thus, three questions must be answered:

- How is an explanation constructed for an idiom?
- To what extent can idioms be explained?
- How does explanation effect idiom application in speech?

So far, only Pazzani [Pazzani86] and Lebowitz [Lebowitz86] have suggested models for learning in varying states of explanatory knowledge.

6.3 KNOWLEDGE REPRESENTATION

Explanation-based learning depends critically on contextual knowledge and its representation. The context in our model is given in terms of two types of planning knowledge: *abstract*, and *specific* planning situations.

Abstract Planning Knowledge: Abstract goal situations [Wilensky83], such as *goal-conflict*, *goal-competition*, *goal-concordance*, etc., apply across many domains. In particular consider the situation under analysis:

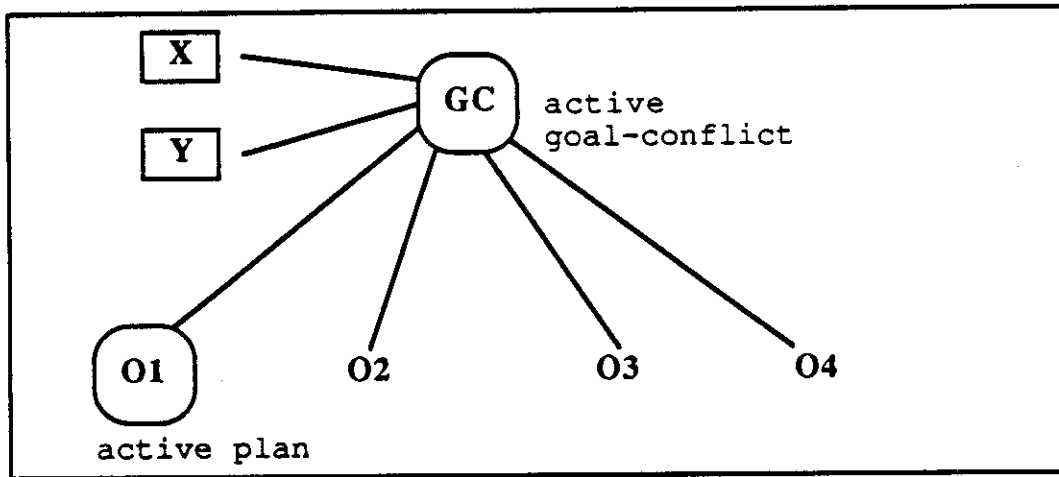


Figure 6.1: The Conflict Situation and Four Options

Two parties, X and Y are related by a *goal competition* situation (GC). There are four options for this situation to develop: (O1) X and Y fight, (O2) X and Y negotiate, (O3) X give up, (O4) Y give up. In our case, X and Y have been engaged in a fight (option O1). The sentence "they signed a peace treaty" implies that the parties have abandoned O1 for O2.

Specific Planning Knowledge: Specific goals in a certain domain can be accomplished by specific plans. For example, consider the following set of specific plans, all involving fights at various situations.

	name	party	tool	goal
1	dog-fight	aircraft	cannon	shoot down
2	nation-fight	nation	person	kill
3	family-dispute	person	dish	intimidate
4	street-fight	person	knife	inflict wound

Figure 6.2: Specific Plan Boxes

These plans encode conditions and alternative options for fights between two parties, according to the type of the fight. For example, in a domestic fight, the combatants are supposed to attack each other verbally, or maybe throw dishes. In a fight between nations, the parties may use people and weapons in order to kill people. In a street fight, the parties typically use fists or knives in order to wound each other. Maintaining specific plans in many domains is a difficult problem in knowledge representation. *Here we assume that all knowledge required for learning is given.*

The Metaphor: Two domains, the *primary* domain, and the *metaphor*, are brought to bear in understanding the following sentence (assuming that the idiom is yet unknown):

After a long conflict, Israel and Egypt buried the hatchet.

On the one hand, there is knowledge of the political situation between Israel and Egypt—a conflict which is being resolved by negotiations. The phrase itself has been applied in this *primary* domain; by understanding the sentence, a reader augments his/her knowledge of this domain. On the other hand, there is knowledge of violent fights in general, and the fact that a knife is a tool in a street fight in particular. This domain is *not* under focus in reading the sentence above, and it must deliberately be

accessed. This domain is called the *metaphor*. It is the structural similarities between these two domains that facilitate learning of the new idiom.

Lakoff and Johnson [Lakoff80] have investigated the role of metaphor in human thought. In their analysis, complex situations are understood through mappings to other well-known situations. So, for example, due to the existence of the *war metaphor*, an argument between two people can be talked of metaphorically as a war. The thrust of Lakoff and Johnson's effort was in showing how a large number of novel situations can be understood via a set of such mappings.

In contrast to our approach [Schank77, Dyer83, Wilensky83, Gentner83], in which analysis is based on structural similarities of the *underlying representation*, in Lakoff and Johnson's model, the mappings between pairs of situations are *direct*. This aspect is problematic computationally, since it requires the proliferation of metaphoric connections. Moreover, how is a mapping in that model constructed in the first place?

6.4 THE ALGORITHM

A *phrase* is an *association* of a syntactic *pattern* with a conceptual *meaning* [Wilensky80]. Accordingly, learning is defined as formation of pattern-concept *associations* (unlike other systems [Mitchell86, DeJong86, Lebowitz86] which acquire concepts in general). The learning algorithm is *provided* with two knowledge sources: (a) linguistic knowledge and (b) world knowledge.

The lexicon: A partial lexicon is assumed to be given, including the single words and the phrases involved in the input text—except for the phrase under construction.

World knowledge: The representation of the explanatory context is assumed accessible to the program. This includes knowledge about the specific domain as well as planning knowledge in general.

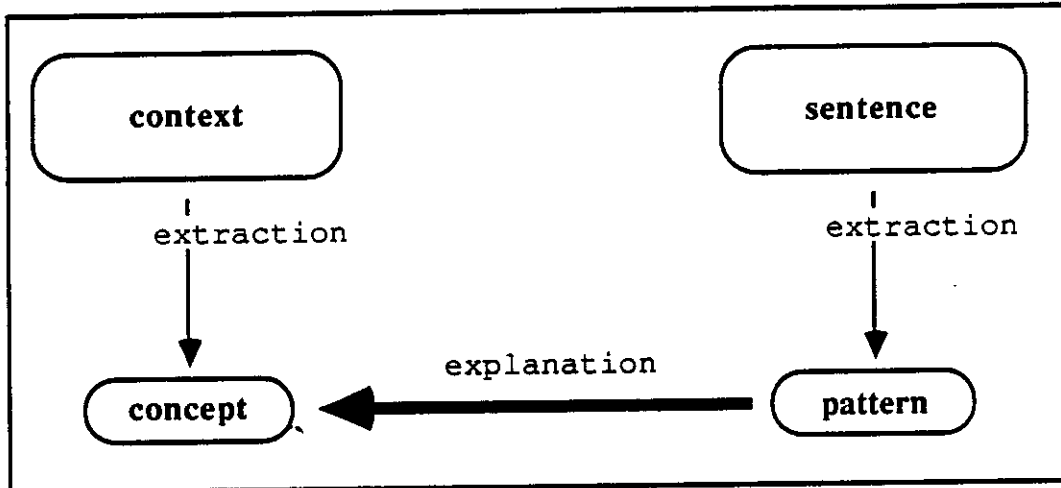


Figure 6.3: The Learning Algorithm

The algorithm takes as input (a) the *sentence* provided by the user, and (b) the explanatory *context*. The inductive leap in learning involves (a) the extraction of the *pattern* from the context, and (b) the extraction of the *concept* from the context. *Explanation* provides the association between the two acquired elements, and supports both steps (1) and (2). The construction of the explanation is not a process on its own*, as in [Mitchell86] rather it is a by-product of learning. Moreover, explanation cannot always be constructed. In fact, the point of this paper is to compare cases where explanation can and cannot be constructed.

6.5 THE PROCESS MODEL

Next, we show the steps in learning bury the hatchet from an explanatory context, as shown in Section 6.1.2.

Detecting a Discrepancy: The acquisition of a new phrase is initiated only if there is no other valid interpretation for the sentence. There is an interpretation for the input sentence In 1977, Israel and Egypt buried the hatchet, based on the single words.

* To avoid confusion, notice that "explanation" is used here to denote a static object, and not a process.

pattern: X:person bury Y:phys-obj

concept: X ptrans Y, causing Y to be under the ground

While syntactically intact, this interpretation, which amounts to "in 1977 they put a hatchet under the ground", fails on semantic grounds. Burying a hatchet does not relate to the given goal-plan situation. In fact, the program detects a near miss [Winston72]. The buried object is not some arbitrary physical object, but it is a fighting tool—albeit for a different kind of fight. Thus, a new hypothesis must be formed.

Extracting a Phrase Pattern: Four discrepancy-driven strategies are employed in this case, where the general intuition is to maintain *frozen* elements which cause parsing failures, and to *generalize* elements which can be explained by the parser. An illustrative example is in learning the phrase Admiral Nelson went to Davey Jones' locker, which includes two references. Nelson is variabilized (turned from a literal into a variable), since a person by that name is accessible to the learner, but the reference Davey Jones' locker is taken as a literal since no referent can be found in the context (who is Davey Jones?).

- (1) The modifier In 1977 is excluded altogether as it is matched by an existing lexical phrase, and it is taken as a standard modifier (the prepositional phrase In T:year exists as a lexical phrase which is taken as a time modifier.
- (2) Both Israel and Egypt can both be resolved in the context, and therefore, these names are variabilized.
- (3) The conjunction itself is not taken as a mandatory element. Conjunction is explained by the context, since Israel and Egypt are both linked by the goal-conflict situation. Thus, the reference becomes:

<Israel and Egypt> --> <N:nation>

- (4) In contrast, the reference the hatchet cannot be resolved, and henceforth it is maintained as a literal:

<the hatchet> --> <the hatchet>

Clearly, this process depends on the actual contents of the lexicon (In T:year exists in the lexicon) and actual world knowledge (in a different culture, where hatchets

serve in inter-nation wars, burying a hatchet could be taken as a concrete act). In this particular state of knowledge the new formed pattern is:

X:nation bury:verb the hatchet

Extracting a Phrase Meaning: The meaning is constructed by parsing the user's clarification:

Israel and Egypt had been involved in a bloody conflict.
In 1977, they signed a peace treaty.

The first sentence establishes the goal-conflict situation (GC) between Israel and Egypt (I&E), which is being pursued in a fight (O1). The second sentence is taken as a switch from O1 to O2 (negotiate). Thus, the meaning is taken as:

goal conflict (GC) exist between Israel and Egypt (I&E)
I&E quit the fight

At this point, the new phrase has a pattern and a meaning:

pattern: <N:nation> bury <H: the hatchet>
meaning: N is involved in a conflict C
N quits fight F

In contrast to the variable N which is bound to a concept in the meaning, the variable H is unbound. No hatchet was found in the context, and as explained previously, H is bound to an element in the metaphor.

The Explanation: The explanation maintained by a phrase *connects* the phrase *concept* to the phrase *pattern*. This is only a "connection" and not a "proof", since by their nature, metaphors do not fully account for meanings of idioms. In our example, four clauses are used in the explanation, obtained by the idiom metaphor:

- (1) I&E bury hatchet H % the sentence itself
- (2) if X bury Y
and Y is a tool
====> X disenable-use of Y % generalizing "bury"
- (3) hatchet H ====> war-implement H % generalizing "hatchet"
- (4) if X disenable-use of implement I for plan P
% an inference rule

phrase. Suppose for example the following input context:

John was standing on the bucket with a rope around his neck.
Then he kicked the bucket away, and fell to his death.

In this case an explanation could be constructed as:

- > X kick the bucket B
- > X is not supported by B
- > X is hanging from the rope R
- > R strangles X
- > X dies

Interestingly, second language learners, when presented with this context, *are able* to generalize the phrase and even use it in the passive voice, since they have an explanation.

6.7 CONCLUSIONS

We have analyzed phrase acquisition as a function of information provided to the learner. We have identified three information sources and evaluated their contribution:

- (1) Lexical entries given for the words constituting a phrase.
- (2) The context underlying the utterance of a phrase.
- (3) Metaphors associated with the words in the phrase.

Both in parsing and in learning, meanings of word combinations are derived by using the single words. However, in parsing meaning derivation involves simply looking up single words in the lexicon. In learning, on the other hand, the literal derivation is not acceptable, and the process is complicated: meaning derivation involves a search for metaphors implied by the single words, and selection of concepts from the context.

We have explained two surprising phenomena:

- (1) How can a phrase be learned from a single example? To that end, we have shown strategies for concept extraction which rely on linguistic clues and on existing metaphors.
- (2) How can a phrase be learned even when a metaphor does not exist (kick the bucket)? The same learning strategies as in (1) are applied with and

without a metaphor.

However, when a metaphor does not exist, the association between the pattern and the meaning remains unexplained. The phrase cannot be used in a general way, and its application is limited.

Part III:

Parsing for Learning

Chapter 7: Performing in the Presence of Incomplete Lexical Knowledge

In the previous part of the dissertation we described the learning algorithm. In describing learning we assumed that the learning algorithm is *provided* with a set of discrepancies which are used for upgrading a hypothesis. However, how are these discrepancies detected, and how is the text processed in the first place in the presence of lexical gaps? Since learning is based on language analysis, or *parsing*, we investigate in this part *parsing in a learning system*. There are three topics: In Chapter 8 we explain how a parser can cope with lexical gaps; in Chapter 9 we describe the monitoring mechanism; and in Chapter 10 we analyze the relation between syntax and semantics.

7.1. INTRODUCTION

How can humans cope with lexical gaps? It turns out that humans are able to *construct* meanings of novel elements by finding analogies with existing elements. Thus, when a specific lexical concept is required, but is yet unavailable, as frequently happens in second-language acquisition, learners are able to process the text by "borrowing" similar existing lexical elements. This process might lead to errors of generation

and comprehension, however it enables second-language speakers to convey (or understand) at least a partial meaning, rather than give up the communication task altogether.

We show here how a single mechanism can be applied in cases of complete as well as incomplete knowledge. By using a hierarchical lexicon, a parser can cope with knowledge gaps, applying knowledge at various level of generality. By their nature, general entries are less informative than the specific ones, and their use might lead to performance errors. Our model explains (1) how people carry out communication tasks in spite of incomplete knowledge, and (2) how people produce such communication errors.

7.1.1 The Linguistic Behavior

The problems arising from incomplete lexical knowledge are illustrated through the following scenario, as they are reflected by the program RINA's behavior. First, RINA reads a paragraph provided by a user, in which she encounters new words. RINA forms a hypothesis regarding the meaning of the input, and then she generates text which conveys the state of her knowledge to the user. The first input sentence is given below:

User: Corinne needed help with her homework.
Frank called and **plended** her to come over.

The word **plend** does not exist in RINA's lexicon*. However, RINA is able to extract partial information: Frank communicated a concept to Corinne regarding coming over. It is not clear however, who comes over. Did Frank *promise* Corinne to come over to her, or did Frank *ask* Corinne to come over to him? The input paragraph, continues:

User: But she **dooved** to stay home.

The word **doove** is also unknown. Here too, RINA can guess the main concept: Corinne *decided* not to come over. This hypothesis is not necessarily correct. However, it fits the context and the structure of the sentence well.

* **plend** is just a hypothetical English word which brings home, even to native speakers, the problems under consideration.

At this point, RINA must respond to the input text by generating a paraphrase which conveys her hypothesis. However, also in generation, RINA faces the problem of incomplete lexical knowledge.

RINA: Frank suggested her to come over.

But she turned down the suggestion.

In absence of specific knowledge regarding the use of *suggest*, RINA produced an incorrect sentence.

At this point the reader might ask why we promote a program which is error prone, and why we focus on incorrect human behavior—picking on second-language speakers' errors. The answer is twofold: this behavior demonstrated that in spite of missing lexical information humans are able, (a) to form an initial hypothesis regarding an unknown word combination, and (b) to convey the essence of the hypothesis of another person. These two elements facilitate learning [Zernik86b].

7.1.2 The Issues

The general problem is this: how can any program parse a sentence when a lexical entry such as *doove* or *plend* is missing? And equivalently, how can a program use a lexical entry—*suggest*—which is not precisely specified? Four issues must be answered to resolve this problem.

Syntax and Control: In *Frank asked Corinne to come over*, the word *ask* actually *controls* the analysis of the entire sentence [Bresnan82b]. The embedded phrase *to come over*, which does not have an explicit subject obtains its subject from the *control matrix* [Bresnan82b] of the entry for *ask*. Accordingly, Corinne is the subject of "coming over".

On the other hand, in *he plended her to come over*, the controlling word *plend*, is yet unknown. In absence of a control matrix it is not clear how to interpret *to come over*. How can a program then, glean even partial information from text in such circumstances?

Initial Hypothesis: The form of the sentence *X plended Y to come over*, suggests that "X communicated to Y a concept regarding coming over". This intuition facili-

tates the hypothesis which initiates the learning process. How is this intuition encoded in the lexicon?

The Context: But she dooved to stay home, might have several possible meanings, among them planning acts such as decide or turn down, aimed at current goals of the character. This intuition stems from the prior context. How does the context affect the parsing process?

Overgeneralization: In absence of specific knowledge, there is a need to generalize. While presenting a necessary capability, generalization might also yield "overgeneralizations". For example, in he suggested her to come over the program committed such an overgeneralization. What is a process model for predicting overgeneralization errors?

Some of the issues above can be handled by specific heuristic rules, custom tailored for each case. However, the challenge of this entire enterprise is to show how a unified model can employ its "normal" parsing mechanism in handling "exceptions".

7.1.3 The Approach: A Unified Parsing Mechanism

So far, natural language parsers have maintained two distinct functions [Granger77, Carbonell84, Pinker84, Zernik85a]. "Normal" parsing of text proceeds as long as "exceptions" are not encountered. When an exception, such as a missing lexical entry is encountered, a second function is called in for handling the exception. However, two distinct functions cannot cover many intermediate situations. For example, in encountering elements such as `plend` or `doove`, where only partial knowledge exists, what model should be applied? This case falls into both the categories above, and thus, the need arises for a uniform analysis. Accordingly, the task is described *not* as (a) text analysis, and (b) language acquisition, but as parsing in the presence of incomplete lexical knowledge.

This task is difficult to model computationally. In order to make an initial hypothesis about an unknown element, a model must first process the text. However, in order to process the text, the model must possess knowledge about all the elements in the text, including the unknown element. Therefore, the problem is to process the text in spite of the present unknown.

Accordingly, the application of such lexical entries in text analysis yields conceptual goal-plan structures. For example, the *homework* situation underlying the text in Section 7.1, is represented in terms of the following goal-plan structure.

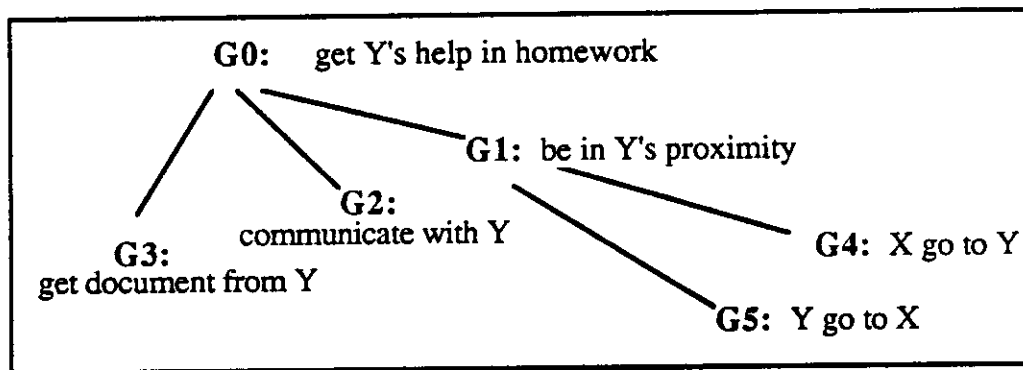


Figure 7.1: Goal-Plan Structure

This structure describes goal-subgoal relations among acts. These relations are domain specific—there is nothing universal about a person getting help in her homework from another person—and they are assumed to be given as part of the program’s knowledge. RINA’s hypothesis, based on this structure is: Corinne has goal G0 (to be helped by Frank). Frank suggests subgoal G1, and in particular subgoal G4 of G1. Corinne in turn, rejects this plan.

This underlying contextual structure is an important factor in the parsing process, especially when lexical knowledge is missing.

7.3 A HIERARCHICAL PHRASAL LEXICON

Two aspects of the lexicon must be specified: contents, given by representations of single entries; and structure, given by the global hierarchy.

7.3.1 Single Phrasal Entries

Consider the representation of word *ask* in the sentence below:

- (1) The meeting was long and tedious. So Frank asked to leave early.

The word *ask* is given below* as an entire phrase, or a *pattern-concept* pair [Wilensky81].

pattern: X:person ask:verb Z:act
concept: X communicated that act Z by X can achieve a goal G of X.

P1: First Lexical Entry for *ask*

The pattern of the phrase has two constituents: a subject X (Frank) and a complement Z (to leave early). In particular, the semantic concept of the phrase specifies that X is the subject of the embedded act Z, a fact which is not explicit in the text. However, this specification fails in capturing further sentences, such as the following one.

- (2) Frank asked **the chairman** to adjourn the meeting.

There are two discrepancies: (a) the sentence above includes a direct object (the chairman), and (b) Frank is not the subject of the complement as prescribed in phrase P1. Thus, a second phrase P2 is added on to account for sentences of this kind.

pattern: X:person ask:verb Y:person Z:act
concept: X communicated to Y
that act Z by Y can achieve a goal G of X

P2: Second Lexical Entry for *ask*

(The differences between P1 and P2 are marked.) However, as further sentences involving *ask* are encountered, and as other complement-taking verbs are considered, we observe that certain properties given in these instances can be abstracted and generalized.

* This notation stresses semantic aspects. For precise syntactic specifications of patterns see [Dyer86b]. In particular, the syntactic definition of Z dictates interaction through the infinitive form: to leave early.

7.3.2 Generalized Features

The phrases P1 and P2 above can be abstracted in three ways.

First, semantic properties of *ask* itself can be generalized across other instances of *ask* by the general phrase:

pattern: X:person ask:verb Z:act
concept: X communicate that act Z can achieve a goal G of X

P3: A Generalized Phrase for *ask*

This generalized phrase simply states the meaning of *ask*, namely "X says that he wants Z to be executed", regardless of (a) who is the object of the communication act, and (b) who executes the act Z.

Second, semantic properties determining the identity of the subject in verb complements can be generalized across many verbs, by general phrases (called also *equi-rules*) which are stated as follows:

pattern: X:person V:verb Z:act
concept: X is the subject of the embedded act Z

pattern: X:person V:verb Y:person Z:act
concept: Y is the subject of the embedded act Z

P4 and P5: Generalized Phrases for Verb Interaction

These phrases dictate the identity of the subject in complement-taking verbs such as *tell*, *decide*, *force*, *start*, etc.

Third, Semantic properties of communication acts can be even further abstracted:

pattern: X:person V:verb Y:person Z:act
concept: Y communicated Z to X

P6: A Generalized Phrase for Communication Verbs

Phrase P6 can be applied across many verbs which share the same pattern as P3.

However, P6 does not mean that any clause abiding by P6's pattern conveys a communication act; there are other phrases which carry different meanings, but share this common syntactic pattern (e.g., select, force, etc).

7.3.3 The Hierarchical Structure

Consequently, a hierarchical structure is introduced in DHPL.

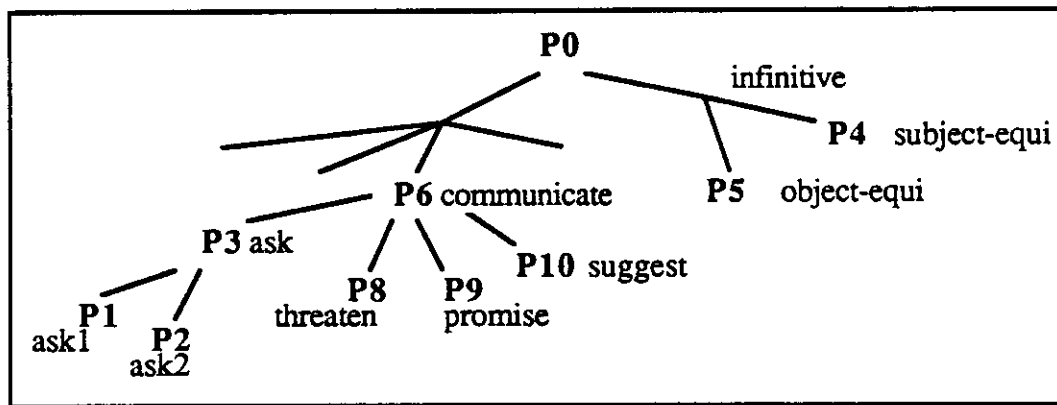


Figure 7.2: The Hierarchy for Complement-Taking Verbs

This hierarchy* specifies interaction of phrases with embedded phrases. The full hierarchy contains many more ways of interactions than we have shown here.

- (1) At the top of the hierarchy, there are general phrases denoting general "grammar rules" such as P4 and P5.
- (2) Lower in the hierarchy there are phrases such as P6, the *communication* phrase.
- (3) Under P6 in the hierarchy there are phrases such as P3 (the generalized verb for ask) and phrases for other communication verbs (e.g., suggest, tell, etc).
- (4) At the bottom of the hierarchy there are specific phrases which describe specific forms of each verb, such as P1 and P2 for ask.

* Notice that each node in this hierarchy is a full-fledged phrase; the mnemonic words are for reference only.

Therefore when a new word, or a novel use of a word (a use for which there is no specific lexical entry) is encountered, a program can apply generalized entries in predicting the meaning.

7.4 PHRASE INTERACTION

The previous section defines the structure of the lexicon. Yet, we must explain how the lexicon becomes operational in parsing text. Consider the following three sentences, ordered according to their complexity.

- (1) Frank came over.
- (2) Frank asked Corinne to come over.
- (3) Frank plended Corinne to come over.

(a) Sentence (1) is analyzed by a simple table lookup. A phrase (P7) is found in the lexicon, and its concept is instantiated.

pattern: X come over
concept: X ptrans to the location of another person Y

P7: The Specific Phrase for *come over*

(b) No **single** lexical phrase matches sentence (2). Therefore, the analysis of (2) involves interaction of two lexical phrases (P2 and P7).

(c) No **specific** lexical phrase matches (3), since it includes an unknown word. Therefore the analysis of (3) involves interaction with generalized phrases.

7.4.1 Specific Interaction

Two phrases, P2 and P7, take part in parsing sentence (3), as shown schematically in Figure 7.3 below.

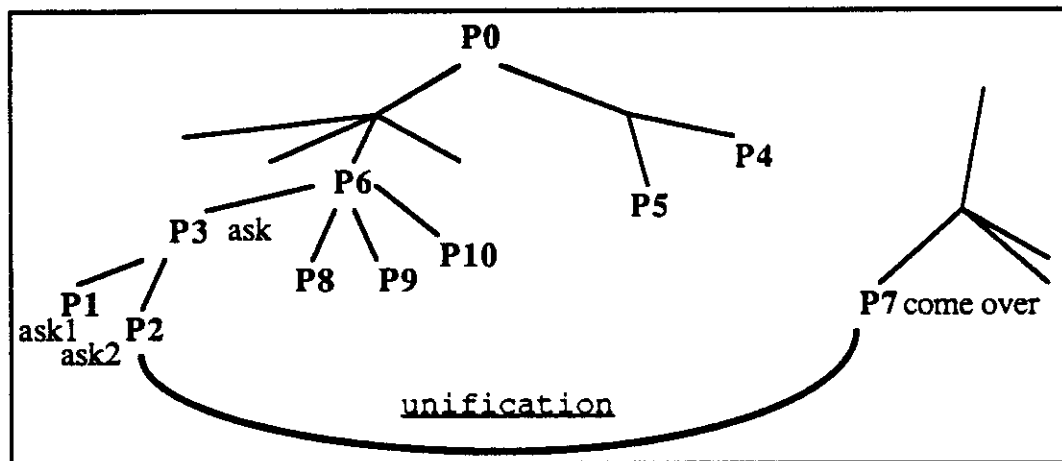


Figure 7.3: Interaction with a Specific Phrase

Interaction is by unification [Kay79], and it is carried out in two steps:

- (1) **Unify** the individual patterns of the phrases P2 and P7. In particular, the variable X in the embedded phrase (P7) inherits the value of Y from the embedding phrase (P2).
- (2) **Instantiate** the composite concept of P2 and P7:

F.13 communicated to C.17 that:

C.17 coming over to F.13 will achieve a goal of F.13

where F.13 and C.17 are the objects representing Frank and Corinne in the database.

7.4.2 General Interaction

No specific phrase in the lexicon matches the word *plend*. However, the clause matches two generalized phrases P5 and P6, as well as the specific phrase P7, as shown in Figure 7.4 below:

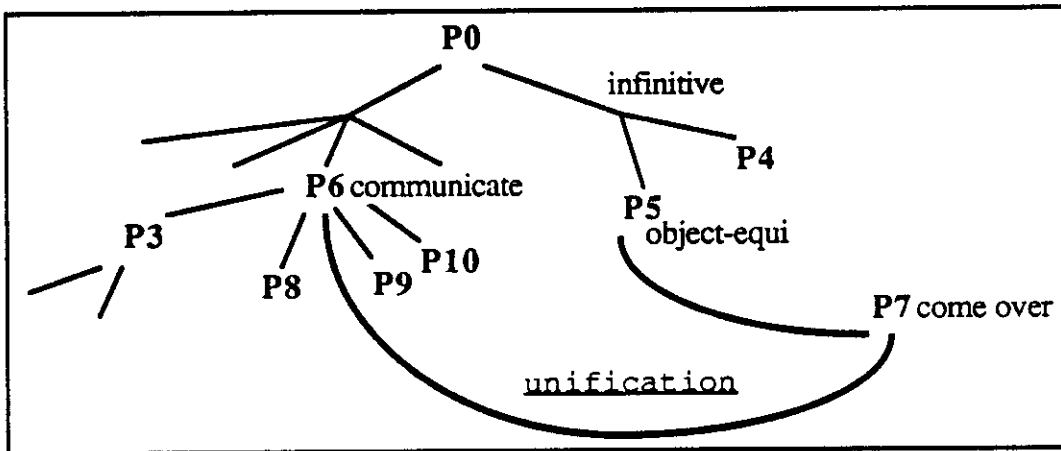


Figure 7.4: Interaction with a Generalized Phrase

The generalized phrase, P6 implies that *plend* is a communication act, it leaves some parameters unspecified. In particular, the identity of the subject of the embedded phrase is yet unknown. Phrase P5 contributes that parameter by unification (P5 dictates *object-equi*). As a result, the constructed concept is:

F.13 communicated to C.17 that:
C.17 will come over to F.13

However, how was P6 selected in the first place? Obviously, the situation is not unambiguous. A second matching phrase P8 could have meant he influenced her to come over, (which is not a communication act) or P9, he promised her to come over which implies that F.13 will come over to C.17 (*subject-equi*). However regardless of the precise concept, the main goal, namely forming an initial hypothesis, has been accomplished. In general, resolution of this kind of ambiguity cannot be handled at this stage, since there is no sufficient information. Further refinement and correction of the hypothesis must be pursued by receiving additional input, such as:

User: He just wanted her to come over.

This new input allows the program to refine the meaning of the word `plend`.

7.5. GENERATION: A PROCESS MODEL

Consider the task involved in generation of a paraphrase for a concept.

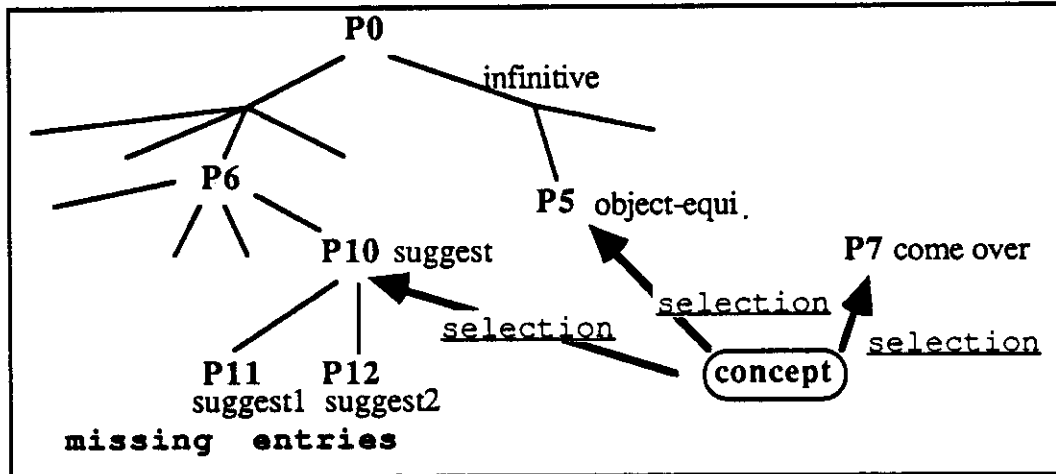


Figure 7.5: Generation Using a Generalized Phrase

As shown in the scheme above, the generator is given two elements:

- (a) The concept to be described by the generated text:

X communicated to Y that:

A goal G of Y can be achieved by act Z (X come over to Y)

- (b) A hierarchical lexicon. In particular, notice that P10, the general phrase for `suggest` is given, yet P11 and P12, the phrases describing specific interactions of `suggest` (i.e., he suggested to come over, and he suggested to her that she come over) are missing.

In this particular configuration of knowledge, the generated sentence is this:

Frank suggested Corinne to come over.

Generation proceeds in three steps:

- (a) Select a specific entry for the embedded act Z. P7 is selected to describe C.17 come over to F.13.
- (b) Select a specific entry for the communication act. P10, the phrase for suggest matches the goal-plan situation: "X communicated to Y that: act Z achieves Y's goal". However, P10 does not specify the precise syntactic interaction of suggest with its constituents:

X suggest...Y...come over

There is a need for an interaction phrase to connect these elements.

- (c) Select an interaction phrase. The input concept matches P5, the the general *object equi* phrase, due to the structure of the communication act in the concept. This lexical selection causes the error.

This explains similar errors made by second language speakers. Although the sentence sounds awkward (to a native speaker), it certainly conveys the main concept, and a user is acknowledged of the program's hypothesis. The general principle is summarized below:

Specific phrases are preferred to general phrases. However, in absence of a precise specific phrase, resort to the use of a general phrase.

This presents a basic principle of knowledge-based systems. Wilensky [Wilensky81] has applied this principle in parsing and in generation: in case of two competing lexical phrases, select the more specific one. However, this principle has not yet been applied in language acquisition.

7.6 CONCLUSIONS

Text analysis tasks can be ordered by increasing difficulty: (1) Performing by assuming perfect knowledge: analysis fails when unknown elements are encountered. (2) Performing even in the presence of incomplete knowledge: partial concepts are produced when unknowns are encountered. Yet, new elements are not incorporated. (3) Filling in knowledge gaps by acquisition: new elements encountered in the text are acquired and added on to the lexicon.

We have addressed here tasks (1) and (2). Whereas in task (3) the lexicon itself is upgraded as a consequence of parsing, tasks (1) and (2) involve the parser itself. We have described a uniform, *unification-based* mechanism which accounts for parsing in the presence of missing knowledge. Coping with knowledge gaps is done by applying phrases at various levels of generality.

7.7 LIMITATIONS

Ambiguity in Learning: Parsing as well as learning involves formation of a hypothesis, and both tasks are fraught with ambiguity. However, in learning, when a specific lexical entry does not exist, hypothesis formation is an *under-determined* problem. Namely, many possibilities exist for the initial hypothesis. For example, in the sentence:

He plended her to come over,

it is not clear which one of the following concepts *plend* is similar to: *ask* *promise* *influence* or *wish*. We are not able, with the given information, to determine the precise category. Hypothesis correction requires further input, such as:

He insisted that she come at 8pm.

By this input, the cases of *promise*, and *wish* are ruled out. Thus, the initial hypothesis is a basis for further refinement.

Sound Patterns: People's hypotheses are biased by *sound patterns* [Chomsky68]. Accordingly, two frequent guesses for *plended* are *pleaded* and *prodded* which share some phonetic features with *plend*. Our model of language analysis cannot combine linguistic clues at that level. A possible mechanism, which could incorporate linguistic clues at many levels, currently under consideration, is similar to the connectionistic model suggested by Waltz and Pollack [Waltz85].

Chapter 8:

Parsing is Monitored

As shown in the previous chapters, in acquiring new words and phrases, a learning model corrects its hypothesis according to the nature of detected failures. Therefore, a learning model based on language analysis, must not only analyze the text, but it must also *monitor* the parsing process itself, and make inferences *about* parsing. However, this task is complicated by *ambiguity*. Since even simple sentences can lead to more than one interpretation, the model must systematically monitor the entire set of possible interpretations, rule out the inappropriate ones, and account for interpretations which present "near misses".

In this chapter we describe a mechanism for language analysis which handles both ambiguity and detection of errors by monitoring multiple interpretations. It features three properties:

- (1) Lexical entries are represented as phrases, which *associate* syntax and semantics.
- (2) Language analysis is based on *entire-phrase interaction* (rather than interaction of single words).
- (3) Multiple interpretations are monitored, where syntactic and semantic *discrepancies* are detected and treated.

While in previous chapters we focussed on aspects of *error-correction*, in this chapter

we focus on *error detection* and monitoring of the parsing process itself.

8.1 INTRODUCTION

Semantic parsers convert sentences into semantic representation. Ideally, each sentence should have a unique interpretation, but unfortunately, natural language analysis is fraught with two problems: (a) ambiguity and (b) knowledge gaps. On the one hand, even a simple sentence such as

Mary was taken by the car dealer.

might have many interpretations, two of which are: "she fell in love", and "she was cheated". On the other hand, a language learner reading the same sentence, might not be familiar with the phrase "to be taken", and for him/her the sentence might not have any interpretation at all.

Detection and analysis of parsing failure is a precondition for learning: only by failing to construct a valid interpretation for *she was taken*, does a learner identify the existence of an unknown phrase. As long as a valid interpretation exists, it might be used by the program, and learning would not take place.

In this chapter we describe the mechanism driving parsing in the program RINA. This mechanism is *non-deterministic* since it spans simultaneously all possible interpretations, upon which the program either (a) selects the most appropriate interpretation, if one or more interpretations exist, or (b) initiates learning, if no valid interpretation exists.

8.1.1 The Linguistic Phenomenon

Text analysis is determined by the *context*. Consider for example the following sentence, which is given in absence of context (imagine hearing an isolated sentence on entering a room):

P1: She took it up with her dad.

In absence of context, this sentence hardly makes sense. What are the referents for *she* and *it*, and what is the meaning of *take*? In an appropriate context, however, this sentence can be unambiguously interpreted:

P2: Jenny thought jogging was an easy sport.
She took it up with her dad.

This sentence now means that Jenny started jogging with her dad. However, in a second context, the sentence assumes an entirely different meaning:

P3: Jenny needed money for her new car.
She took it up with her dad.

Here, it is understood that Jenny discussed a problem with her dad. And yet in a third context, there is another interpretation:

P4: Jenny's mother needed the vacuum cleaner upstairs.
She took it up with her dad.

Here *take* comes in its simple meaning: physically transferring (*ptrans*) an object upstairs.

Text interpretation is also relative to the state of *lexical knowledge*. Consider the situation in which a language learner encounters a sentence for which his/her lexicon does not include the appropriate phrase.

Native:	Jenny wanted to buy a new car. She took it up with her dad.
Learner:	She took the car uphill?
Native:	No. She took up the problem with him.
Learner:	They discussed buying a new car?

In this case, in absence of the appropriate phrase, the learner first applies a different phrase which seems applicable. Thus, unless the learner can identify the discrepancy with this interpretation, the new phrase would be passed over unnoticed.

8.1.2 The Theoretical Issues

Semantic ambiguity must be accounted for at all levels of communication. In the paragraphs above we have identified three sources of ambiguity.

Lexical Ambiguity: Many phrases in the lexicon involve the word *take*. Some are shown below:

X take up Y (an issue) with Z (an authority)
X take up Y (an activity-theme)
X take up Y (a solution)
X take up with Z (a mate)
X take Y (a physical object)

How can the appropriate phrase be selected in each one of P1-P4?

Referential Ambiguity: References, and pronouns in particular, might refer to many objects in the context. For example in P3 above, the reference *it* can refer to any one of the objects mentioned in the prior context:

the car (a certain car instance called CAR.74)
Jenny's goal to buy a car (an instance of a goal called GOAL.15)
the money required for the deal (MONEY.14)
Jenny herself (JENNY.13) - this one is refuted trivially,
since Jenny is a person

How is selection of a referent (reference resolution) related to the selection of the entire verb phrase?

Modifier Ambiguity: Phrase complements, and modifiers in general, introduce further ambiguity. For example, there are three phrases for *with* in the lexicon:

X do V with Y:	X act with Y's company, as in she dined with her family.
X do V with Y:	X act with Y's assistance, as in she practiced tennis with a pro.
X do V with Y:	X act with an instrument Y, as in she ate rice with chopsticks.

How is the appropriate complement selected in each one of the cases above? The complexity of the problem is illustrated schematically by the graph in Figure 8.1 below.

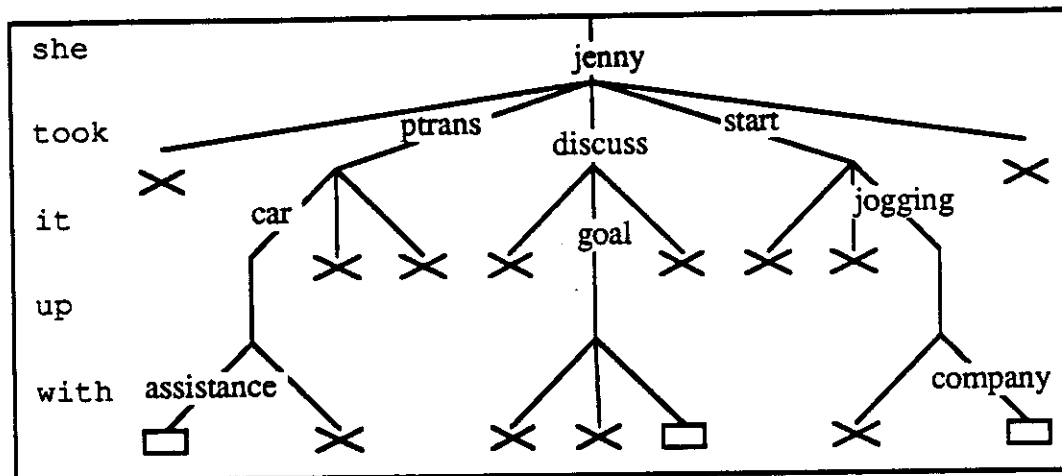


Figure 8.1: Multiplicity of Interpretations

In this tree, each leaf represents an interpretation, and each branch is a separate word meaning. Inappropriate interpretations are crossed out as discrepancies are detected. Notice that a large number of interpretations was created, although we ignored many interpretations.

8.1.3 Theoretical Approaches

There are two theoretical approaches to semantic disambiguation: the quantitative and the qualitative. The quantitative approach, is represented by *connectionistic* models such as [Waltz85, Cottrell85, McClelland86]. Selection of interpretations is according to a preference function which is computed for each interpretation. A variety of preference functions have been introduced so far, where selection is done by a measure of connection weights. In Waltz's model [Waltz85] for example, two interpretations exist for a sentence such as the sailor ate a submarine. As parsing proceeds, the total weight shifts from one interpretation (submarine = a vessel) to the second (submarine = a sandwich).

The qualitative approach, on the other hand, is based on examining the validity of logical predicates. This set of predicates is resolved as a *constraint-satisfaction* problem. The first mechanism for processing systematically multiple competing interpretations

was presented by [Small82]. In Small's model, procedurally-encoded "experts" attached to individual words, were spawned in order to check validity of word interactions.

The phrasal approach [Becker75, Wilensky81, Fillmore87] is another *qualitative* approach, emphasizing *modularity* and *declarativeness*. This approach provides an abstraction by elevating language analysis from interaction of single words to entire-phrase interaction. Lexical entries are not given as single words (e.g., take), rather as entire phrases (e.g., take over, take it up with, take on, etc), and consequently, disambiguation involves selection of entire phrases. This method proves effective in parsing [Wilks75, Wilensky80] and in generation [Jacobs85b].

Since our task involves language acquisition, we selected the method which is most amenable to learning. The phrasal approach, being modular and declarative, turns out to be advantageous since it enables us to make inferences *about* language analysis.

8.1.4 The Program

A phrasal parser which monitors multiple interpretations consists of three main functions:

Phrase Interaction: Identifies lexical phrases accessed through the text. Constructs entire sentence interpretations by *unification* [Kay79] of individual phrases.

Discrepancy Detection: This function detects syntactic and semantic discrepancies in the interpretations constructed by (1).

Error Analysis: Based on detected discrepancies, this function uses learning strategies to form new hypotheses (this function is not elaborated in this chapter).

This mechanism depends for its operation on existence of two kinds of knowledge sources:

Lexical knowledge: parsing is relative to a certain *incomplete* lexicon which is augmented by discrepancy-driven strategies.

World knowledge: error analysis is supported by given rules and facts of the domain.

In fact, one way of testing out the program is to observe its behavior under various

sets of lexical and world knowledge.

8.2 THE PHRASAL LEXICON

In order to apply a single unifier in text analysis, lexical entries must all be represented uniformly. Here, a lexical entry—a *phrase*—is a triple associating a linguistic *pattern* with its *concept* and a *situation*. We show how a variety of lexical elements (verb phrases, modifiers, and noun phrases) are given using this method.

8.2.1 A Verb Phrase

Consider the phrase in the following text.

Jenny needed money. She took it up with her dad.

This phrase is represented as a triple:

LP1: **pattern:** X:person <take up> Y:goal-situation
<with Z:person>
situation: Z can solve problem Y for X
concept: X discuss Y with Z

After reading the first sentence, the context contains several concepts including *jenny.13* and *jenny.13's* goal to possess money (*goal.15*). The second sentence is parsed in four steps:

- (1) The pattern is matched successfully against the text. Consequently, X and Y respectively are bound to people called Jenny and Willard (Jenny's father) respectively. Z is bound to *goal.15*.
- (2) The situation is validated using the context. Jenny's father has money he can give Jenny.
- (3) Since both (1) and (2) are successful, then the pattern itself is instantiated, adding to the context: *jenny.13 discussed goal.15 with willard.17*.

Phrase *situation*, distinguished from phrase *concept*, is introduced in our representation since it solves two problems: (a) in *disambiguation* it provides a discrimination condition for phrase selection, and (b) in *acquisition* it allows the incorporation of the context of the example as part of the phrase.

For reading convenience, the phrase pattern was abbreviated above as follows:

pattern: X:person <take up> Y:goal-situation <with Z:person>

However, internally, this pattern is denoted using a *slot-filler* notation as shown below:

(subject	(verb
((concept X)	((verb take)
(class person)))	(modifier up)))
(object	(object
((concept Y)	((marker with)
(class goal-situ)))	(variable Z)
	(class person)))

This representation is geared to unification and instantiation. Notice, that syntactic relations are not specified explicitly in the pattern itself: word-order information, for example, is inherited from general syntactic templates.

8.2.2 Modifiers

Consider the following paragraph:

Jenny's mother needed the vacuum cleaner upstairs.
She took it up **with her dad**.

The modifier *with* is represented as the triple below:

LP2: **pattern:** X:person Y:act <with Z:person>
concept: Z supports X in executing Y
situation: Z is in a position to support X

The *situation* is used in discriminating a sentence such as she runs with a coach, from sentences such as she runs with her friends. Notice that a single word *with* is denoted here as a full-fledged phrase.

8.2.3 References

References too are given in the same phrase notation. For example consider two words which appear in paragraphs P1-P4, *it* and *father*.

LP3: **pattern:** father
situation: class family
role parent
concept: class person
gender male

LP4: **pattern:** it
concept: class *not* person

The *situation* (in *father*) is used for identification of the *family* structure in the context, since *father* is a relation which is part of *family*. This is the declarative implementation of Cullingford's Cullingford78 algorithm for resolving role-holder referents. Thus, we have shown how verbs, modifiers, and references, are all represented uniformly as phrases.

8.3 THE CONTEXT

The context is represented as a structure of plans and goals. For example, the context underlying paragraph P3 is given in the figure below.

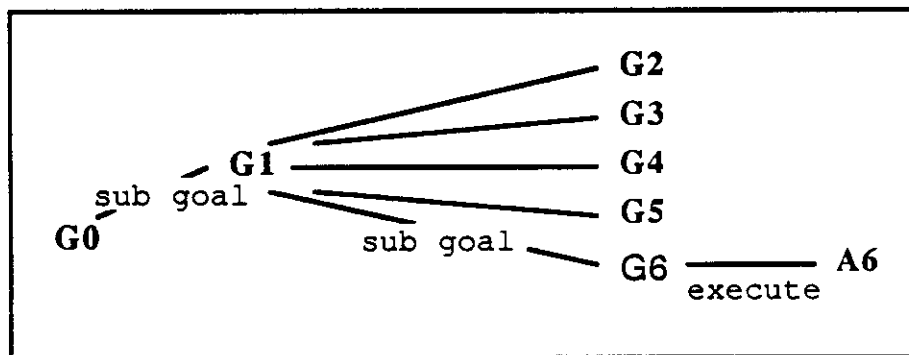


Figure 8.2: A Goal-Plan Representation

This scheme gives relations between goals and their subgoals [Schank77]. The major

goal G0 (buying a car) is preconditioned by its subgoal G1 (getting money). G1 on its part, can be accomplished by one of five *alternative* plans: (G2) work and earn the money, (G3) draw money from one's bank account, (G4) steal, (G5) sell old car, and (G6) borrow money from a relative. This scheme of plans and goals is given as domain knowledge. The parser on its part identifies G6 as the plan selected by Jenny. In executing this plan, Jenny communicates the situation to her father, in order for him to give her the money.

8.4 PHRASE INTERACTION

In this section we describe how entire interpretations are constructed from their constituent phrases.

8.4.1 Levels of Interpretation

Consider for example, the parsing process in analyzing the second sentence in the following paragraph:

```
Jenny needed money for her car.  
  She took it up with her dad.
```

The meaning is constructed at four levels.

Single Words: Single words are packaged into case frames:

```
<She> <took up> <it> <with her dad>
```

Although `took` and `up` are separated in the text, they are packaged into the same case frame (`up` is taken as the verb modifier).

Case Frames: case frames get bound to concepts which are found in the context.

```
she: JENNY.13   took: ptrans   it: CAR.74   with her dad: WILLARD.17  
                                GOAL.15  
                                MONEY.14
```

Notice that three referents (non-person objects) are found for `it` in the context. At this level, the word `it` cannot be uniquely resolved, since the context contains three non-person concepts.

Phrases: Multiple case frames are packaged into phrases:

X take up Y with Z - JENNY.13 took up GOAL.15 with WILLARD.17
X take Y - JENNY.13 took CAR.74

Notice that the referential ambiguity for *it* disappeared at this level, since each phrase selected one of the three objects according to *class specifications*.

Interpretations: Interactions of entire phrases yield entire-sentence interpretations.

she (J.13) took it (GOAL.15) up with her (J.13) dad (W.17)
she (J.13) took it (CAR.74) -- up (upwards) -- with her dad(W.17)

The first interpretation consists of a single lexical phrase ("she discussed a problem"). The second interpretation, on the other hand, consists of a verb phrase and two complements.

8.4.2 Types of Interactions

Pairs of phrases interact in two steps, in a process of *unification* and *instantiation* [Charniak80].

- (1) The patterns of the individual phrases are *unified*.
- (2) If the unification in (1) is successful, then the composite concept is *instantiated*.

Phrase interaction is simplified through the case-frame abstraction, as shown by two types of interactions: *internal interaction*, and *external interaction*.

Internal Interaction: An interaction of a phrase with a constituent such as a reference or an embedded phrase is called *internal interaction*. Consider for example the marked reference *it* in the text below: Jenny's mother needed the vacuum cleaner upstairs. She took it up with her dad.

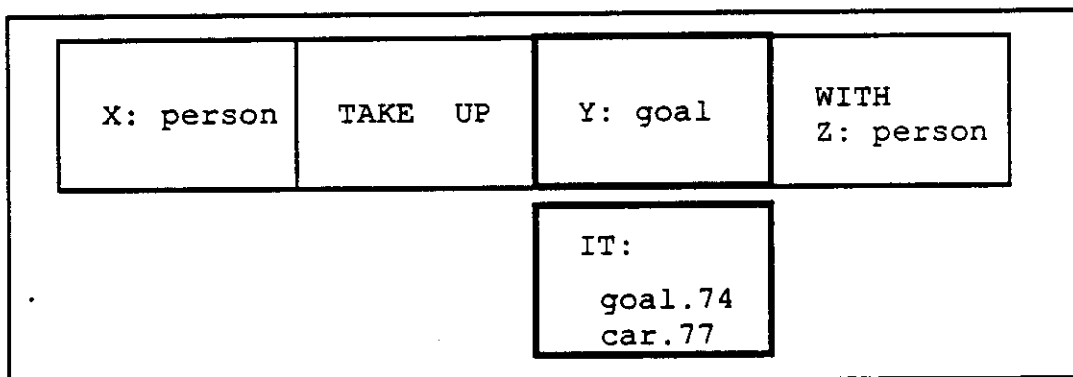


Figure 8.3: Interaction with References

The reference *it*, packaged in a case frame, interacts with the lexical phrase:

Each case frame includes candidate referents. The case frame for *she* is associated with *jenny.13* and *wilma.4* as referents, while *it* is associated with *vacuum.98* and *goal.101*. Due to *class specifications* on X, *goal.101* is rejected, and X gets bound to *jenny.13*. The discrimination between *jenny.13* and *wilma.4* (*jenny's mother*) cannot be resolved by class specifications, and thus there are two interpretations at this point, which need to be pruned by other means. The instantiated interpretations are as follows:

```
X:jenny.13 take:ptrans Y:vacuum.98
X:wilma.4 take:ptrans Y:vacuum.98
```

External Interaction The interaction of a phrase with phrase modifiers is called *external interaction*. For example in the sentence she took it up with her father, the main verb phrase interacts with two modifiers: up, and with her father. The interaction is shown schematically below:

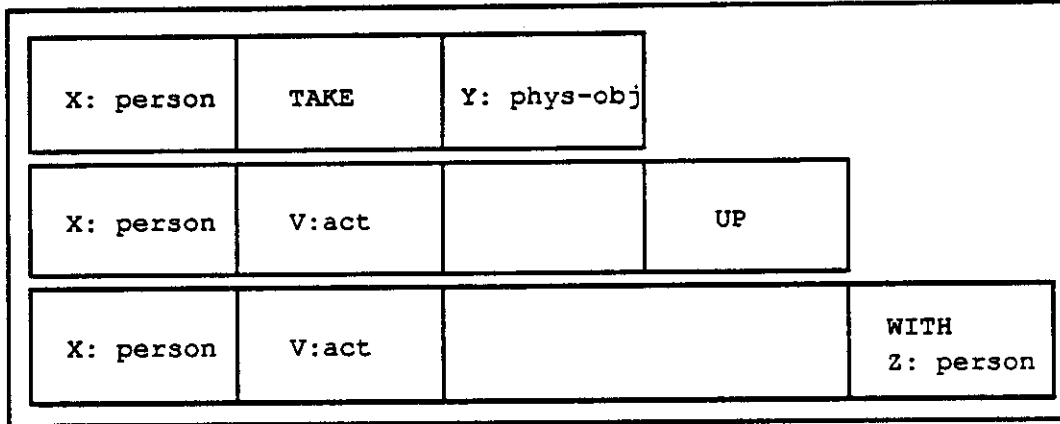


Figure 8.4: Interaction with a Complement

Interaction is by case frame unification, as the unifier guarantees that there are no syntactic or semantic contradictions between matching case frames. The composite concept resulting from the instantiation of the three phrases is:

```

act      ptrans
actor    jenny.13
object   vacuum.98
direction positive-vertical
supported-by willard.17
  
```

However, in coming up with a single concept, we ignored ambiguity stemming from various sources:

- (1) There are two instances for she took it (as shown above).
- (2) There are two possible phrase for with: with *assistance* and with *company*.
- (3) There are two instances for her in with her mother (whose mother?)
- (4) The syntax here is not unambiguous either. The reference her father could actually be interpreted as two separate references. This kind of ambiguity is better demonstrated in the following pair of sentences:

He took her home (he also took the rest of her property).
He took her home (he took her in his new car).

- (5) Finally, this entire discussion assumed only one meaning of *take* (*ptrans*). Other senses of *take* must also be considered.

Thus, we ended up with at least 16 interpretations for this simple sentence. Some of the interpretations in this set are obviously inappropriate:

wilma.4 take:ptrans vaccum.98 with wilma.4

However, the point of this entire exercise is to show that interpretations cannot not be ruled out at the outset: all interpretations must be systematically constructed and examined before they can be ruled out.

8.5 DISCREPANCY DETECTION

So far we have shown how a large set of interpretations is constructed. However, how is this large set reduced so that a single relevant interpretation is left out? Phrases are ruled out by syntactic as well as semantic discrepancies. We describe here context-related semantic discrepancies.

Lexical Presupposition: Phrase *presupposition* accounts for the matching of a phrase with its anticipated context. For example consider this pair of paragraphs:

- P3: Jenny needed money for a car.
She took it up with her father.
P5: Jenny started jogging.
She took it up with her father.

Compare the application of lexical phrase LP1 (X discuss a problem with Y) in both cases. The presupposition of LP1 requires that the father would be in a position to solve a problem of Jenny. In P3, a suitable problem is found: Jenny's father can support with giving her money. Thus the interpretation *Jenny discussed the problem with her father* is appropriately applied. In P5, such a problem of Jenny is not found and a discrepancy is detected with the interpretation suggested by LP1.

Referent Not Found: Another kind of discrepancy is based on *class specifications*.

For example, consider phrase LP6 below:

LP6: **pattern:** X:person <take up> Y:activity-theme
concept: X started activity Y

The variable Y can be bound to referents such as jogging, stamp collecting, etc. However, in the context under consideration:

Jenny wanted to buy a car,

there is no such activity, and the phrase LP6 is ruled out. Indeed an activity, namely, "buying a car" does exist, however, this is a one-time act and not an activity-theme.

Common Sense: Consider again the following context:

Jenny wanted to buy a car.

An interpretation such as "jenny drove the car up the hill" is refuted by rules of the domain: a person cannot drive a car he does not possess. Clearly, this ruling is based on given domain knowledge, and under certain given rules, this interpretation could make sense ("she tested the new car by driving it uphill")

8.6 ERROR ANALYSIS

Now consider the case described in Section 1.1, in which a phrase such as LP1 does not exist in the model's lexicon, and must be acquired.

input: Jenny wanted to buy a new car.
She took it up with her dad.
interpretation: she ptransed it --- up --- with her dad
output: She drove the car uphill with her dad?

In absence of the phrase LP1 (X discuss a problem with Y), the learner interpreted the sentence using an interaction of three existing phrases. The output was marked by a ? (question mark) to denote a detected discrepancy (how can Jenny drive a car she did not possess?). Next the user provides the second sentence, as given below:

input: No. She took up **the problem** with her dad.

The user indicates explicitly that the object (it) pertains to the problem and not to the car. This input invokes three discrepancies which is associated with error-correction

strategies. One such strategy is given below:

old-hypothesis: X take Y:possession
discrepancy: referent class is not subsumed by reference class
action: change reference class
new-hypothesis: X take Y:goal-situation

The entire learning process and a large set of error-correction strategies is described elsewhere [Zernik85c]. In general, hypothesis modification is driven by discrepancy detection.

8.7 CONCLUSIONS

Semantic analyzers have dealt with the construction of a few designated interpretations, out of which a single one was conveniently selected. However, no parser so far has been designed to facilitate learning.

Reasoning About Parsing: In order to facilitate learning, a parser must be able to identify and draw conclusions from existing discrepancies. However, since even simple sentences lead to multiple interpretations, it is important to consider systematically all possible interpretations. The model of semantic parsing described here, can monitor interpretations in situations of multiple ambiguity, as required by a learning program. Only by being able to make inferences about parsing, can a program learn new phrases.

The Case-Frame Abstraction: Two abstractions have been used in our model, to factor out low level details of parsing.

- (1) Sentence meanings are constructed by the meanings of individual phrases. Thus, meaning construction is elevated from interaction of single words to interaction of entire phrases.
- (2) Phrase unification is done by unifying entire case frames, assuming that low level parts of speech are already packaged into case frames.

This method enables us to express error-correction rules at a higher level of abstraction.

8.8 LIMITATIONS

It Does not Make Sense: In ruling out interpretations we tried to identify interpretations which do not make sense. This task is difficult for theoretical reasons. Within a model, a proof can be found for a certain proposition—if the connecting rules exist in the model's database. However, if a proof for a proposition cannot be found, it does not *positively* refute the proposition—it simply means that a proof was not found within the model's memory. Thus, it is necessary to find *positive* conditions for proposition refutation. This task has not been within our scope, so in our model, we resorted to simple heuristics instead.

Tie Breaking: Indeed we have shown how inappropriate phrases can be rejected. However, there is no way of selecting among a set of multiple interpretations which have not been rejected. Consider the following sentence:

Jenny needed a partner for the party.
Finally she decided to take Fred.

Although the interpretation "she took him dancing" is preferred by a human reader, our model cannot determine between that interpretation and the alternative interpretation "she cheated him" (as in he was taken by the car dealer). This problem stems from the nature of the *qualitative* approach. An interpretation is either valid or is not valid. There are no intermediate preference values (between 0 and 1). This problem may be remedied by using a Waltz and Pollack-style connectionistic mechanism [Waltz85] which features *preference* of certain interpretations, as well as *inhibition* of others.

Chapter 9:

Bootstrapping Syntactic Patterns from Semantic Concepts

In this chapter we investigate a syntax-semantics *duality*, and its manifestation in parsing and in acquisition. Lexical phrases are comprised of two parts: (a) a syntactic pattern, and (b) a semantic concept. In parsing a sentence, the meaning of the sentence can be derived either (a) by a syntactic interaction of the entities in the text, according to grammar rules, or (b) by a semantic interaction according to knowledge of the domain. Normally the syntactic derivation prevails since it provides a *short cut* to the long semantic derivation. In learning, however, there are situations in which the syntax is yet unknown. In those cases the dual derivation facilitates acquisition of the new syntactic pattern. Thus, the unknown pattern is *bootstrapped* from the given semantic concepts.

9.1 INTRODUCTION

In the *phrasal* approach to language processing rather than maintaining a single generic lexical entry for each word, e.g.: *take*, the lexicon contains many phrases, e.g.: *take a break*, *take it easy*, *take someone to task*, *take to the streets*, etc. In this approach, the task of disambiguation as in *the workers took to the streets* vs. *the juvenile delinquent took to the streets* is based on selection between entire phrases. Also the task of concept formation shifts to the interaction between entire phrases. Although the lexicon contains isolated phrases such as: *make up one's mind* and *take to task*, it is necessary to analyze phrase combina-

tactic pattern and the semantic concept of the interaction from the example given in the *teacher-student* context. On another occasion, RINA hears a dialog between Jenny and her mother.

Jenny: I asked to go to the bathroom.

Mom: You asked who to go to the bathroom?

Jenny: I asked Mr. Wilson to go to the bathroom.

(The last sentence is said with anger. Jenny's tone implies "now I gave you all the details, please leave me alone!") Clearly, Jenny did not mean that the teacher was supposed to go to the bathroom. In spite of the apparent deviation from English conventions, RINA realizes what Jenny really meant.

RINA: Jenny wanted to go to the bathroom.

She asked for permission from her teacher.

9.1.2 The Issues

Six issues must be addressed with regard to phrase interaction, while parsing the text above:

Forming Composite Concepts: The major objective is to construct composite concepts for complex clauses. Consider the following two sentences:

Judge Wilson **decided that** Mary threw the book at John.

The Judge **decided to** throw the book at Mary.

Although in both sentences the constituent verbs (*decide* and *throw*) are the same, the composite meanings are entirely different. The respective meanings are:

- (a) "Wilson **mbuild** a fact: Mary **propelled** a book at John"
- (b) "Wilson **select** a plan: Wilson will **punish** Mary severely".

Only the use of two different interaction forms: *decide to* vs. *decide that* can account for the construction of these different meanings. Thus, what is the process through which different meanings are constructed for such sentences?

Syntactic Interaction: One lexical entry for *throw* is:

Person1 throw the book at Person2

However, this pattern cannot be matched against the clause in the text:

Judge Wilson decided to throw the book at John.

since the Person1 is not explicit in the text. Only by phrase interaction can this pattern matching be satisfied and Person1 be bound to Wilson. Syntactic patterns resolve bindings of such phrase roles. Thus, the interaction of syntactic patterns must be used in constructing composite concepts.

Semantic Interaction: Contrary to textbook grammar, people do use semantics to override strict syntax rules:

My father did not let me see that movie.
But he promised me to go to Disneyland.

Such "violations" of textbook grammar are easy to demonstrate using children's language. However, they can be heard also in adult speech, and be appropriately understood in context by a human listener. Thus, the program must construct the meaning of a clause by negotiating also its semantic context.

RINA must apply knowledge beyond syntax also in analyzing the next sentences:

John **started running** after the bus.
Since it was a long stretch he needed to **stop to catch his breath**.

Since the verb *stop* can interact through the infinitive, the appropriate meaning must be inferred. Note that the meaning is *not* that he stopped breathing, but that he stopped in order to breathe. Semantic concept interaction must be used as a second way in constructing composite concepts.

Generalization: A pervasive dilemma in lexical representation is whether a feature should be encoded as a single-phrase *idiosyncrasy*, or it should be encoded as a *generality* which is shared among many phrases. For example, consider the behavior of MTRANS verbs (e.g.: ask, tell, instruct, inform, etc.) as they interact with their embedded phrase through the infinitive form. In a phrase such as John told Mary to do something, Mary holds two roles in the constructed concept: (a) Mary is the recipient of the MTRANS act, and (b) Mary is the actor of the embedded act.

This interaction must appear as a general lexical entry and be applied in all specific phrases, for two reasons:

- (a) Modification, which occurs by learning, requires changes only in a single entry.
- (b) When a new verb is encountered (such as *remind*) it is possible to apply the general form before hearing a specific example, and thus make a prediction.

Recovering from Overgeneralization: While a feature is shared among a set of verbs, there are always exceptions. For example, the behavior of the verb *promise* deviates from the general infinitive behavior of MTRANS verbs.

User: John promised him to come early the next day.
RINA: The boss will come early?

Another example of overgeneralization (elaborated in Chapter 6), involves the verb *suggest*:

He suggested her to come over.

While the program must be able to make correct as well as incorrect predictions, it must also be able to backtrack from overgeneralizations in case of an incorrect prediction.

Bootstrapping a Syntactic Pattern: We have specified two ways in which composite concepts can be derived: by the syntax of lexical patterns and by semantic inferences. While this duality presents a redundancy in most cases, it has a significance for learning. In learning, when the syntax of a lexical item is still unknown, the semantic derivation will present the only possible way to construct the concept. Moreover, it can also serve in acquiring the syntactic pattern. For example, assume that the special interaction pattern for *promise* is still unknown, when the following sentence is encountered.

John was late to work today,
so he **promised his boss to come** early tomorrow.

It is clear from the context (*employer-employee* authority relationship) that the action referred to is John's coming early. This causes a conflict with the syntactic derivation

(the default MTRANS infinitive) which suggests that "the boss" is the actor. By detecting this discrepancy, the syntactic pattern for `promise` is modified. Thus, the program must analyze such discrepancies and come up with the modified interaction patterns.

9.1.3 Lexical Representation

Lexical Phrases: RINA uses a declarative phrasal lexicon. Below are two sample phrasal patterns.

```
P1:      X:person give:verb <a break> <to Y:person>
P2:      X:person decide:verb <to Y:phrase(X)>
```

These patterns are used for reference only. They actually stand for the slot fillers given below:

```
P1:  subj  (inst X) (class person)
      verb  (root decide)
      comp  (inst Y)
           (form infinitive)
           (subj (inst X))

P2:  subj  (inst X) (class person)
      verb  (root throw)
      obj   (inst Z) (marker at) (class person)
      obj   (det the) (noun book)
```

In P1, the *infinitive* form accounts for detecting the appropriate morphology of the embedded clause. The variable *X* accounts for equating the subject of the embedded phrase with the subject of the controlling phrase.

Lexical Semantics: The concepts used in our examples involve knowledge of goal/planning and of interpersonal relationships. The semantic representation of phrases is given as a *situation-concept* pairs. The presupposition provides the required background for the phrase application, and the concept is the element to be instantiated. For example, the situation of the phrase `X:person throw the book at Y:person`, is given below:

```
(authority (high X) (low Y))
```


meaning that X presents an authority for Y. The concept itself is given as:

```
(auth-decree (actor X)
              (object (thwart-goal
                       (goal p-goal)
                       (goal-of Y))))
```

This means: X decreed that a preservation goal of Y will be damaged. Thus, in order to understand text involving this phrase, the program must have knowledge about *authority* and people's plans and goals.

9.1.4 Phrase Acquisition

Learning is by error correction. For example, in acquiring the phrase for *promise*, RINA first hypothesizes the pattern:

```
X:person promise:verb Y:person <to Z:phrase(Y)>
```

This hypothesis might incorrectly cause RINA to understand he promised her to go as he promised her that she would go. By providing an appropriate context (he promised her to come to her place), the user causes RINA to modify her hypothesis into:

```
X:person promise:verb Y:person <to Z:phrase(X)>
```

which has the correct syntax for *promise*. The hypothesis is updated (1) by detecting a discrepancy, and (2) by correcting the pattern causing that discrepancy.

9.2 PARSING A COMPLEX CLAUSE

In the previous section we described the three steps involved in parsing a simple clause. However, the process gets complicated when the clause includes two interacting phrases, as shown in the last sentence in the story S1 below.

```
S1:  Mary went to court for her traffic violation.
      She had gone through a red light.
      She asked Judge Wilson to give her a break,
      but the judge decided to throw the book at her.
```

The phrases taking part in the clause above are P1 and P3 below; P2 and P4 are candi-

dates which must be discarded through disambiguation.

P1:	pattern:	X decide <to Y:phrase(X)>
	concept:	X select-plan Y
P2:	pattern:	X decide Y:sentence
	concept:	X mbuild Y
P3:	pattern:	A throw <the book> <at B>
	presupposition:	A is an authority for B
	concept:	A punish B severely
P4:	pattern:	A throw ?c <at B>
	concept:	A propel ?c to B

The context prior to reading the last sentence is \$trial (the trial script) [Dyer86b], where mary1 is the defendent and wilson1 is the judge. Parsing the clause requires (a) that the patterns of the selected phrases be unified successfully with the text, and (b) that the situations of the selected phrases (if it is prescribed) be unified successfully in the context. Left-to-right parsing of the sentence proceeds through the following steps:

- (1) decided triggers two lexical phrases: P1 and P2 (where X is bound to wilson1).
- (2) to throw triggers phrases P3 and P4.
- (3) The infinitive is detected (in to throw) and is used to mark the *form* in P3 and P4.
- (4) the book and at Mary match the patterns of both P3 and P4 (in both, B is bound to mary2).
- (5) The situation of P3 matches the context through \$trial, which includes the relationship:

(authority (high wilson1) (low mary2))

As a result, A in P3 is bound to wilson1.

- (6) The pattern of P1, in particular the *comp* part which is now:

(**subj** (inst wilson1))
(**verb** (root decide) (tense past))

```
(comp (inst Y)
      (form infinitive)
      (subject (inst wilson1)))
```

is matched successfully:

- (a) The form of P1's complement (P3) is indeed the infinitive.
 - (b) The subject of P3 matches the subject of P1, wilson1.
- (7) The concept of P3 is instantiated as:

```
(auth-decree (actor wilson1)
             (object (thwart-goal (goal p-goal)
                                   (goal-of mary2))))
```

- (8) The concept of P1 is instantiated. It includes the concept of P3.

```
(select-plan (actor wilson1)
             (object
              (auth-decree (actor wilson1)
                           (object (thwart-goal
                                     (goal p-goal)
                                     (goal-of mary2)))))))
```

Disambiguation was carried out to select between P1 and P2, and between P3 and P4. P2 was ruled out (and P1 was selected by default) since its syntactic pattern (specifically its complement's form) did not match the sentence. P3 and P4 were both unified successfully; P3 was selected by being more *specific*.

The purpose of this entire exercise has been to demonstrate how the implicit subject was derived in two ways. Through the syntactic pattern (step 6b), and through the situation (Step 5). In this case, both derivations yielded the same result, namely wilson1. When a conflict arises between these two derivations, then learning is initiated.

9.3 LEARNING A NEW INTERACTION

Now, consider the case in which an unknown phrase is encountered. For example, the phrase in the following sentence from the story S1 above.

Mary asked Judge Wilson to give her a break.

The lexicon does contain a phrase to process this sentence.

Context Representation: The trial script is given as a chain of events.

1. The prosecutor tells the judge about the law violation.
2. The defendant tells the judge about the law violation.
3. The prosecutor asks the judge to give a severe penalty.
4. The defendant asks the judge to lighten the penalty.
5. The judge decides, either to:
 - a. give the defendant a light penalty, or
 - b. give the defendant a severe penalty.

This script, given in greater detail in Chapter 4, represents the context.

Given Lexical Knowledge: Initially the program's lexicon includes two given phrases. The phrase for ask:

P5: pattern: X:person ask:verb <Z:question-sentence>
concept: (mtrans (actor X)
(object (goal-of X)
(d-know Z)))

This phrase means X say that X want to know Z. It can produce sentences such as he asked if it rained, he asked whether it rained, or he asked when it rained. However, it cannot produce a sentence such as he asked to go. Second, there is the knowledge of the general MTRANS phrase:

P6: pattern: X:person ?v:verb Y <to Z:phrase(Y)>
concept: (mtrans (actor X)
(to Y)
(object (goal-situation Z)))

This general phrase is shared among a set of phrases, such as tell, order, instruct, etc. It denotes the common syntactic interaction through the infinitive and the

common structure of the semantic representation: the actor informs the recipient a certain fact which conveys a common goal/plan of the speaker.

Initial Hypothesis: Two discrepancies are encountered in parsing the sentence using the existing phrase, P5: syntactic and semantic. First, P5's pattern cannot match the *infinitive* form which appears in the input text. Therefore, the program resorts to the *general* phrase P6, to extend the syntax of P5. Using the syntax of P6 with the specific meaning of P5 yields a new hypothesis.

P7: **pattern:** X ask Y <to Z:phrase(Y)>
concept: (mtrans (actor X)
(to Y)
(object (goal-of X)
(d-know Z)))

This means X tells Y that knowing Z is a goal of X, where X is the actor in the event Z, as implied by the syntactic pattern. Using this phrase, a conceptual discrepancy is encountered. From the trial script which contains several expected events, Mary is expected to make a plea or to explain her situation to the judge. She is not expected to ask the judge a question.

Updating the Hypothesis: The context, given by \$trial contains a slot in which the Defendant appeals to the Judge to minimize the penalty. This slot is matched with the event described in Mary asked the judge to give her a break. Using this match, the hypothesis is corrected:

P8: **pattern:** X ask Y <to Z:phrase(Y)>
concept: (mtrans (actor X)
(to Y)
(object (goal-of X)
(act Z)))

This means: X tells Y that doing Z is a goal of X. This phrase describes the speech-act performed by "X ask to do something." The *interpretation* of such a speech-act is according to the context. Thus, in Mary asked the judge to give her a break, the act is interpreted as *auth-appeal*, while in the judge asked Mary to approach the bench, the act is interpreted as *auth-decree*, due to the asymmetry of the *authority* relationship.

Thus, RINA has used the general MTRANS phrase and the knowledge of the context to (a) extend the syntax of the given lexical phrase, and (b) to learn a new meaning.

9.4 HANDLING AN OVERGENERALIZATION

In a similar way RINA acquires the special syntax of the word `promise`. Initially the lexicon contains only the general phrase P5 and the phrase:

```
P9: pattern: X promise <to Y:phrase(X)>
      concept: (mtrans (actor X)
                (object (plan-of X)
                        (act Z)))
```

This phrase can handle sentences such as `he promised to go`. When the next sentence is heard, RINA makes an overgeneralization by using the syntax of P5:

User: The judge promised Mary to punish her lightly.

RINA: The judge told Mary that she would punish herself lightly?

The generated concept is:

```
(mtrans
  (actor wilson1)
  (to mary2)
  (object (plan-of wilson1)
    (act (actor mary2)
      (plan (auth-decree
        (actor mary2)
        (object (polarity neg)
          (cause (thwart-goal
            (goal p-goal)
            (goal-of mary2)
```

This concept contains two obvious discrepancies. First, the original sentence should have been `The judge promised Mary to punish herself lightly`, and second, this concept does not fit in the context. By matching it with the existing slots in `$trial`, the identity of the "punishing" act is attributed to `the judge`. Accordingly, the syntactic pattern is corrected.

9.5 CONCLUSIONS

In this chapter we have described the aspect of phrase interaction, focusing our attention on the interaction of complement-taking verbs with their constituent phrases. This specific language feature and the specific context chosen as the background for the given text, serve to highlight our theory of phrasal processing.

Semantic concepts as well as syntactic patterns take part in the process of conceptually analyzing text. Normally, the syntactic derivation of a clause is much simpler than the tedious semantic reasoning process. However, when the syntactic pattern is still unknown, the conceptual interaction provides an alternative method in computing the meaning of the clause. Moreover, the syntax itself can be bootstrapped from the semantic derivation. Thus, the syntax is viewed as a short-cut, or a macro-operator, which is based on previous lengthy derivations.

Part IV:

Design and Implementation

Chapter 10:

Implementing a Self-Extending Parser

In the previous parts we have presented the learning and the parsing algorithms. However, we did not describe in detail the mechanisms themselves. In this part, we describe in detail: (a) the operation of the parser (Chapter 10), and (b) the learning strategies (Chapter 11).

In this chapter we describe the actual implementation of RINA's parser. RINA is designed to cope with incomplete lexical knowledge, and accordingly, RINA's parser is required: (a) not to stall in the presence of unknown phrases in the input text, and (b) to detect discrepancies in parsing, thus to motivate the learning process. Accordingly we have stated two principles:

- (a) For linguistic knowledge to be *learnable*, it should not reflect either the logic of the idiosyncratic parsing mechanism or the specifics of the programming language in which the parser is written.
- (b) For the parsing to motivate learning, the parser must monitor interpretations. Parsing is *rigorous* rather than *lenient* [Granger83]. In other words, the parser is not required to function in spite of discrepancies by ignoring them, rather is expected to highlight detected discrepancies.

Accordingly we have designed the structure of the lexicon and the mechanism which applies lexical entries in parsing.

10.1 INTRODUCTION

The program RINA [Zernik87b] is structured in a way that emphasizes parsing at the phrase level. RINA's mechanism can be compared to DYPAR [Dyer83] and to PHRAN [Wilensky84].

- (1) DYPAR was an *expectation-based* parser. It operated by identifying verbs and keywords in the input stream of words. Each keyword spawned demons which anticipated other words and concepts. Only by satisfying the syntactic and the semantic expectations of its demons, could the concept of the keyword be instantiated.
- (2) PHRAN was a *phrase-based* parser. On the identification of a keyword, a set of lexical phrases was activated. Each phrase was matched against the words and concepts coming in the input stream. In case that the matching was successful, the concept of the phrase was instantiated.

RINA is both phrase based and expectation based.

- (1) Like PHRAN, RINA too activates entire phrases which associate syntax and semantics. However, in RINA phrases, are comprised of entire *case frames* [Carbonell84] (and not of single words as in PHRAN) which enable abstraction of low-level parsing.
- (2) While expectations in DYPAR were embodied by procedural demons, RINA's phrases account for context dependency by declarative lexical *presuppositions*.

Thus, the program RINA is structured in a way that emphasizes parsing at the phrase level.

10.1.1 The Structure of the Program

The program is described along two lines: (a) the functional components, and (b) the data structures.

Case-Frame Parsing: Two Processes and a Stream: RINA analyzes phrases at the

case-frame

[Carbonell84] level. This is in contrast to DYPAR [Dyer83] and PHRAN [Arens82] which inspect an input stream of single words and concepts. Thus, RINA operates as two piped *processes*, interacting through a *stream*. The two processes are:

- (1) Case-frame parsing (the "low level" process). This process pre-packages words into entire case frames.
- (2) Phrasal parsing (the "high level" process). This process can be thought of as a parser whose input tokens are not words but entire case frames.

This separation between these processes is only conceptual. The two processes operate simultaneously as text is read left-to-right. One process places case frames on the stream while the second process retrieves them as input tokens. The structures in both processes are similar, and both involve pattern unification. Yet they were separated in order to enable phrasal parsing at a higher level of abstraction. It is advantageous to carry out phrase unification at the case-frame level, since it is preferred to report parsing discrepancies at this level (As shown in Chapter 7, an error reported at the case level would be: a case marker ON is missing. The same error would be reported at the word level as: word ON, which is the 4th word in the pattern, is missing).

The Data Structures: The program maintains five data structures.

- (1) The *phrasal lexicon* which provides the linguistic database.
 - o Each entry, a phrase is a triple: a syntactic *pattern* associated with a *concept* and a *presupposition*.
 - o The lexicon includes specific phrases which pertain fixed word combinations, as well as general phrases which encompass many word combinations.

The lexicon provides the entire linguistic knowledge required by the parser. There is no other linguistic source in the system.
- (2) The *context* reflects the conceptual structure of the discourse.
 - o The context accounts for reference retrieval by organizing concepts mentioned in the text.

- o The context provides a port through which the output of the parser is accessible to the external world including the rest of the system itself.
- (3) The *database* contains domain knowledge in terms of rules and facts. Phrases are interpreted using such knowledge.
 - (4) The *case-frame-stream* is an intermediate representation of the input stream of words. Case frames on the stream are *produced* by the case-frame parser, and are *consumed* as input tokens by the phrasal parser.
 - (5) The *active-phrase-stack* is the structure containing all candidate interpretations.
 - (6) The *current hypothesis* is the tentative phrase under construction. This phrase is being upgraded through the examples provided by the user.

10.1.2 The Functionality of the Program

The behavior of the program is specified in two ways. By parsing, and by learning.

The Basic Parsing Cycle: Parsing presents the short-term effect of reading text. By parsing, text is converted into internal representation. The scheme of the operation is shown below:

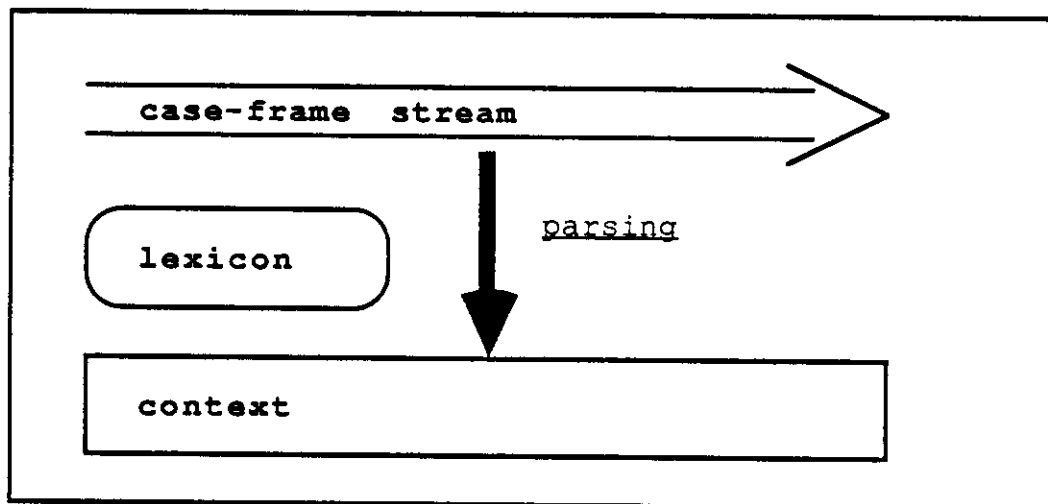


Figure 10.1: The Parsing Scheme

By reading a sentence, input tokens, or case frames, are read in the input. Relative to

(a) the lexicon, and (b) the current context, the meaning of the sentence is constructed and placed in the context.

The Basic Learning Cycle: Learning presents the long-term effect of parsing. By learning, discrepancies discovered in the *parsing process itself* cause the upgrading of the lexicon. The scheme is shown below:

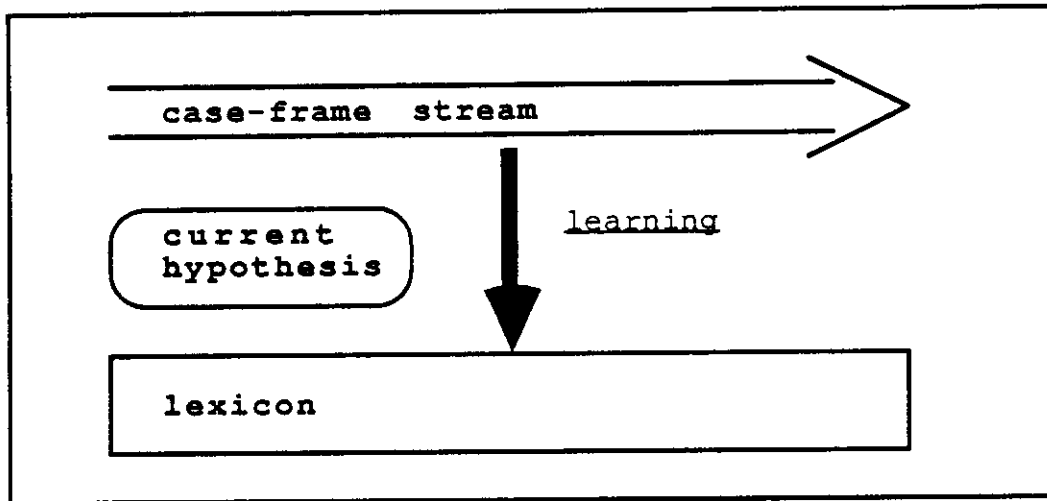


Figure 10.2: The Learning Scheme

By reading a sentence, a discrepancy is detected between the current hypothesis and the input sentence. Relative to a set of discrepancy-driven strategies the hypothesis is upgraded and placed in the lexicon.

10.1.3 Design Goals

The original set of design goals are stated below:

- (1) *Declarative* representation. Neither LISP, nor any other programming language, should be a prerequisite for using the parser, and for encoding lexical entries.
- (2) *Transparency* of the mechanism. No knowledge of the idiosyncratic logic of the parser should be required in order to use the parser.
- (3) *Rigorous* parsing. Information should be gleaned from linguistic clues at all levels of speech. Thus, although we focus on semantic parsing, we do not downplay the significance of linguistic clues.

- (4) *Multiple* interpretations. Ambiguity arises at all levels of speech. All possible interpretations of the text must be maintained and monitored for the parser to eventually select the appropriate one.
- (5) *Concurrency* at the cognitive level. Clearly, any program may be executed on a parallel computer by relying solely on the allocation mechanism of the machine itself. However, our purpose to maintain concurrency at the *interpretation* level.
- (6) *Phrasal unification*. Linguistic knowledge represented modular declarative units become operational by unification.

A prerequisite to implementing a self-extending parser is a parser whose lexicon is easy to upgrade manually— the rationale being that if lexical entries are not modular, not declarative, and do not reflect the idiosyncratic logic of the specific parser, then (a) a human operator might find it difficult to upgrade the lexicon *manually*, let alone (b) it would be impossible to modify lexical entries *programmatically*.

In this chapter we describe the design issues and their solutions as implemented in RINA. The intention of the chapter is threefold:

- (1) To provide a user's manual to the program RINA.
- (2) As a parsing algorithm which can be duplicated in other environments.
- (3) As a set of conventions for setting up an operational lexicon.

10.2 CASE-LEVEL PARSING

The case-frame parser converts a stream of input words into a stream of case frames. It carries out four tasks:

- (1) **Match Syntactic Patterns**
-identify simple syntactic patterns at the level of word-group morphology
- (2) **Instantiate semantic features**
-glean the semantic information conveyed by the syntax.
- (3) **Look up the lexicon.**
-for content words, find word meanings in the lexicon.

(4) **Retrieve referents.**

-use the lexical descriptions of step (3) to retrieve instances from the context.

Each one of these steps is given in greater detail in the next sections.

10.2.1 Matching Syntactic Patterns

Combine words and morphemes into *noun groups* and *verb groups* such as:

the young man
had not been taken

Figure 10.3: Two Word-Group Examples

Identify simple syntactic patterns, such as:

(type determiner) (type describer) (type noun)
or
(root be) (suff en)

Figure 10.4: Two Syntactic Patterns

Each one of these patterns specifies properties of adjacent words (i.e. <the young man> and <been taken>).

10.2.2 Constructing a Case Frame

Using the meanings of such patterns, construct the entire case frame for a word group. For example, for the group:

had not been taken

Figure 10.5: A Verb Group

determine the following properties:

((head verb)
(root take)
(tense past)

```
(aspect perfect)
(voice passive)
(polarity negative))
```

Figure 10.6: The Constructed Case Frame

Word group properties are given as pattern-concept pairs.

10.2.3 Looking Up the Lexicon

Access the lexicon. For example, for the group:

```
the young man
```

Figure 10.7: A Noun Group

where the constructed case frame was:

```
((determiner the)
 (describer young)
 (root man))
```

Figure 10.8: The Constructed Case Frame

get from the lexicon the phrases for *young* and *man*,

```
((age N-))      ((class person)
                 (gender male))
```

Figure 10.9: The Lexical Concepts

and construct the description:

description1:

```
((head person)
 (gender male)
 (age N-))
```

Figure 10.10: The Combined Concept

10.2.4 Retrieving Referents

According to **description1** above, find a matching concept in the context. Place the concept in the *concept* slot of the case frame.

```
((head noun)
 (root man)
 (describer young)
 (determiner the)
 (concept john.4 paul.62))
```

Figure 10.11: The Instances Found in the Context

Notice that more than one referent can be found for a reference. The packaged case frame is placed on the case-frame-stream.

10.2.5 The Case-Frame-Stream

The contents of the *case-frame-stream* is illustrated by a typical example, as yielded in parsing the sentence:

```
Last week she took it up with her dad.
```

is given below:

```
CASE-FRAME.148:
  ((CONCEPT ^WEEK.2071)
   (ROOT WEEK)
   (MARKER NONE)
   (DESCRIBER LAST)
   (DETERMINER NONE))
```

The case frame is given in a slot filler notation. It packages syntax and semantics. *Concept* holds the instance of the object referred to. *root*, *marker*, *describer*, and *determiner* hold syntactic features identified in the reference.

```
CASE-FRAME.149:
  ((CONCEPT ^JENNY.1095)
   (ROOT SHE)
   (MARKER NONE))
```

Jenny which was found in the context is taken as the referent.

```
CASE-FRAME.150:
  ((TENSE PAST)
   (ROOT TAKE)
   (VOICE ACTIVE)
   (MODIFIER UP)
   (POLARITY POS))
```

This case frame describes the single word `take` and its modifier `up`. Although `up` and `take` are separated in the text, they belong in the same case frame.

```
CASE-FRAME.151:
  ((CONCEPT
    ^CAR.2100
    ^GOAL.2125)
   (ROOT IT)
   (MARKER NONE)
   (DETERMINER NONE))
```

The reference `it` retrieved two referents: the car, and Jenny's goal to buy a car. The slot-filler representation allows multiple values per slot.

```
CASE-FRAME.152:
  ((CONCEPT ^HAROLD.2758)
   (ROOT DAD)
   (DETERMINER ((HEAD POSSESSIVE)
                (ROOT SHE)
                (CONCEPT ^JENNY.1095))))
```

This case frame involves two references: `her` and `dad`. Both need to be resolved. `her` refers to Jenny and `dad` (in conjunction with `her`) refers to Jenny's dad.

case frame	mnemonic name
CASE-FRAME.148: ((CONCEPT ^WEB.2071) (ROOT WEEK) (MARKER NONE) (DESCRIPTOR LAST) (DETERMINER NONE))	<week>
CASE-FRAME.149: ((CONCEPT ^WEB.1095) (ROOT SHE) (MARKER NONE))	<jenny>
CASE-FRAME.150: ((TENSE PAST) (ROOT TAKE) (VOICE ACTIVE) (MODIFIER UP) (POLARITY POS))	<take-up>
CASE-FRAME.151: ((CONCEPT ^WEB.2100 ^WEB.2125) (ROOT IT) (MARKER NONE) (DETERMINER NONE))	<it>
CASE-FRAME.152: ((CONCEPT ^WEB.2758) (ROOT DAD) (DETERMINER ((HEAD POSSESSIVE) (ROOT SHE) (CONCEPT ^WEB.1095))))	<with-dad>

Figure 10.12: The Case-Frame Stream

10.3 PHRASE-LEVEL PARSING

The major task in phrasal parsing is resolving ambiguity at the phrase level, while coping with multiple values for case-frame concepts. Phrasal parsing is carried out in seven steps:

- (1) **Trigger a phrase.**
-add the phrase to *active-phrase-stack*.
- (2) **Inherit general linguistic properties.**
-inherit word-order from general linguistic patterns.
- (3) **Unify the pattern.**
-unify the lexical case frames against the tokens on *case-frame-stream*.
- (4) **Prove the presupposition.**
-use the facts in the *context* plus the facts and rules in the *database*, try to prove the presupposition.
- (5) **Instantiate the concept.**
-using the binding list acquired in steps (3) and (4) instantiate the phrase concept.
- (6) **Interact with other phrases.**
-in case there are "juxtaposed" phrases, create the combined meaning.
- (7) **Select a phrase.**
-select one of the phrases on the *active-phrase-stack* and place its meaning in the *context*

These actions are described in greater detail in the next sections.

10.3.1 Triggering a Phrase

An indexing scheme is used for reducing the number of phrases being actually processed. Theoretically, phrasal parsing could be done by matching *all* the phrases in the lexicon. However, theoretically, there are tens of thousands of phrases in the phrasal lexicon, and it is desirable to process only the relevant ones.

A simple solution would be to index each phrase by its head, i.e.: take would be the index for take it up with. However, in the phrasal lexicon there are hundreds of phrases under take, and it is not practical to consider them all in each sentence including take.

The solution is to index verb phrases under the *fixed* elements in their pattern. For example consider the phrases below and their list of indices:

PATTERN	INDEX-LIST
X take Y up with Z	take up with
X throw the book at Y	throw the book at
X take up Y	take up
X put X's foot down	put foot down

Figure 10.13: Lexical Entries and their Indices

Only if *all* the indices for a phrase are encountered, then the phrase is triggered. This way, only a minimal set of phrases is considered at each instant.

For example as the following sentence is read from left to right,

She took it up with her dad.

these are the phrases triggered on each word:

WORD	TRIGGERED-PHRASE
she	
took	X take Y
it	
up	X take up Y
with	X take Y up with Z
her	
dad	

Figure 10.14: Phrases Triggered in Reading a Sentence

Notice, the fact that a certain phrase is triggered does not mean the phrase has been matched. It merely means the phrase is a candidate for matching.

In the future we intend to use also semantic features in the indexing scheme, so that phrases such as:

X take Y (as in She was taken by the car dealer)
X take Y (as in I take it that you are tired)

would not be considered each time *take* is encountered. Such a scheme would use the presupposition as an index.

10.3.2 Unifying Case Frames

In order for a phrase to be instantiated its *pattern* must first be matched successfully with the input stream. Therefore, unification is carried out for each phrase, between:

- (1) The cases of the phrase *pattern*.
- (2) The cases of the *case-frame-stream*.

Unification is based on GATE's unifier [Mueller84] which matches a template against a concept. Accordingly, the partially filled template is provided by the lexical case; the fully specified case frame is provided by the *case-frame-stream*. Unification is demonstrated for the sentence:

Last week, she took it up with her dad.

The *case-frame-stream* consists of the following case frames (the names are mnemonic):

The case-frame-stream :

.. <week> <jenny> <take-up> <it> <with-dad>...

The active-phrase-stack :

P1: <?x> <take> <?y> (as in take her home)
P2: <?x> <take-up> <?y> (as in take up jogging)
P3: <?x> <take-up> <?y> (as in take up an offer)
P4: <?x> <take-up> <?y> <with-?z> (as in take it up with an authority)

Figure 10.15: Input Cases vs. The Lexical Cases

where names of case frames are mnemonic. A pattern is matched if all its cases are matched. Therefore, in this case:

- P1: <take-up> does not match <take>
- P2: <?y> requires a *soc-activity* which is not found under <it>
- P3: <?y> requires an *offer* which is not found under <it>
- P4: All the cases are matched.

As a result of the unification of P4, a binding list is created, containing the following bindings:

```
((x jenny.1) (y goal.3) (z harold.4))
```

Figure 10.16: The Binding List

In carrying out this unification we ignored the aspect of *case-frame order*. Namely, the lexical pattern of P4, for example, does not specify that <?x> should appear before <take-up>. This information is provided by inheritance.

10.3.3 Inheriting Case Properties

General linguistic properties of phrases are **not** specified in each individual phrase. The properties which are not specified in individual phrases are inherited from general lexical phrases.

Inheriting by Unification: An Example: Consider the inheritance of word-order properties in parsing the sentence:

```
She took it up with her dad.
```

The pattern of **phrase5** is merged with general templates which can contribute further properties. For example two inheritance patterns which can possibly unify with the lexical pattern above are:

```
T1:  
((head pattern)  
 (subject ((loc bef) (marker none))))
```

```

(verb ((loc ref) (voice active)))
(object1 ((loc aft)))
(object2 ((loc aft)))

```

T2:

```

((head pattern)
 (object1 ((loc bef) (marker none)))
 (verb ((loc ref) (voice passive)))
 (subject ((marker by) (loc any)))
 (object2 ((loc aft))))

```

Figure 10.17: Word Order Templates

Each one of these templates is *merged* with the lexical pattern above. Merging the pattern **P1** with the template **T1** yields the pattern:

P2:

```

head    pattern
verb    ((loc ref) (root take) (modifier up) (voice active))
subject ((loc bef) (marker none) (concept ?x) (restriction r1))
object1 ((loc aft) (marker none) (concept ?y) (restriction r2))
object2 ((loc aft) (marker with) (concept ?z) (restriction r3))

```

Figure 10.18: P2: Augmenting Pattern P1 by Template T1

This pattern incorporates word-order information, and is amenable to unification with the case-frame stream below.

```

((no 11) (describer last) (root week) (concept web.112))
((no 12) (root john) (concept web.3))
((no 13) (root take) (modifier up) (tense past) (voice active))
((no 14) (root it) (concept web.12 web.107))
((no 15) (root dad) (determiner his) (concept web.3))

```

Figure 10.19: The Case-Frame Stream

The *location* information in the augmented pattern enables unification of input case frames with lexical case frames. For example, the *verb* is taken as input case *no 13*, and consequently, *subject* is taken as *no 12*, *object1* is taken as *no 14*, and *object2* is taken as *no 15*.

10.3.4 Proving the Presupposition

For a phrase to be instantiated, its presupposition needs to be satisfied by the context. Rules and facts of the domain support the proof. For example, consider **paragraph1** below:

```
Mary wanted to buy a car.  
She took it up with her dad.
```

The presupposition of **phrase5** (x take up Y with Z) is:

```
((head authority)  
(class soc-relation)  
(high ?x)  
(low ?y))
```

Figure 10.20: The Presupposition of Phrase5

A function call is issued to prove the proposition:

```
(prove target facts rules binding-list)
```

where the arguments of the function call are:

- (1) *target* is the presupposition itself.
- (2) *facts* is the list of concepts in the context. The fact supporting this actual proof is:

```
fact1:  
((head family)  
(class soc-structure)  
(parent harold.4)  
(actor jenny.1))
```

Figure 10.21: A Fact Found in the Context

- (3) *rules* is the list of rules in the database. The rule actually supporting the proof is:

```

rule1:
  ((head rule)
  (IF
    ((head family)
     (class soc-structure)
     (parent ?p)
     (person ?q)))
  (THEN
    ((head authority)
     (class relation)
     (high ?p)
     (low ?q))))

```

Figure 10.22: A Rule Pertaining to Authority

Rule1 encapsulates general common knowledge of people's relationship.

- (4) *binding-list* is the list of variables already bound through the unification of the pattern. At this point, the binding list is:

```
((x john.1) (y mary.2) (z goal.3))
```

Figure 10.23: The Binding List Input to the Proof

Proving proceeds in *backward chaining*. The actual chain is trivial:

```
presupposition --> rule1 --> fact1
```

Proving phrase presupposition usually does not involve a lengthy chain, such as required for example in planning.

10.3.5 Instantiating the Concept

The *binding list* created as a result of (a) unifying the pattern, and (b) proving the presupposition, is used in instantiating the concept of the phrase. In our example (parsing the sentence she took it up with her dad), the binding list is:

```
((x jenny.1) (y goal.3) (z harold.4))
```

Figure 10.24: The Input Binding List

The uninstantiated concept is:

```
(concept
  ((head auth-appeal)
   (class event)
   (from ?x)
   (obj ?z)
   (to ?y)))
```

Figure 10.25: The Uninstantiated Concept

The concept instantiated using the binding list above is:

```
(concept
  ((head auth-appeal)
   (class event)
   (from jenny.1)
   (obj goal.3)
   (to harold.4)))
```

Figure 10.26: The Instantiated Meaning

This instantiated concept is taken as the meaning of the phrase.

Merging Concepts: A problem arises with phrases which do not actually instantiate an additional concept, but rather *modify* an existing phrase. This is the case with *modal* verbs and modifiers in general. The problem is to specify and execute this modification through the *unify-instantiate* discipline. The solution is described by an example, `phrase3` (given in Section 5.3 above). The uninstantiated concept is:

```
(concept
  (*merge* ?z
    ((actor ?x)
     (mode initiated))))
```

Figure 10.27: The Uninstantiated Concept

where `?z` is the modified concept and the other expression is the modifier. The binding list is:

```
((z ((head running-for-fitness) (x john.1))
    (class soc-activity))
```

Figure 10.28: The Binding List

this is the *modified* concept

```
((head running-for-fitness)
 (class soc-activity)
 (actor john.3)
 (mode initiated))
```

Figure 10.29: The Merged Concept

10.3.6 Juxtaposing Phrases

Complements such as prepositional phrases require a special treatment. Consider the role and the meaning of the prepositional phrase in the following three sentences.

```
s1:      She took up jogging with her dad.
s2:      She learned swimming with a tutor.
s3:      She took up the issue with her dad.
```

In *s1* the complement means "she was *accompanied* by her dad". In *s2* the complement has a different meaning: "she was *assisted* by the tutor". In *s3* the complement is actually an integral part of the pattern. The selection of the appropriate meaning of the complement, relies on the concept of the main verb.

Phrase Interaction: an Example

We show complement interaction through unification, in regard to the following paragraph:

paragraph6:

```
Jenny wanted to go jogging.
She took it up with her dad.
```

Four phrases reside on the *active-phrase-stack* when the word `with` is encountered.

- P1: X take Y, (as in take her to school)
- P2: X take up Y, (as in take up an offer)
- P3: X take up Y, (as in take up jogging)
- P4: X take up Z with Y, (as in take it up with an authority)

At this point two phrases involving `with` are triggered. The representation is given below:

P5:

```
((comment (?x ?v:verb WITH ?y))
 (pattern
  ((subject ((concept ?o)
             (restriction ((class person))))))
  (verb      ((root ?v)
             (prase-concept ?pc)
             (restriction ((class soc-activity))))))
  (object    ((marker WITH)
             (restriction ((class person))
                          (concept ?p))))))
 (concept
  (*merge* ?pc
   ((company ?p))))))
```

Figure 10.30: With a Companion

P6:

```
((comment (?x ?v:verb WITH ?y))
 (pattern
  ((subject ((concept ?o)
             (restriction ((class person))))))
  (verb      ((root ?v)
             (prase-concept ?pc)
             (restriction ((class soc-activity))))))
  (object    ((marker WITH)
             (restriction ((class person))
                          (concept ?p))))))
```

```

(concept
  (*merge* ?pc
    ((assist ?p))
  (presupposition
    (head professional)
    (class role-theme)
    (actor ?p)
    (activity ?pc))))

```

Figure 10.31: With an Assistant

The interaction is by unification between: (a) A complement and (b) a main verb phrase, for each one of the phrases on the active-phrase-stack. The unification involves four steps:

- (1) Check juxtaposition (the phrases do not overlap).
- (2) Check semantic conditions.
- (3) Merge the concepts of both phrase.

Each one of these steps is detailed below.

(1) An interaction does *not* take place *if* the complement is part of the phrase itself. Thus, P4 does not interact with either P5 or P6 (since the complement is part of P4).

(2) Conditions are placed on the concept of the main phrase. The concept of the main phrase is accessible (by unification) through the slot *phrase-concept*, filled in by the variable *?pc*. In interacting with P1, for example, *?pc* gets bound to:

```

((head ptrans)
 (class act)
 (actor ?x)
 (obj ?y)))

```

Figure 10.32: The Phrase-Concept in Case of P1

Conditions can be expressed in two ways:

- (a) *restriction* is a proposition to be satisfied by *?pc*. This condition is actually met in the interaction of P5 with P3, where the concept bound to *?pc* is:


```
((head running-for-fitness)
 (class soc-activity)
 (actor john.3)
 (mode initiated))
```

Figure 10.33 The Phrase-Concept in Case of P3

satisfying the restriction (class soc-activity).

- (b) *presupposition* is a proposition to be proved which might include ?pc as an argument. Unless Jenny's father is known to be a jogging coach, the presupposition is not satisfied by any phrase.
- (3) Thus, the interaction is successful only with P3, for which the *concept* of P5 is instantiated. Since this is a **merge**, the concept bound to ?pc is augmented. The resulting concept is:

```
((head running-for-fitness)
 (class soc-activity)
 (actor john.3)
 (mode initiated)
 (company harold.4))
```

Figure 10.34: The Merged Concept

10.4 CONCLUSIONS

Thus the parser maintains multiple interpretations in the course of parsing a sentence. The set of active interpretations grows as ambiguities are encountered and shrinks as certain interpretations are ruled out.

Interpretations are generated as a cross product of:

- (1) multiple referents (e.g.: it)
- (2) multiple phrases (e.g.: take)

Interpretations are pruned out through:

- (1) phrase restrictions
- (2) phrase presupposition

(3) phrase juxtaposition

Eventually, it is desirable that only a single interpretation is left out. However, the parser *cannot* guarantee the uniqueness of the outcome of the parse. For example if the sentence is:

Jenny wanted to go jogging.
She took it up with her dad.

Then there are two interpretations which cannot be resolved by the program.

Chapter 11:

Strategies in Learning

In previous chapters we presented the learning scheme and the parsing mechanism. We showed (a) how single phrases are extracted from episodes, and (b) how an entire hierarchy is upgraded by generalization and specialization. Those schemes are general and could be applied in any domain. We have not yet shown the rules used in those schemes, which are specific to the linguistic domain.

In this chapter we detail the strategies which facilitate the learning process. These strategies are specific to a language learning program, and account for its inductive behavior. In order to illustrate the use these rules, we first outline a process model in which rules are applied, and then we detail the rules themselves.

11.1 FIGURATIVE PHRASE ACQUISITION: A PROCESS MODEL

In this section we describe a process model of figurative phrase acquisition, driven by learning strategies. Consider the situation in which the figurative phrase throw the book at somebody is first encountered.

User: The mobster eluded prosecution for years.

 Last month, they threw the book at him
 for income-tax evasion.

RINA: The prosecutor propelled a book at him?

User: No. A judge threw the book at him.

RINA: The judge threw the book at him. He found him guilty.

And later on:

User: The dean of the school threw the book at John.

RINA: He punished him.

Process 1: A Day in Court

There are three stages in the acquisition process: (1) Applying the literal interpretation; (2) Acquiring the figurative phrase; (3) Generalizing the new phrase beyond the specific context.

11.1.1 Literal Interpretation

In the absence of the appropriate phrase in the lexicon, RINA utilizes other available knowledge sources, namely (a) the literal interpretation and (b) the context. The literal interpretation is given by the phrase:

pattern:	X:person	throw	Y:phys-obj	<at Z:person>
concept:	(act	(propel	(actor X)	
			(object Y)	
			(to	(location-of Z)))
	(purpose	(thwart	(goal p-health)	
			(goal-of X))	

Figure 11.1: Propel a Phys-Obj

This phrase describes propelling an object in order to hit another person. Notice that no presupposition is specified. General phrases such as *take*, *give*, *catch*, and *throw* do not have an expressed presupposition since they can be applied in many situations.*

The literal interpretation fails by plan/goal analysis. In the context laid down by the first phrase (prosecution has active-goal to punish the criminal), "propelling a book" does not serve the prosecution's goals. In spite of the discrepancy, RINA spells out

* Notice the distinction between *preconditions* and *presupposition*. While a precondition for "throwing a ball" is "first holding it", this is not part of the phrase presupposition. Conditions which are implied by common sense or world knowledge do *not* belong in the lexicon.

that interpretation above with a question mark, The prosecutor propelled a book at him? to notify the user about her current state of knowledge, and the fact that a discrepancy has been detected.

11.1.2 Learning by Feature Extraction

In constructing the new hypothesis, the program must extract the relevant features from the given episode.

- (a) The initial phrase *presupposition* is taken as the entire *trial* script.
- (b) The *pattern* is extracted from the sample sentence.
- (c) The *concept* is extracted from the script.

In extracting either the pattern or the concept, the problem is to distinguish between features which are relevant and should be taken in as part of the phrase, and features which are irrelevant and thus should be left out. Moreover, some features should be taken as is, where other features must be abstracted before they can be incorporated.

11.1.3 Forming the Pattern

Four rules are used in extracting the linguistic pattern from the sentence:

Last month, they threw the book at him for income-tax evasion.

- (1) Initially, use an existing literal pattern. In this case, the initial pattern is:

pattern1: X:person throw:verb Z:phys-obj <at Y:person>

- (2) Examine other cases in the sample sentence, and include in the pattern cases which could not be interpreted by general interpretation. There are two such cases:

- (a) Last month could be interpreted as a general time adverb (i.e.: last year he was still enrolled at UCLA, the vacation started last week, etc.).
- (b) For income-tax evasion can be interpreted as a *element-paid-for* adverb (i.e.: he paid dearly for his crime, he was sentenced for a murder he did not commit, etc.).

Thus, both these cases are excluded.

- (3) Variablize references which can be instantiated in the context. In this case X

is the Judge and Y is the Defendant. They are turned into as variables, as opposed to the other case:

(4) Freeze references which cannot be instantiated in the context. No referent is found for the reference *the book*. Therefore, that reference is taken as a frozen part of the pattern instead of the case *Z:phys-obj*.

The resulting pattern is:

pattern25: X:person throw:verb <the book> <at Y:person>

11.1.4 Forming the Concept

In selecting the concept of the phrase, there are four possibilities, namely the events shown in Figure 4.1. The choice of the appropriate one among these four events is facilitated by linguistic clues. As opposed to the phrase *they threw the book to him* which implies cooperation between the characters, the phrase *they threw the book at him* implies a goal conflict between the characters. *At* implies not taking acknowledgement protocol into consideration. E.g., *x throws the rock to y* implies that *x* catches *y*'s attention, and gets acknowledgement for *y*'s receipt of the rock. On the other hand, *x throws the rock at y* implies that *y* may not be aware or ready to receive the rock. This analysis applies also to *talk at* vs. *talk to*, etc. Since this property is shared among many verbs, it is encoded in the lexicon as a *general* phrase:

pattern: X:person ?v:verb Y:physobj <at Z:person>
concept: (propel
 (actor X)
 (object Y)
 (to Z)
 (mode no-acknowledge))

Figure 11.2: *propel at*, a General Phrase

Notice that rather than having a specific root, the pattern of this phrase leaves out the root of the verb as a variable. From lack of acknowledgement, a goal conflict may be inferred.

```
(thwart-goal
  (goal p-health)
  (goal-of Z))
```

Using this concept as a search pattern, the "punishment-decision" is selected from *\$trial*. Thus, the phrase acquired so far is:

```
pattern:      X:person throw <the book> <at Y:person>
concept:      (select-plan
              (actor X)
              (plan (thwart-goal
                    (goal p-goal)
                    (goal-of Y))))
presupposition: (head trial)
                (judge X)
                (defendant Y)
```

Figure 11.3: The Acquired Phrase

11.1.5 Phrase Generalization

Although RINA has acquired the phrase in a specific context, she might hear the phrase in a different context. She should be able to transfer the phrase across specific contexts by generalization. RINA generalizes phrase meanings by analogical mapping. Thus, when hearing the sentence below, an analogy is found between the two contexts.

```
The third time he caught John cheating in an exam,
the professor threw the book at him.
```

The trial-script is indexed to a general *authority* relationship. The actions in a trial are explained by the existence of that relationship. For example, by saying something to the Judge, the Defendant does not dictate the outcome of the situation. He merely informs the Judge with some facts in order to influence the verdict. On the other hand, by his decision, the Judge does determine the outcome of the situation since he presents an authority.

Three similarities are found between the trial and the scene involving John and the professor: (a) The authority relationship between X and Y; (b) A law-violation by Y; (c) A decision by X.

Therefore, the phrase presupposition is generalized from the specific trial-script into the general *authority-decree* situation which encompasses both examples.

11.2 THE STRATEGIES

Phrases are extracted from input sentences by failure-driven strategies. For example, consider the sentence:

Last month, they threw the book at Al Capone.

The scope and the variability of the yet unknown pattern must be determined, and a hypothesis about the new pattern must be constructed as follows:

X:person throw:verb <the book> <at Y:person>

By applying learning strategies, an instance sentence is converted into a lexical pattern which can be applied in other instances.

Strategies are applied not only in forming an initial hypothesis, but also in upgrading an existing hypothesis. For example for the phrase David took on Goliath, the initial hypothesis is:

X:person take:verb <on Y:person>

This incorrect hypothesis is upgraded when the user provides the second example:

He took him on.

This sentence does not match the pattern above, and consequently a discrepancy is detected. The hypothesis must be upgraded by analyzing the discrepancy. Thus, learning is an on-going process motivated by detection of failures.

11.2.1 Extracting a New Pattern

The user's first example is the following sentence:

User: Last month, they threw the book at him for tax evasion.

From this sentence the program must extract a pattern. There are two issues in extracting the pattern.

Scope: Which parts of the sentence belong in the pattern and which parts do not?

Variability: Among the included parts, which elements should be *frozen* (taken verbatim) and which elements should be variabilized?

(1) **General Complements:** The prepositional phrase *for tax evasion* is taken as a general phrase. There are several general phrases for *for* in the lexicon:

For a time unit, as in *for a short period of time*.

For a purpose, as in *he worked hard for his degree*.

For a matching act, as in *he was punished for his crimes, or he was rewarded for his bravery*.

For a beneficiary, as in *he worked hard for his family*.

for in the input sentence is assumed to belong in the third class, where *tax evasion* is taken to be the law violation Capone was sentenced for.

modifier:	<i>for tax evasion</i>
lexical phrase:	<i>for A:act (for a matching act)</i>
action:	<i>exclude a general modifier</i>

Figure 11.4: Excluding a prepositional phrase

Similarly *last month* is also taken as such a general modifier.

modifier:	<i>last month</i>
lexical phrase:	<i>last T:time-unit</i>
action:	<i>exclude a general modifier</i>

Figure 11.5: Excluding a modifier

Learning is Relative to the Lexicon

An apparent weakness of this strategy is its dependence on the current lexicon contents. It does not sound right that learning should depend on the particular contents of the lexicon. However, this is the main strength of our theory. Learning *must* be relative to the lexicon since by learning the lexicon itself is modified. Thus learning is a *negative feedback* mechanism. This idea is further explained in Section 11.5.

Specific Attachments: At him, on the other hand, cannot be interpreted as such a general phrase. There are two general interpretations for at:

at time unit, as in at noon.

at place-of-activity, as in she works at school.

Since these two phrases do not match the current case, the phrase at him is included in the pattern.

modifier :	at him
lexical phrase :	none
action :	include a specific modifier

Figure 11.6: Including a prepositional phrase

Similarly, the book cannot be interpreted as a general modifier.

modifier :	the book
lexical phrase :	none
action :	include a direct object

Figure 11.7: Including the direct object

Thus, last month, and for a tax evasion are left out, while the book and at him are taken in.

Therefore, *References which can be resolved in the context are turned into variables.* On the other hand, *references which cannot be resolved are kept as literals.*

Variabilized Elements: The reference him is resolved by the parser as Al Capone. Thus him is turned into a variable.

old-reference: him

referent: Capone.104
action: variabilize resolved reference
new-reference: Y:person

Figure 11.8: Turning a reference into a variable

Frozen Elements:

The reference **the book** cannot be resolved in the context, as a book was not mentioned in the discourse. Therefore the reference is taken as a frozen unit.

old-reference: the book
referent: none
action: freeze unresolved reference
new-reference: the book

Figure 11.9: Freezing a reference

Similarly, references are frozen in other examples:

He kicked **the bucket**.
They buried **the hatchet**.
They carried **the water**.
His father laid down **the law**.
He went to **Davey Jones' Locker**.

The references in all these phrases cannot be resolved in their corresponding contexts. However, consider the same reference in a different context:

The judge was carrying a heavy book.
He threw the book at Al.

In this context, since a book has been mentioned, the reference is not taken as a frozen element, rather it is interpreted as a variable.

Resolving a Reference by a Script: Referents must not be given explicitly in the context. They can be derived indirectly, as in the following text:

Al went on trial.
The judge threw the book at him.

A judge has not been mentioned explicitly in the context, however the reference the judge is taken as a role holder in the *trial script*. This strategy is based on a parsing strategy first introduced by Cullingford [Cullingford78] in script application.

old-reference: the judge
referent: judge.106 in trial.105
action: variabilize reference resolved in a script
new-reference: X:person

Figure 11.10: Variabilizing a reference resolved in a script

This is in contrast to a the following paragraph (which appeared as a newspaper title):

Judge Wins the California Lottery.

Here, a trial script has not been instantiated. In fact, it is merely a coincidence that the winner in the *lottery script* is a judge.

11.2.2 A Phrase-Modification Cycle

To illustrate pattern-modification rules, we introduce here a second process model, in which the program acquires the phrase to take it up with:

User: Greg needed money for his trip.
 He took it up with his dad.
RINA: He took the money from his dad?

User: No. He took up **the problem** with his dad.
RINA: He solved the problem with his dad? *

 He took it up?

User: No. He discussed the problem with his dad.

RINA: He appealed to an authority?
 He took it up with his dad?

User: Ok.

Process 2: The Trip to Palm Springs

* Notice that although the user's example looks as if it is a "give away", RINA fails in zeroing on the appropriate meaning. *Take it up with* does not mean "solve a problem". How could the program miss such a straightforward clue? See Section 11.2.5 for explanation

Notice the stages in the sequence:

- (1) **Literal interpretation:** RINA first attempts the use of an existing entry: "he took the money from his dad".
- (2) **New Hypothesis formation:** the program extracts the wrong meaning for the phrase, assuming that to take it up with means "to solve a problem".
- (3) **Hypothesis correction:** At this stage the program forms the correct meaning.

Compare the initial and the second hypotheses as they are given in the figure below:

before:	pattern:	X:person take Y:possession
	concept:	X atrans Y
after:	pattern:	X:person <take up> Y:goal-situation
	concept:	X know a plan for Y

Figure 11.12: The hypothesis before and after

The sentence which motivates the modification is given below:

User: He took up **the problem** with his dad.

Three discrepancies are encountered by comparing the initial hypothesis with the new input:

- (1) Up in the sentence is extraneous.
- (2) with his father is extraneous.
- (3) The problem does not refer to a *possession*.

Accordingly, the following strategies are applied.

Identifying a Verb Modifier: Up appears in the lexicon as a verb modifier which defines a vertical positive line (.e.g., he looked up, he put it up on the shelf, etc.). However, this interpretation does not fit the meaning above ("his dad helped him"), thus the modifier is taken as a mandatory part of the phrase.

old-hypothesis:	X take Y
input:	he took up the problem (with his father)
discrepancy:	extraneous particle up

action: include particle as a verb modifier
new-hypothesis: X <take up> Y

Figure 11.13: Including a verb modifier

Identifying a Case Marker: *with* appears in the lexicon in four phrases, as follows:

- (1) *Act with instrument*, as in he dug a hole with his fingers.
- (2) *Act with an assistant*, as in she learned tennis with a pro.
- (3) *Social-act with a company*, as in he walked with his brother.
- (4) *Communication-act with a partner*, as in he discussed it with his
atives.

The second lexical interpretation fits well in the hypothesized meaning above ("his father helped him"), and therefore this element is taken as a general complement—and is not included in the pattern.

old-hypothesis: X <take up> Y
input: he took up the problem with his father
discrepancy: extraneous phrase **with Z**
lexical-phrase: X V:act <with Z>
action: exclude a modifier
new-hypothesis: X <take up> Y

Figure 11.14: Excluding a general complement

A Class Discrepancy: The class discrepancy motivates the following strategy:

old-hypothesis: X take Y:possession
input: he took (up) **the problem** (with him)
discrepancy: referent class is not subsumed by reference class
action: change reference class
new-hypothesis: X take Y:goal-situation

Figure 11.15: Changing a reference class

Accordingly, the class specifications are modified.

Correcting a Particle's Role: The process model below is given in order to introduce another strategy involving class specifications:

User: David took on Golaith.

RINA: He took on him. He won the fight.

User: No. He decided to fight. He took him on.

RINA: He accepted the challenge. He took him on.

Process 3: David and Goliath

In acquiring the phrase *take on*, RINA has initially assumed that *on* is a case marker. This is corrected by the user's sentence:

User: He took him on.

The hypothesis before and after the correction is given below.

before:	pattern:	X:person take <on Y:person>
	concept:	X win a fight with Y
after:	pattern:	X:person <take on> Y:person
	concept:	X decide to fight Y

Figure 11.16: The hypothesis before and after

In making the first hypothesis, the program classified the phrase incorrectly along with phrases such as *he nibbled on the corn*, *he hit on her*, *he turned on him*, etc. The user's sentence causes the detection of *two* discrepancies:

X take	<on Y>
he took	him on

Figure 11.17: Comparing the Hypothesis with the Example

(a) *on* is absent from its designated location as a case marker.

(b) *on* appears extraneously after the object.

This combination of discrepancies motivates the following strategy:

old-hypothesis:	X take <on Y>
example:	he took him on
discrepancy:	extraneous particle on
	missing case marker on
lexical-phrase:	<V:verb on>
action:	change role of particle

new-hypothesis: X <take on> Y

Figure 11.18: A case marker becomes a verb modifier

This hypothesis is further generalized in later iterations.

11.2.3 Modifying Class Specifications

In this section we describe the strategies pertaining to class specifications. The dilemma presented in this section is when to modify and when not to modify class specifications.

Indirect References: Class specifications are problematic. They look nice on paper but in real parsing they never work out properly. Consider the following set of sentences:

John listened to **the sound of his car** .
He listened to **Mozart** before he went to sleep.
He listened to **the kids in the backyard** .
He listened to **his parents** .

The marked reference attains a different *class* in each instance, although the lexical *class specifications* for *listen* require a unique class, *sound-type*, as shown in the following phrase:

P0: **pattern:** X:person listen Y:sound-type
 concept: X attend X's ears to Y

Figure 11.19: The phrase for *listen*

However, in light of the diversity of types in the sentences above, how could we specify a fixed class in the lexicon? Alternatively, consider a learning algorithm which modifies lexical class specifications each time it encounters a class discrepancy. In that case, the lexical representation would appear pretty fluid. The phenomenon above is called *indirect reference* and is described by Fauconnier [Fauconnier85] and Hershkovitz [Hershkovits85] In general, people avoid the use of lengthy descriptions when shorter ones suffice. Thus a person would not say:

I listened to the air-waves created by a needle of
a record-player in playing a record of music
written by **Mozart** before going to sleep.

A typical *short cut* would be:

I listened to **mozart** before going to sleep.

Similarly, the following sentences are short cuts:

I ran into my neighbor on a red light.
She likes horses.

They both stand for longer sentences which have been reduced:

My car ran into my neighbor's car.
She like's horseback riding. or perhaps:
She likes eating horses.

Indirect references pose a difficult problem in forming a discrepancy-based learning strategy. The problem is: when a difference exists between the nominal class and the actual class, when is class modification appropriate, and when it is not appropriate?

A Class Discrepancy: Assume that the current hypothesis for the phrase *X take up Y with z* is as given below (see Process Model 1).

X:person take Y:possession

The user corrects this hypothesis by providing the following sentence:

He took up the problem with his dad.

Due to this sentence, the program changes the class of the variable Y from a possession to a goal-situation.

old-hypothesis: X take Y:possession
input: he took (up) **the problem** (with him)
discrepancy: referent class is not subsumed by reference class
action: change reference class
new-hypothesis: X take Y:goal-situation

Figure 11.20: Changing a reference class

Generalizing a Reference Class: A similar discrepancy occurs with the phrase *take on*. as shown in the Process Model 2. The current hypothesis is given below as:

X:person <take on> Y:person

This hypothesis fails in handling the next example:

John took on a new job.

Since a new job is not a reference to a person, a modification is required. Thus, according to the strategy in Figure 11.20, should the class specifications be modified to accommodate for the new reference as show below?

X:person <take on> Y:task

The learning situation here is different than the previous one. In take it up with, there was a single episode whose interpretation required a correction: "no, it is the problem, not the money that he took up with his father". In contrast, here there are two episodes which must be encompassed by a general concept. A generalization is required to cover both *person* and *task*. The generalization is found based on the underlying contexts. In both examples, the referent presents a *challenge* to the actor. Therefore, the pattern is generalized:

X:person <take on> Y:goal-situation

The specific goal-situation, namely *challenge*, is not given in the pattern, rather it is imposed as an underlying presupposition.

old-hypothesis: X <take on> Y:person
input He took on a new job.
discrepancy: two referents subsumed by a common class
action: generalize the class
new-hypothesis: X take Y:goal-situation

Figure 11.21: Finding a common generalization

Indirect Reference: Consider the lexical phrase P0:

P0: **pattern:** X:person listen <to Y:sound>
 concept: X attend X's ears to Y

The input sentence is:

John listened to Mozart.

In this case, should the strategy in Figure 11.21 be applied to update the class specifications from *sound* to *person*? In this case, no modification should occur since the reference Mozart is *dereferenced* indirectly:

mozart --> name --> person --> musician --> record --> sound

Since the ultimate referent is of type *sound* the modifying strategy is not applied. The rule is:

old-hypothesis: X listen <to Y:sound>
input John listened to Mozart.
discrepancy: referent resolved indirectly
action: none
new-hypothesis: X listen <to Y:sound>

Figure 11.22: Finding an indirect referent

This strategy accounts for cases where a referent is found indirectly.

11.2.4 Learning General Phrases

So called "optional" modifiers have been problematic in any lexical representation. Consider for example, the phrase in the sentence:

John sold the car to Mary for \$1500.

What is the representation of the phrase for *sell*? In particular, does it include the modifiers *for* and *to*? One possible representation for this phrase and two similar phrases is given below:

P1: U:person **sell** V:possession <to X:person>
<for Y:possession>
P2: U:person **buy** V:possession <from X:person>
<for Y:possession>
P3: U:person **purchase** V:possession <form X:person>
<for Y:possession>

* This way linguistic systems such as [Wilensky84, Wilks75] represent phrase complements. In fact, in those systems, the parts which are not mandatory are marked by "optional".

Three problems are identified regarding the appearance of the modifier <for Y:possession> in all these phrases:

- (1) This modifier is not mandatory. For example, P1 can appear as in I sold my car, without the modifier.
- (2) This representation is not economical, since the same modifier is repeated in the same form, carrying the same meaning in all three instances.
- (3) This representation does not allow processing of phrases which do not appear explicitly on the list.

DHPL's solution is to maintain a single generalized phrase for *for* which is applicable in P1-P3, as well as in many other phrases:

P4:
pattern: X:person V:atrans-act Z:possession
<for Y:possession>
presupposition: persons X and U are involved in a transaction T
concept: within T, Y is exchanged for Z

Figure 11.23: The phrase for *for*

In addition to the three problems above, this representation also solves the problem of representing word senses in the lexicon. Phrase P4 conveys one sense of the word *for*. Next, consider the impact of this generalized phrase on the learning process.

The General Phrase P4 Already Exists: Consider the case in which the program encounters the new word *to trade*, in the following sentence:

- (1): John traded the book for a journal.

How can a program guess the meaning of the word *trade*? An important clue is provided by the modifier *for a journal* as given in phrase P4, which suggests that one possession was exchanged for another. However, in acquiring the phrase *for trade*, the modifier should be excluded from the new pattern since it can be derived by P4. The applied strategy, is the same as in Figure 11.23:

modifier for a journal
lexical phrase: P4- for a matching possession
action: exclude a general modifier

Figure 11.24: Excluding the general modifier

P7 for sell:

P7:

pattern:	X:person	sell	Y:possession	
				<for Z:possession>
presupposition:	X is involved in a transaction T			
concept:	X exchanges Y for Z			

Figure 11.28: The new phrase for *sell*

At this point, by detecting the similarity between the features of P5 and P6, the model creates and adds to the lexicon the new general phrase P4 above. The strategy applied in the generalization is given as follows:

If two new phrases PA and PB have a common concept and a common presupposition, and they also have a common modifier, then

- (1) Create a new general phrase PC by changing the verb into a variable, and maintaining the modifier itself.
- (2) Link PA and PB to the general phrase PC.

Notice that although a general phrase PC could account for the modifier M in both PA and PB, M is not removed from PA and PB themselves. Thus, a small hierarchy is added on to the lexicon, as shown in Chapter 6.

11.2.5 Extracting Phrase Concepts

So far we have described methods for pattern formation. By giving three examples, we describe a general strategy for concept formation. The general problem in guessing the meaning is that the context includes many concepts, some appropriate and some inappropriate for meaning formation. The task is to identify and to extract one of these concepts as the phrase meaning. The general strategy is to select the concept which best matches the given linguistic clues.

Story Points: The story of David and Goliath is given in terms of *story points* [Wilensky82]. Story points are used to memorize past events, indexing them by high-level knowledge structures. Learners who are asked to reconstruct the biblical

story* cannot tell the exact details of the fight between the characters. However, they all remember two points:

- In spite of his physical inferiority, David decided to fight Goliath.
- In spite of his physical inferiority, David won the fight.

These story points serve as the context in learning the new phrase.

The second factor in learning a phrase is the senses conveyed by the single words themselves. In particular, the word *on* has a meaning "a positive state". Thus, when the new phrase *take on*, is encountered in the presence of the story points given above, the second story point which means "David won the fight" is selected. The positive outcome of the fight as conveyed by that point best matches the meaning of the particles.

Scriptal Events: The phrase *throw the book at somebody* is encountered in the context of the *trial-script*. As shown in Chapter 4, this script describes the events taking place in court. Three parties are involved: the judge (J), the defendant (D), and the prosecutor (P). The script is given as a sequence of events:

- (1) P communicates to J in order for J to punish D.
- (2) D communicates to J in order to avoid the punishment.
- (3) D selects either one of two acts:
 - (a) punish (thwart a preservation goal of) D.
 - (b) do not punish D.

The partial concept constructed for *J throw the book at D* (in absence of the figurative phrase) is:

J thwart a preservation goal of D

This particular partial concept was constructed due to the sense of the word *at* (as described in Section 11.1). Event 3a is selected due to its match with the partial concept obtained by the literal interpretation.

General Planning Knowledge: Another aspect of world knowledge being used in learning is knowledge of planning behavior. As shown in Chapter 4, there are two

We consider here only learners who had heard the story earlier.

kinds of planning knowledge: abstract strategies, and specific strategies. For example, consider planning aspects in the context created by the following sentence.

Greg needed money for his trip.
He took it up from his dad.

The major goal is for Greg (G) to go on a trip (G1). This goal G1 has an active subgoal, that G possess money (G2). G2 on its part has four potential subgoals: earn money (G3), borrow money (G4), draw money from bank (G5), and steal money (G6). This specific-planning level is assumed by the learner in his first hypothesis:

He took the money from his dad.

Here subgoal G4 is selected, assuming that he took it up with his dad means "he borrowed the money from him".

Next, the user replies:

No. He took **the problem** with him.

This causes the program to shift its search to abstract-planing. In fact, it seems that the user gave away the entire meaning. How could the program make an error in spite of this explicit example? The reference *the problem* points to a goal situation (rather than a physical object as first hypothesized by the learner). Moreover, a *problem* is represented as follows:

A Problem:

active goal G of person A for which a feasible plan P is not known to A.

Accordingly, one planning act associated with such a goal situation is

A find a plan P for achieving goal G.

This act is selected as the meaning of the phrase, and is spelled out by the program:

He solved the problem with his dad.

(Here *with his dad* is just a complement and not a mandatory part of the phrase). Similarly, the next iteration with the user involves a planning act. One possible action in face of a problem is consulting with an authority about the problem.

A communicate to B (an authority) about G.

This planning act is selected by the learner in response to the user's example:

He discussed it with his dad.

Finally, after three iterations, the resulting phrase is:

pattern: X <take up> Z:goal-situation
<with Y:person>
presupposition: Y present an authority to X
concept: X communicate about Z to Y

11.3 CONCLUSIONS

We have presented learning rules for creating and modifying lexical phrases. We have shown how these rules, or learning strategies, take part in three process models where unknown phrases are acquired. In each case, learning is a gradual process of refinement and generalization.

In forming patterns, the strategies were motivated by detecting syntactic discrepancies. The entire body of rules is motivated by one basic intuition: when a discrepancy is encountered, correct the source of that discrepancy.

In forming concepts, we have shown one general strategy. Two elements are given:

- (a) the context
- (b) the single words comprising the new phrase

However, each one of these two elements individually cannot yield the desired meaning, since:

- (a) The context contains many concepts
- (b) The single words construct only a partial meaning.

Therefore, the rule is to select the concept which best matches the *partial meaning* constructed by the single words.

Chapter 12:

Summary, Conclusions and Future Work

We have presented a dynamic lexical structure called DHPL, and we described a learning model embodied by the program RINA.

12.1 CONCLUSIONS

DHPL (Dynamic Hierarchical Phrasal Lexicon) supports language acquisition by four features:

Phrases: The lexicon contains entire phrases, accounting uniformly for an entire *range* including productive as well as non-productive phrase.

Hierarchy: The lexicon organizes in a hierarchy phrases ranging from specific "lexical entries" at the bottom, to general "grammar rules" at the top.

Lexical Presupposition: Contextual conditions are incorporated into the lexicon through lexical presuppositions. Presuppositions account for disambiguation in parsing, and for phrase selection in generation.

Integration of Syntax and Semantics: Each phrase specifies a *relation* (in the logical sense) between syntax and semantics. Thus, the question whether any lexical feature is syntax or whether it is semantics, becomes insignificant. For example, consider

thematic roles for a phrase such as *promise*. Are they syntactic or are they semantic? They can be viewed as either.

In the program RINA we have shown three results in language processing:

Coping with Lexical Gaps: The hierarchical structure of the lexicon enables parsing of text even when certain lexical elements are unknown. A partial meaning for the text, which serves as an initial hypothesis, is formed by applying *general* knowledge when *specific* knowledge is missing.

Using Lexical Clues: In learning meanings of phrases we have used "linguistic clues". For instance, the word *at* in the judge threw the book at Al, supports the learning process of that idiom. What is the justification for drawing inferences from apparently vague senses of words? In making the lexicon amenable as a linguistic database, from which inference rules can be drawn, we have *systematically* organized words in a hierarchy, representing words such as *at*, *to*, *around* and *away*. Thus, the use of linguistic clues per se is not inappropriate; however, all linguistic clues used in a reasoning system, must be drawn from a well-organized lexicon.

Knowledge Propagation through Generalization and Specialization: Hierarchy is a precondition for learning by generalization. Through the hierarchical scheme, there are two ways of propagating knowledge: First, *bottom-up*—from instantiated episodes up towards specific phrases, and even higher to generalized word senses. Second, *top-down*—generalized word senses are propagated down for prediction of new specific phrases. In both cases, effective learning depends on the existence of a well refined hierarchy.

Any linguistic system must accommodate not only for spanning a static language, but also for augmenting the original linguistic system itself. In DHPL we have shown how, for a variety of linguistic features, the lexicon itself can be augmented through linguistic experiences. Thus we have accomplished a *dynamic* linguistic behavior.

12.2 STATUS OF IMPLEMENTATION

The computer program RINA is written in T [Rees84], which is an object-oriented LISP dialect. RINA has been developed using GATE [Mueller87], a graphic artificial-intelligence development environment. RINA is an ongoing project, whose current status is described by these figures:

- (1) RINA's lexicon has more than 200 phrases.
- (2) There are 35 learning rules.
- (3) RINA can learn 10 different types of phrases.
- (4) RINA has about 10,000 lines of code.
- (5) Parsing a sentence on an APOLLO workstation takes about 40 seconds, in the presence of 5 competing interpretations.
- (6) Each step in a learning session takes about 1 minute. Most of this time is spent by the theorem prover.
- (7) RINA uses in representing world knowledge: 5 goal situations, 3 scripts, 4 interpersonal relations (each of these structures is associated with rules), and more than 200 specific planning rules.

12.3 LIMITATIONS

The following aspects have *not* been covered by the current model.

Learning Concepts: Our model does not concern the *construction* of new semantic concepts. Learning in RINA amounts to making new associations between patterns and already existing concepts. This is obviously a serious limitation for two reasons:

- (a) Knowledge required for learning of new phrases must be hand coded. Thus, RINA falls into the same class as all current programs which *consume a lot of knowledge, and produce only a little knowledge*.
- (b) *RINA is able to refine its representation only in discrete steps*. In learning a phrase, the program shifts its hypothesis from one concept to another. It cannot refine concepts gradually.

Language Transfer: The aspect of language *transfer* has been ignored. How does knowledge of language I impact acquisition of language II? This line of research is

carried out by Mike Gasser [Gasser86b] who investigates English generation by a Japanese speaker. Gasser's research complements ours, by explaining errors stemming from different lexical organizations.

Incomplete Grammar: Only fragments of the English language have been encoded by DHPL in a principled way. The syntactic simplicity of examples given throughout this dissertation is not accidental. Our current model accounts only for a modest number of constructs. While we have dedicated effort to syntactic issues of control and thematic-role assignment, we have ignored many other constructs, such as relative clauses, and global sentence constructs.

Learning Segments of Hierarchies: We have shown in Chapter 5 how a partial hierarchy can be augmented. We focussed on a few types of phrases. Many other types, at various levels of generality have not been considered in this model. Consequently, a full hierarchy cannot be constructed by a program due to large portions which can still not be acquired programmatically.

Idioms: The current model is not capable of acquiring phrases such as in the following examples:

We do not drink supermarket water, **let alone** tap water.
LA water is full of Chlorine **as well** as other minerals.
Here is some more asbestos. **So much for** healthy water!
Water here consists of .7%, .03% and .9% of
A, B and C **respectively**.
This component is not unhealthy,
only when it interacts with Oxygen it reeks.

Each one of these phrases features syntactic, semantic and pragmatic properties which are difficult to explain. We have focused mainly on acquisition of phrases which are derivatives of verb phrases.

Generation has not been implemented in RINA in a general way. Issues such as phrase selection and phrase interaction in generation have not been addressed.

12.4 IMPORTING ARTIFICIAL INTELLIGENCE METHODS INTO LINGUISTICS

This model is an additional step in integrating computational linguistics with general artificial intelligence methodology.

The Knowledge-Based Approach: Following Wilensky [Wilensky81], we have shown how a linguistic system can be viewed as body of rules and facts. Lexical entries are given as declarative modular units, incorporating syntax and semantics.

Unification: Following Kay [Kay79], we have shown how text can be produced by general *unification*. Subsequently, general mechanisms such a PROLOG can be applied in processing text.

Hierarchy: Following Jacobs [Jacobs85b], and [Lytinen84] we have shown that hierarchy by generalization can be used in representing lexical concepts. Thus, the linguistic database can be integrated with the rest of a model's memory of concepts. This is advantageous in particular for coping with gaps in lexical knowledge.

Learning: We have shown how general learning methods [Mitchell82, Schank82, Kolodner84, DeJong86, Lebowitz86] can be applied in language acquisition. Accordingly, language acquisition has been viewed as learning of concepts in a lexical hierarchy.

12.5 THE LEARNING ALGORITHM

The learning algorithm is specified as follows:

- (a) **Input:** The input is a set of instantiated *episodes* given as *sentences* embedded in *contexts*.
- (b) **Given:** An initial *partial lexical hierarchy* is given. However, certain sentences cannot be parsed using this partial lexicon (since not all phrases are encoded at the outset).
- (c) **Given:** Full knowledge of the domain. Inference rules enable *explanation* on the one hand, and *discrepancy detection* on the other hand.
- (d) **Output:** The initial hierarchy is augmented. Sentences which could not

be parsed originally, can now be parsed.

The algorithm was presented in two parts. The basic step involves learning a phrase from a single episode. The general step involves generalizing and specializing phrases in a hierarchy.

Learning Phrases from Single Examples: How can a phrase such as *kick the bucket*, or *bury the hatchet* be acquired from a single example? We have shown how (a) the pattern is extracted from the given sentence, and (b) how the concept is extracted from the given context. A metaphor, when it exists, is used to explain *why* the words convey that particular concept.

Learning a Hierarchy: As further examples are received, phrases must be refined. Moreover, generalities must emerge from sets of similar phrases. We have shown how a full hierarchy can be generated from specific examples.

12.6 FUTURE RESEARCH

We have identified five research issues for future investigation.

Learning and Forgetting: Two related issues are system *stability* and *obsolescence*, or forgetting. Stability concerns the ease with which well-established knowledge can be modified. If the behavior of the program is too dynamic, then it might easily get thrown off by one esoteric, or incorrect use of a phrase. It is not desirable that an adult native speaker would get his lexicon ruined by listening to a second language speaker. Forgetting involves inaccessibility of unused phrases, or getting rid of incorrect hypotheses. Are incorrect hypotheses simply destroyed, or is there a more realistic model of obsolescence? These two issues involve quantitative reasoning which require implementation of strength of links and activation. These kind of problems demonstrate the limitations of a strictly *qualitative* approach, such as ours, which rely on manipulation of logical propositions, and it raises the need for *quantitative* approaches such as *connectionism* [Waltz85, McClelland86] and *spreading activation* [Anderson84, Charniak83].

Accumulation of Episodes: The lexicon retains episodes in which phrases have been encountered. For example, in learning *put somebody on the spot*, the phrase is associated with the two episodes in which it has been acquired:

- (1) Ted Koppel put his guest on the spot.
- (2) John's wife put him on the spot.

However, it is impractical to retain such episodes for phrases which are encountered in diverse situations. For example, in learning a general word such as *teach* or *drive* it is not desirable to maintain all the episodes in which they have been encountered. Thus associated episodes must disappear when there are too many of them.

Ambiguity: Both in parsing and in learning a hypothesis is selected from a set of possible interpretations. However, since the set of interpretations for each sentence is huge, we reduce its number by finding discrepancies. However, *inhibition* only cannot reduce that set to a single solution and we must address also *preference*. Consider for example the following sentence.

John needed money for his trip.
He took it up with his dad.

The interpretation preferred by a human reader is "he discussed **the problem** with his dad". However, the second interpretation "he took **the money** up" which is never selected by humans, cannot be *ruled out* by our computer model. There is nothing *logically* incorrect with that interpretation, however the first interpretation is *preferred* due to its connections to the context. We need to incorporate such a preference mechanism in our model.

Generation of Examples: We have identified a difference between generation tasks in general, where the generator describes a state of affairs in the world, and our specific task of *example generation*. In example generation, the program is required to demonstrate its own state of knowledge. For instance, one dialog given earlier proceeds as follows:

User: Greg wanted to buy a new car.
He took it up with his dad.
RINA: He took up **the car** with his dad?

The explicit reference **the car** is important since it conveys RINA's failure in acquiring the phrase. How could a program decide to generate **the car** (and not *it*) in contrast to *he* (and not *Greg*)? The research issue is: how a program or a person can test out its notion of a phrase. Examples must be generated to examine the boundary

conditions in which the phrase can still be applied. This issue has not been investigated so far.

Sound Patterns: We have ignored the impact of sound patterns [Chomsky68] on word acquisition, although we have noticed the significance of this factor in human learning. Consider for the following example:

- Reagan **pledged** to fight on,

Second-language speakers almost invariably assume that **pledge** is actually **plea**, probably due to the similar sound pattern. Accordingly they guess: he asked them to fight rather than he promised to fight. Similarly, native speakers are biased in making a guess about the following word:

- Reagan **plended** his voters to fight on.

The initial hypothesis is biased by the similarity of the sound patterns of the words **plend** and **prod**. In general it is difficult to design a computational model that accounts simultaneously for many layers of cognition.

Concept Generalization: Proliferation of knowledge is the process we try to approximate. The ubiquitous dilemma in comparing two concepts is whether a generalization exists for both, or whether they are distinct concepts. For example, consider the following sequence of examples in teaching the phrase **to take on**.

- (3) David took on Goliath.
- (4) I took on my elder brother.
- (5) I took on a new job.
- (6) We took on a new systems programmer.
- (7) This piece of paper took on the shape of a butterfly.

The second phrase can share the concept acquired for the first one, namely ?x decided to fight ?y. The third phrase; however, requires one to generalize the initial notion since it now appears as ?x accepted a challenge presented by ?y. However, can a generalization be found to encompass the fourth phrase? Notice that although a very general concept which encompasses all of the given examples *could* be found (?x has something to do with ?y), however, the effectiveness of such a generalized notion is totally diminished. Therefore, a shared concept should be sought at the appropriate level of generality.

Deviational Uses of Language: So far, the notion of lexical presupposition has not been developed according to its agreed functional definition. It is agreed that lexical presupposition presents *felicity* conditions for phrase application. When these conditions are violated, phrases sound awkward, ironic, or simply incorrect. Consider the sentences below:

- (8) We refused to let our baby stay up all night,
so he **threw the book** at us.
He yelled and screamed for hours.
- (9) My pals asked me how I got straight A's. I managed
to **explain it away** by telling them
it was a bureaucratic mistake.

In each one of these sentences, a lexical presupposition is being violated. Our baby, as we all know, is not really an authority, as required of the actor of the phrase *throw the book*. Therefore, Sentence (8) sounds ironic. A presuppositional condition is violated also in sentence (9). The entire presupposition states: (a) a planning failure by the actor, (b) a threatening act by a social authority, and (c) an explanation act taken to block that punishment. Now, getting A's is not a planning failure, rather it is a fortuitous success, which makes the situation humorous. Consider the next pair of sentences:

- (10) I made an appointment with my advisor.
I **met** him on time.
- (11) I made an appointment with my advisor.
I **ran into** him on time.

Both *run into* and *meet* make the same statement: two characters got into a physical proximity. However, since *run into* presupposes an unplanned, surprising element which does not exist in the situation, sentence (11) sounds incorrect.

In contrast to previous research in which presupposition was used for deriving secondary inferences which are mostly redundant, we suggest using presupposition for disambiguation, detection of irony [Dyer86a], and even for generation of irony by a computer (by applying phrases in situations where a presuppositional condition has been slightly mutated).

Was It Worth it?

The learning model described here cannot fully account for either one of the original problems, namely:

- How do humans acquire language?
- How can a computer program learn human language?

However, four general conclusions can be drawn from our research experience itself:

- (1) Learning must be an integral function of any linguistic system. **Static grammars** cannot account for important ubiquitous linguistic phenomena.
- (2) Learning must be an integral part of language **research**, since it sheds new light on important outstanding issues. Widening the research scope in that dimension turns out to be profitable.
- (3) Language acquisition requires a systematic **semantic representation**. Even learning of syntax rely on the context as it is brought to bear by world knowledge.
- (4) Language acquisition cannot be studied as an isolated learning domain. Learning of linguistic concepts is just a case, although a very special one, of **general learning in a task domain**.

In this project, we have conducted a top-down, issue-driven research. Rather than coming up with conclusive *answers*, we have pursued issues, and sub issues, which, when eventually are resolved, may explain observed behavior.

References

- [Anderson77] Anderson, J. R., "Induction of Augmented Transition Networks," *Cognitive Science* 1, pp.125-157 (1977).
- [Anderson84] Anderson, John R., *The Architecture of the Mind*, Harvard University Press, Cambridge, Mass (1984).
- [Arens82] Arens, Y., "The Context Model: Language Understanding in a Context," in *Proceedings Fourth Annual Conference of the Cognitive Science Society*, Ann Arbor, Michigan (1982).
- [Bates82] Bates, E. and B. MacWhinney, "Functionalist Approaches to Grammar," in *Language Acquisition: The State of The Art*, ed. E. Wanner L. R. Gleitman, Cambridge University Press, Cambridge (1982).
- [Becker75] Becker, Joseph D., "The Phrasal Lexicon," pp. 70-73 in *Proceedings Interdisciplinary Workshop on Theoretical Issues in Natural Language Processing*, Cambridge, Massachusetts (June 1975).
- [Berwick85] Berwick, R. C., *The Acquisition of Syntactic Knowledge*, The MIT Press, Cambridge MA (1985).
- [Bresnan82a] Bresnan, J. and R. Kaplan, "Lexical-Functional Grammar," in *The Mental Representation of Grammatical Relations*, ed. J. Bresnan, MIT Press, MA (1982).
- [Bresnan82b] Bresnan, J., "Control and Complementation," in *The Mental Representation of Grammatical Relations*, ed. J. Bresnan, The MIT Press, Cambridge MA (1982).

- [Carbonell79] Carbonell, J. G., "Subjective Understanding: Computer Models of Belief Systems," TR-150, Yale, New Haven CT (1979). Ph.D. Dissertation.
- [Carbonell84] Carbonell, J. G. and P. J. Hayes, "Coping with Extragrammaticality," pp. 437-443 in *Proceedings Coling84*, Stanford California (1984).
- [Chafe68] Chafe, W. L., "Idiomaticity as an Anomaly in the Chomskyan Paradigm," *Foundations of Language* 4 (1968).
- [Charniak80] Charniak, E., C. Riesbeck, and D. McDermott, *Artificial Intelligence Programming*, Lawrence Erlbaum Associates, Hillsdale, New Jersey (1980).
- [Charniak83] Charniak, E., "Passing Markers: A Theory of Contextual Influence in Language Comprehension," *Cognitive Science* 7(3) (1983).
- [Chomsky69] Chomsky, C., *Acquisition of Syntax in Children from 5 to 10*, MIT Press, Cambridge MA (1969).
- [Chomsky68] Chomsky, N. and M. Halle, *The Sound Pattern of English*, Harper and Row, New York, NY (1968).
- [Cottrell85] Cottrell, G. W., "Connectionistic Parsing," in *Proceedings The 7th Annual Conference of the Cognitive Science Society*, Irvine, CA (1985).
- [Cullingford78] Cullingford, R. E., "Script Application: Computer Understanding of Newspaper Stories," 116, Yale University, Department of Computer Science, New Haven, Connecticut (1978).
- [DeJong86] DeJong, G. and R. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning* 1(2) (1986).

- [Dong71] Dong, Quang Phuc, "The Applicability of Transformations to Idioms," in *Proceedings Chicago Linguistic Society* (1971). (written by J. McCawley).
- [Dyer86a] Dyer, M., M. Flowers, and J. Reeves, "A Computer Model of Irony Recognition in Narrative Understanding," *Advances in Computing and the Humanities* 1(1) (1986).
- [Dyer83] Dyer, M. G., *In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension*, MIT Press, Cambridge, MA (1983).
- [Dyer86b] Dyer, M. G. and U. Zernik, "Encoding and Acquiring Figurative Phrases in the Phrasal Lexicon," in *Proceedings 24th Annual Meeting of the Association for Computational Linguistics*, New York NY (1986).
- [Fahlman79] Fahlman, S. E., *NETL: A System for Representing and Using Real-World Knowledge*, MIT Press, Cambridge, MA (1979).
- [Fauconnier85] Fauconnier, Gilles, *Mental Spaces: Aspects of Meaning Construction in Natural Language*, MIT Press, Cambridge MA (1985).
- [Fillmore87] Fillmore, C., P. Kay, and M. O'Connor, *Regularity and Idiomaticity in Grammatical Constructions: The Case of Let Alone*, UC Berkeley, Department of Linguistics (1987). Unpublished Manuscript.
- [Fillmore78] Fillmore, C. J., "On the Organization of Semantics Information in the Lexicon," in *Proceedings CLS* (1978).
- [Fraser70] Fraser, Bruce, "Idioms within a Transformational Grammar," *Foundations of Language* 6 (1970).
- [Gasser85] Gasser, M., "Second Language Production: Coping with Gaps in Linguistics Knowledge," UCLA-AI-15, LA CA (July 1985).

- [Gasser86a] Gasser, M. and M. G. Dyer, "Speak of the Devil: Representing Deictic and Speech Act Knowledge in an Integrated Lexical Memory," in *Proceedings 8th Conference of the Cognitive Science Society*, Amherst MA (August 1986).
- [Gasser86b] Gasser, M., "Memory Organization in the Bilingual/Second Language Learner: A Computational Approach," in *Proceedings Eastern States Conference on Linguistics (ESCOL)*, Chicago IL (1986).
- [Gazdar79] Gazdar, Gerold, "A Solution to the Projection Problem," pp. 57-87 in *Syntax and Semantics (Volume 11: Presupposition)*, ed. Choon-Kyu Oh David A. Dinneen, Academic Press, New-York (1979).
- [Gazdar85] Gazdar, G., E. Klein, G. Pullum, and I. Sag, *Generalized Phrase Structure Grammar*, Harvard University Press, Cambridge MA (1985).
- [Gentner83] Gentner, Dedre, "Structure-Mapping: A Theoretical Framework for Analogy," *Cognitive Science* 7(2), pp.155-170 (1983).
- [Granger77] Granger, R. H., "FOUL-UP: A Program That Figures Out Meanings of Words from Context," pp. 172-178 in *Proceedings Fifth IJCAI*, Cambridge, Massachusetts (August 1977).
- [Granger83] Granger, R. H., "Expectation-Based Detection and Correction of Errors during Understanding of Syntactically and Semantically Ill-Formed Text," *American Journal of Computational Linguistics* 9(3) (1983).
- [Grice75] Grice, H. P., "Logic and Conversation," in *Syntax and Semantics (volume 3: Speech Acts)*, ed. P. Cole J. Morgan, Academic Press, NY (1975).
- [Hatch83] Hatch, E. M., *Psycholinguistics: A Second Language Perspective*, NewBury House, Rowley MA (1983).

- [Hendrix77] Hendrix, G., E. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a Natural Language Interface to Complex Data," in *Proceedings The Third International Conference on Very Large Data Bases*, Tokyo, Japan (1977).
- [HersHKovits85] HersHKovits, Annette, "Semantics and Pragmatics of Locative Expressions," *Cognitive Science* 9(3) (1985).
- [Hirst86] Hirst, G. J., *Semantic Interpretation and the Resolution of Ambiguity*, Cambridge, New York, NY (1986).
- [Jacobs85a] Jacobs, P. S., "A Knowledge-Based Approach to Language Production," UCB/CSD 86/254, UC Berkeley, Computer Science Division, Berkeley CA (August 1985). Ph.D. Dissertation.
- [Jacobs85b] Jacobs, Paul S., "PHRED: A Generator for Natural Language Interfaces," UCB/CSD 85/198, Computer Science Division, University of California Berkeley, Berkeley, California (January 1985).
- [Karttunen79] Karttunen, L. and S. Peters, "Conventional Implicature," in *Syntax and Semantics (Volume 11, Presupposition)*, ed. C. K. Oh D. Dinneen, Academic Press, NY (1979).
- [Katz63] Katz, J. and P. M. Postal, "The Structure of a Semantic Theory," 70, MIT, Research Lab of Electronics, Cambridge MA (1963).
- [Kay79] Kay, Martin, "Functional Grammar," pp. 142-158 in *Proceedings 5th Annual Meeting of the Berkeley Linguistic Society*, Berkeley, California (1979).
- [Keenan71] Keenan, Edward L., "Two Kinds of Presupposition in Natural Language," pp. 44-52 in *Studies in Linguistic Semantics*, ed. Charles Fillmore D. T. Langendoen, Holt, Reinhart and Winston, New York (1971).

- [Kiparsky71] Kiparsky, P. and C. Kiparsky, "Fact," in *Semantics, an Interdisciplinary Reader*, ed. D. Steinberg L. Jakobovits, Cambridge University Press, Cambridge, England (1971).
- [Kolodner84] Kolodner, J. L., *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*, Lawrence Erlbaum Associates, Hillsdale NJ (1984).
- [Lakoff80] Lakoff, George and Mark Johnson, *Metaphors We Live by*, The University of Chicago Press, Chicago and London (1980).
- [Langacker86] Langacker, R. W., "An Introduction to Cognitive Grammar," *Cognitive Science* 10(1) (1986).
- [Langley82] Langley, Pat, "Language Acquisition Through Error Recovery," *Cognition and Brain Theory* 5(3), pp.211-255 (1982).
- [Lebowitz86] Lebowitz, M., "Integrated Learning: Controlling Explanation," *Cognitive Science* 10(2) (1986).
- [Lytinen84] Lytinen, S. L., "The Organization of Knowledge in a Multilingual, Integrated Parser," YALEU/CSD/RR #340, Yale Department of Computer Science, New Haven, CT (1984).
- [MacWhinney87] MacWhinney, B., "The Competition Model," in *Mechanisms of Language Acquisition*, ed. B. MacWhinney, Lawrence Erlbaum, Hillsdale, NJ (1987). (in press).
- [Marcus80] Marcus, M., *A Theory of Syntactic Recognition for Natural Language*, The MIT Press, Cambridge MA (1980).
- [Martin86] Martin, J. H., "Views from a Kill," in *Proceedings The Eighth Annual Meeting of the Cognitive Science Society*, Amherst, MA (1986).
- [McClelland86] McClelland, J. L. and D. E. Rumelhart, *Parallel Distributed Processing*, MIT Press, Cambridge MA (1986).

- [Mitchell86] Mitchell, T., R. Keller, and S. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning* 1(1) (1986).
- [Mitchell82] Mitchell, T. M., "Generalization as Search," *Artificial Intelligence* 18, pp.203-226 (1982).
- [Mueller84] Mueller, E. and U. Zernik, "GATE Reference Manual," UCLA-AI-84-5, Computer Science, AI Lab (1984).
- [Mueller85] Mueller, E. and M. Dyer, "Daydreaming in Humans and Computers," in *Proceedings 9th International Joint Conference on Artificial Intelligence*, Los Angeles CA (1985).
- [Mueller87] Mueller, Erik T., "GATE Reference Manual (Second Edition)," UCLA-AI-87-6, UCLA, Computer Science Department, Los Angeles, CA (1987).
- [Newell57] Newell, A., J. C. Shaw, and A. Simon, "Preliminary Description of General Problem Solving Program-I (GPS-I)," CIP Working Paper 7, Carnegie Institute of Technology, Pittsburgh PA (1957).
- [Pazzani86] Pazzani, M., M. Dyer, and M. Flowers, "The Role of Prior Causal Theories in Generalization," in *Proceedings 5th National Conference on Artificial Intelligence*, Philadelphia, PA (1986).
- [Pereira80] Pereira, F. C. N. and David H. D. Warren, "Definite Clause Grammars for Language Analysis- A Survey of the Formalism and a Comparison with Augmented Transition Networks," *Artificial Intelligence* 13, pp.231-278 (1980).
- [Pinker84] Pinker, S., *Language Learnability and Language Development*, Harvard University Press, Cambridge MA (1984).
- [Reeker76] Reeker, L. H., "The Computational Study of Language Learning," in *Advances in Computers*, ed. M. Yovits M. Rubinoff, Academic Press, New York (1976).

- [Rees84] Rees, Jonathan, Norman Adams, and James Meehan, "The T Manual," , Computer Science Department, Yale University, New Haven CT (1984).
- [Richards74] Richards, Jack C., *Error Analysis*, Longman, Norfolk, Britain (1974).
- [Rosch78] Rosch, E., "Principles of Categorization," in *Cognition and Categorization*, ed. B. Lloyd, Lawrence Erlbaum Associates (1978).
- [Schank77] Schank, R. and R. Abelson, *Scripts Plans Goals and Understanding*, Lawrence Erlbaum Associates, Hillsdale, New Jersey (1977).
- [Schank78] Schank, R. and J. Carbonell, "The Gettysburg Address: Representing Social and Political Acts," TR-127, Yale University, Department of Computer Science, New Haven CT (1978).
- [Schank82] Schank, R. C., *Dynamic Memory*, Cambridge University Press, Cambridge Britain (1982).
- [Selfridge82] Selfridge, Malory, "Why Do Children Misunderstand Reversible Passives? The CHILD Program Learns to Understand Passive Sentences," pp. 251-257 in *Proceedings AAAI-82*, Pittsburgh, Pennsylvania (August 1982).
- [Selfridge80] Selfridge, Mallory G. R., "A Process Model of Language Acquisition," 172, Yale University Department of Computer Science, New Haven, Connecticut (May 1980). Ph.D. Dissertation.
- [Shortliffe76] Shortliffe, E. H., *Computer Based Medical Consultation: MYCIN*, American Elsevier (1976).
- [Small82] Small, S. and C. Rieger, "Parsing and Comprehending with Word Experts (A Theory and Its Realization)," in *Strategies for Natural Language Processing*, ed. M. Ringle, Lawrence Erlbaum, Hillsdale, NJ (1982).

- [Thompson87] Thompson, Sandra, "The Passive in English: A Discourse Perspective," in *In Honor of Ilse Lehist, Ilse Lehist Fuhendusteos*, ed. Robert and Linda Shokey, Foris (1987). to appear.
- [Turing50] Turing, A. M., "Computing Machinery and Intelligence," *Mind* 54(236) (1950).
- [Ulm75] Ulm, Susan C., "The Separation Phenomenon in English Phrasal Verbs, Double Trouble," 601, University of California Los Angeles (1975). M.A. Thesis.
- [Waltz85] Waltz, D. L. and J. B. Pollack, "Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation," *Cognitive Science* 9(1) (1985).
- [Wilensky80] Wilensky, R. and Y. Arens, "PHRAN: A Knowledge-Based Approach to Natural Language Analysis," in *Proceedings 18th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA (1980).
- [Wilensky81] Wilensky, R., "A Knowledge-Based Approach to Natural Language Processing: A progress Report," in *Proceedings Seventh International Joint Conference on Artificial Intelligence*, Vancouver, Canada (1981).
- [Wilensky82] Wilensky, R., "Points: A Theory of Structure of Stories in Memory," pp. 345-375 in *Strategies for Natural Language Processing*, ed. W. G. Lehnert M. H. Ringle, Laurence Erlbaum Associates, New Jersey (1982).
- [Wilensky83] Wilensky, R., *Planning and Understanding*, Addison-Wesley, Massachusetts (1983).
- [Wilensky84] Wilensky, R., Y. Arens, and D. Chin, "Talking to UNIX in English: an Overview of UC," *Communications of the ACM* 27(6), pp.574-593 (June 1984).

- [Wilks75] Wilks, Y., "Preference Semantics," in *The Formal Semantics of Natural Language*, ed. E. Keenan, Cambridge, Cambridge Britain (1975).
- [Winston72] Winston, P. H., "Learning Structural Descriptions from Examples," in *The Psychology of Computer Vision*, ed. P. H. Winston, McGraw-Hill, New York, NY (1972).
- [Woods70] Woods, W. A., "Transition Network Grammars for Language Analysis," *Communications of the ACM* 13 (1970).
- [Zernik85a] Zernik, U. and M. G. Dyer, "Towards a Self-Extending Phrasal Lexicon," in *Proceedings 23rd Annual Meeting of the Association for Computational Linguistics*, Chicago IL (July 1985).
- [Zernik85b] Zernik, U. and M. G. Dyer, "Failure-Driven Acquisition of Figurative Phrases by Second Language Speakers," in *Proceedings of the 7th Annual Conference of the Cognitive Science Society*, Irvine CA (August 1985).
- [Zernik85c] Zernik, U., "A Computer Model of Second Language Acquisition," in *Proceedings The 1985 Second Language Acquisition Forum*, Los Angeles, CA (1985).
- [Zernik86a] Zernik, U. and M. G. Dyer, "Language Acquisition: Learning Phrases in Context," in *Machine Learning: A Guide to Current Research*, ed. T. Mitchell J. Carbonell R. Michalsky, Kluwer, Boston MA (1986).
- [Zernik86b] Zernik, U. and M. G. Dyer, "Disambiguation and Acquisition through the Phrasal Lexicon," in *Proceedings 11th International Conference on Computational Linguistics*, Bonn Germany (1986).
- [Zernik87a] Zernik, U., "Acquiring Idioms from Examples in Context: Learning by Explanation," in *Proceedings 13th Annual Meeting of the Berkeley Linguistic Society*, Berkeley, California (February 1987).

[Zernik87b]

Zernik, U. and M. G. Dyer, "The Self-Extending Phrasal Lexicon," *The Journal of Computational Linguistics: Special Issue on the Lexicon* (1987). to appear.

Appendix A:

Program Trace

In this appendix we present three sessions with the program RINA. In the first session, RINA processes text when all the elements are given. In the second session RINA acquires a new phrase.

A.1 PARSING WITH A COMPLETE LEXICON

The input paragraph in the first session is given below:

```
Jenny Wanted to buy a car.  
She took it up with her dad
```

In parsing this text, the program's lexicon includes all the required phrases. Here is the interaction of the user with the program.

```
>>> (show *context*)  
:~~~~~  
The context is initially empty. (see footnote).  
:~~~~~  
  
>>> (parse '(Jenny wanted to buy a car))
```

```
**** reading word: JENNY
```

The following text was produced by the program, however, the text enclosed by two lines of semicolons is comments added by the author.

Single words are packaged into case frames, as the one below.

**** instantiating case frame:

^WEB.1261:

((CONCEPT ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE)))
(ROOT JENNY)
(MARKER NONE)
(DETERMINER NONE))

**** reading word: WANTED

**** triggering phrase: (X WANT TO Y)

**** instantiating case frame:

^WEB.1262:

((TENSE PAST) (ROOT WANT) (MODIFIER NONE) (POLARITY POS))

Infinitive is processed by unification of two phrases.

**** reading word: TO

**** reading word: BUY

**** triggering phrase: (X BUY Y)

**** instantiating case frame:

^WEB.1282:

((ROOT BUY) (MODIFIER NONE) (TENSE PRESENT) (POLARITY POS))

**** phrase: (X WANT TO Y) is matched in the sentence

**** instantiating phrase meaning:

^WEB.1278:

```
((ACTOR ((CLASS PERSON)
        (NAME JENNY)
        (FIRST-NAME JENNIFER)
        (AGE N-)
        (GENDER FEMALE)))
(GOAL ((HEAD ATRANS)
      (CLASS ACT)
      (ACTOR (*VAR* 'X))
      (TO (*VAR* 'X))
      (OBJ (*VAR* 'Y))))
(STATUS ACTIVE)
(CLASS GOAL))
```

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
A partial concept is instantiated for the complex clause.
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

**** reading word: A

**** reading word: CAR

**** instantiating case frame:
^WEB.1268:
((CONCEPT ((CLASS PHYS-OBJECT) (HEAD MOTOR-VEHICLE)))
(ROOT CAR)
(DETERMINER A)
(MARKER NONE))

**** phrase: (X BUY Y) is matched in the sentence

**** instantiating phrase meaning:
^WEB.1290:
((ACTOR ((CLASS PERSON)))
(TO ((CLASS PERSON)))
(OBJ ((CLASS PHYS-OBJECT) (HEAD MOTOR-VEHICLE)))
(CLASS ACT)
(HEAD ATRANS))

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
The embedded phrase is instantiated and attached to the
rest of the concept.
```

//

**** generating text:
JENNIFER HAS AN ACTIVE GOAL
TO POSSESS A MOTOR-VEHICLE.

//
Next, we show the context after parsing the first sentence.

//
>>> (show *context*)

^WEB.1278:
((ACTOR ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE)))
(GOAL ((HEAD ATRANS)
 (CLASS ACT)
 (OBJ ((CLASS PHYS-OBJECT) (HEAD MOTOR-VEHICLE)))
 (TO ((CLASS PERSON)))
 (ACTOR ((CLASS PERSON))))))
(STATUS ACTIVE)
(CLASS GOAL))

//
The object above is the main concept of the sentence. It
represents Jenny's goal to buy (atrans) a car.
//

^WEB.1296:
((HEAD MOTOR-VEHICLE) (CLASS PHYS-OBJECT))

//
The object above represents the car itself.
//

^WEB.458:
((GENDER FEMALE)
 (AGE N-)

(FIRST-NAME JENNIFER)
(NAME JENNY)
(CLASS PERSON))

////////////////////////////////////
The object above represents Jenny. Notice that the order
of objects in the context is by recency.

////////////////////////////////////
>>> (parse '(she took it up with her dad))

**** reading word: SHE

**** instantiating case frame:

^WEB.2658:
((CONCEPT ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE)))
(ROOT SHE)
(MARKER NONE)
(DETERMINER NONE))

////////////////////////////////////
Here the program read the reference "she", and by recency,
realized that the intended referent was Jenny.

////////////////////////////////////

**** reading word: TOOK

**** triggering phrase: (X TAKE Y)

case frame:
^WEB.2659:
((TENSE PAST)
(ROOT TAKE)
(MODIFIER NONE)
(POLARITY POS))

**** reading word: IT

**** instantiating case frame:

^WEB.2664:

```
((CONCEPT
  ((CLASS PHYS-OBJECT)
   (HEAD MOTOR-VEHICLE))

  ((CLASS GOAL)
   (STATUS ACTIVE)
   (GOAL ((HEAD ATRANS)
          (CLASS ACT)
          (OBJ ^WEB.2656)
          (TO ^WEB.1113)
          (ACTOR ^WEB.1113)))
   (ACTOR ((CLASS PERSON)
            (NAME JENNY)
            (FIRST-NAME JENNIFER)
            (AGE N-)
            (GENDER FEMALE))))))
```

```
(ROOT IT)
(MARKER NONE)
(DETERMINER NONE)
```

**** more than one concept found for reference: IT

```
////////////////////////////////////
Here the program read the reference "it"; there
is more than one possible referent: the car and the goal.
////////////////////////////////////
```

**** phrase: (X TAKE Y) is matched in the sentence

**** instantiating phrase meaning:

^WEB.2670:

```
((ACTOR ((CLASS PERSON)
         (NAME JENNY)
         (FIRST-NAME JENNIFER)
         (AGE N-)
         (GENDER FEMALE)))
 (OBJ ((CLASS PHYS-OBJECT)
       (HEAD MOTOR-VEHICLE)))
```

```
(TO ((CLASS PERSON)
      (NAME JENNY)
      (FIRST-NAME JENNIFER)
      (AGE N-)
      (GENDER FEMALE)))
(CLASS ACT)
(HEAD PTRANS))
```

```
;;;;;;;;;;;;;
First, the program assumes the following meaning: she took the
car.
;;;;;;;;;;;;;
```

```
**** reading word: UP
```

```
**** triggering phrase: (X V:PTRANS Y:PHYS-OBJECT UP)
```

```
**** phrase merge:
      (X TAKE Y)
      (X V:PTRANS Y:PHYS-OBJECT UP)
```

```
^WEB.2670:
((HEAD PTRANS))
(ACTOR ((CLASS PERSON)
         (NAME JENNY)
         (FIRST-NAME JENNIFER)
         (AGE N-)
         (GENDER FEMALE)))
(OBJ ((CLASS PHYS-OBJECT)
      (HEAD MOTOR-VEHICLE)))
(TO ((CLASS PERSON)
     (NAME JENNY)
     (FIRST-NAME JENNIFER)
     (AGE N-)
     (GENDER FEMALE)))
(CLASS ACT)
(DIRECTON VERTICAL-POSITIVE))
```

```
;;;;;;;;;;;;;
The particle "up" is used to modify the meaning, as shown
above: she moved the car upwards.
```

////////////////////////////////////

**** reading word: WITH

**** triggering phrase: (X TAKE Y UP WITH Z)

////////////////////////////////////

At this point a second phrase is considered.

////////////////////////////////////

**** triggering phrase: (X V:ACT WITH Y)

**** reading word: HER

**** reading word: DAD

**** instantiating case frame:

^WEB.2677:

((CONCEPT ((TITLE FATHER)
 (CLASS PERSON)
 (GENDER MALE)))
(ROOT DAD)
(MARKER WITH)
(DETERMINER ((HEAD POSSESSIVE)
 (ROOT SHE)
 (CONCEPT ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE))))))

////////////////////////////////////

"Her dad" is a complex clause which has two referents:
Jenny as the possessor, and the father who is the possessed.

////////////////////////////////////

**** phrase merge:

(X TAKE Y)
(X V:PTRANS Y:PHYS-OBJECT UP)
(X V:ACT WITH Y:PERSON)


```

^WEB.2670:
((HEAD PTRANS)
  (ACTOR ((CLASS PERSON)
    (NAME JENNY)
    (FIRST-NAME JENNIFER)
    (AGE N-)
    (GENDER FEMALE)))
  (OBJ ((CLASS PHYS-OBJECT)
    (HEAD MOTOR-VEHICLE)))
  (TO ((CLASS PERSON)
    (NAME JENNY)
    (FIRST-NAME JENNIFER)
    (AGE N-)
    (GENDER FEMALE)))
  (CLASS ACT)
  (DIRECTON VERTICAL-POSITIVE)
  (ASSISTED-BY ((TITLE FATHER)
    (CLASS PERSON)
    (GENDER MALE))))

```

```

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
The particle "with" is used to modify the meaning, as shown
above: she moved the car upwards with her dad' assistance.
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

```

```

**** phrase: (X TAKE Y UP WITH Z)
is matched in the sentence

```

```

**** proving (trying to prove) presupposition of phrase:

```

```

^WEB.2714:
((HIGH ((TITLE FATHER)
  (CLASS PERSON)
  (GENDER MALE)))
  (LOW ((CLASS PERSON)
    (NAME JENNY)
    (FIRST-NAME JENNIFER)
    (AGE N-)
    (GENDER FEMALE)))
  (CLASS RELATION)
  (HEAD AUTHORITY))

```

::
In order for this interpretation to be validated, its
presupposition must be proved.
::

PROVE #(^WEB.2714)
RULE #(^WEB.2297)
UNIFIES WITH #(^WEB.2714)
PROVE ALL (#(^WEB.2731))
PROVE #(^WEB.2731)
FACT #(^WEB.2727)
UNIFIES WITH #(^WEB.2731)
PROVED #(^WEB.2731) 1 WAY(S)
PROVED #(^WEB.2714) 1 WAY(S)

**** instantiating phrase meaning:

^WEB.2715:

((FROM ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE))))
(OBJ ((CLASS GOAL)
 (STATUS ACTIVE)
 (OBJ ((HEAD ATRANS)
 (CLASS ACT)
 (OBJ ^WEB.2656)
 (TO ^WEB.1113)
 (ACTOR ^WEB.1113))))
 (ACTOR ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE))))))
(TO ((TITLE FATHER)
 (CLASS PERSON)
 (GENDER MALE)))
(CLASS EVENT)
(HEAD AUTH-APPEAL))

**** proving (trying to prove) a discrepancy:


```

                (NAME JENNY)
                (FIRST-NAME JENNIFER)
                (AGE N-)
                (GENDER FEMALE))))
    (TO ((TITLE FATHER)
        (CLASS PERSON)
        (GENDER MALE)))
    (CLASS EVENT)
    (HEAD AUTH-APPEAL))

```

```

^WEB.2726:
((GENDER MALE) (CLASS PERSON) (TITLE FATHER))

```

```

^WEB.2727:
((HEAD FAMILY)
 (CLASS SOC-STRUCTURE)
 (PARENT ((TITLE FATHER)
          (CLASS PERSON)
          (GENDER MALE)))
 (ACTOR ((GENDER FEMALE)
         (AGE N-)
         (FIRST-NAME JENNIFER)
         (NAME JENNY)
         (CLASS PERSON))))

```

```

^WEB.1113:
((GENDER FEMALE)
 (AGE N-)
 (FIRST-NAME JENNIFER)
 (NAME JENNY)
 (CLASS PERSON))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
The context includes the concepts brought in by parsing,
ordered by recency.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

In conclusion, notice two points: (a) how the reference "it" was resolved by selectional restrictions; (b) how the appropriate phrase was selected and proved; (c) how the context takes place in the parsing process.

A.2 LEARNING A NEW PHRASE

Next we show how the program handles the same input when a lexical entry (X take Y up with Z) is not included in the lexicon. The input text is the same as in A.1:

Jenny wanted to buy a car
She took it up with her dad.

>>> (show *context*)

```
.....  
The context is initially empty.  
.....
```

>>> (parse '(Jenny wanted to buy a car))

```
.....  
The trace for this sentence is identical to the trace in A.1,  
and it is not shown here. However, the trace for the next  
sentence is different, since the phrase is unknown.  
.....
```

**** generating text:

JENNIFER HAS AN ACTIVE GOAL
TO POSSESS A MOTOR-VEHICLE.

>>> (parse '(she took it up with her dad))

**** reading word: SHE

**** reading word: TOOK

**** reading word: IT

**** reading word: UP

```
.....  
So far, the trace is identical to the previous trace and it is  
not shown here. The continuation is different.  
.....
```

**** triggering phrase: (X V:PTRANS Y:PHYS-OBJECT UP)

**** phrase merge:

(X TAKE Y)

(X V:PTRANS Y:PHYS-OBJECT UP)

^WEB.3530:

((HEAD PTRANS))
 (ACTOR ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE)))
 (OBJ ((CLASS PHYS-OBJECT)
 (HEAD MOTOR-VEHICLE)))
 (TO ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE)))
 (CLASS ACT)
 (DIRECTON VERTICAL-POSITIVE))

////////////////////////////////////
The particle "up" is used to modify the meaning, as shown
above: she moved the car upwards.
////////////////////////////////////

**** reading word: WITH

**** triggering phrase: (X TAKE Y UP WITH Z)

**** triggering phrase: (X V:ACT WITH Y)

**** reading word: HER

**** reading word: DAD

**** instantiating case frame:

^WEB.3621:

((CONCEPT ((TITLE FATHER)
 (CLASS PERSON)
 (GENDER MALE)))
(ROOT DAD)
(MARKER WITH)

```
(DETERMINER ((HEAD POSSESSIVE)
              (ROOT SHE)
              (CONCEPT ((CLASS PERSON)
                          (NAME JENNY)
                          (FIRST-NAME JENNIFER)
                          (AGE N-)
                          (GENDER FEMALE))))))
```

```
////////////////////////////////////
"Her dad" is a complex clause which has two referents:
Jenny as the possessor, and the father who is the possessed.
////////////////////////////////////
```

```
**** phrase merge:
      (X TAKE Y)
      (X V:PTRANS Y:PHYS-OBJECT UP)
      (X V:ACT WITH Y:PERSON)
```

```
^WEB.3530:
((HEAD PTRANS)
 (ACTOR ((CLASS PERSON)
         (NAME JENNY)
         (FIRST-NAME JENNIFER)
         (AGE N-)
         (GENDER FEMALE)))
 (OBJ ((CLASS PHYS-OBJECT)
       (HEAD MOTOR-VEHICLE)))
 (TO ((CLASS PERSON)
      (NAME JENNY)
      (FIRST-NAME JENNIFER)
      (AGE N-)
      (GENDER FEMALE)))
 (CLASS ACT)
 (DIRECTON VERTICAL-POSITIVE)
 (ASSISTED-BY ((TITLE FATHER)
               (CLASS PERSON)
               (GENDER MALE))))
```

```
////////////////////////////////////
The particle "with" is used to modify the meaning, as shown
above: she moved the car upwards with her dad' assistance.
////////////////////////////////////
```

//

**** proving (trying to prove) a discrepancy:

```

^WEB.3530:
((HEAD PTRANS)
  (ACTOR ((CLASS PERSON)
    (NAME JENNY)
    (FIRST-NAME JENNIFER)
    (AGE N-)
    (GENDER FEMALE)))
  (OBJ ((CLASS PHYS-OBJECT)
    (HEAD MOTOR-VEHICLE)))
  (TO ((CLASS PERSON)
    (NAME JENNY)
    (FIRST-NAME JENNIFER)
    (AGE N-)
    (GENDER FEMALE)))
  (CLASS ACT)
  (DIRECTON VERTICAL-POSITIVE)
  (ASSISTED-BY ((TITLE FATHER)
    (CLASS PERSON)
    (GENDER MALE))))

```

//

In order for this interpretation to be disproved, its concept must be proved inconsistent with the context.

//

```

PROVE #(^WEB.3530)
////////////////////////////////////////////////////////////////
By chaining two rules, the concept is proved inconsistent with
the context (the actual trace fills several pages, and it shows
a lot of backtracking):
person drive a car --> person possess the car
person want to buy a car --> person do not possess the car
////////////////////////////////////////////////////////////////
PROVED #(^WEB.3530) 1 WAY(S)

```

**** generating text:
 JENNIFER DROVE A MOTOR-VEHICLE UPWARDS WITH HER FATHER?

::
This hypothetical meaning is known to be incorrect,
yet the user must know RINA's state of knowledge.
::

current-hypothesis

^WEB.3706:
((HEAD PHRASE)
 (PATTERN ((SUBJECT ((CONCEPT ^WEB.1113)
 (MARKER NONE)))
 (VERB ((ROOT TAKE)
 (MODIFIER UP)))
 (OBJECT1 ((CONCEPT ^WEB.1296)
 (MARKER NONE)))
 (OBJECT2 ((CONCEPT ^WEB.2726)
 (MARKER WITH))))))
(CONCEPT ((HEAD PTRANS)
 (ACTOR ^WEB.1113)
 (OBJ ^WEB.1296)
 (TO ^WEB.1113)
 (CLASS ACT)
 (DIRECTON VERTICAL-POSITIVE)
 (ASSISTED-BY ^WEB.2726))))

::
The user reviews the hypothesis. This hypothesis is not
yet generalized into a template since the program has not
yet decided a new phrase is involved.
::

>>> (parse '(no.))
::
The user provides a correction.
::

**** reading word: NO

**** ready to accept another example.

>>> (parse '(she took up the problem with her dad.))
**** reading word: SHE

**** instantiating case frame:
^WEB.3713:
((CONCEPT ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE)))
(ROOT SHE)
(MARKER NONE)
(DETERMINER NONE))

**** reading word: TOOK

**** triggering phrase: (X TAKE Y)

case frame:
^WEB.3716:
((TENSE PAST)
(ROOT TAKE)
(MODIFIER NONE)
(POLARITY POS))

**** reading word: UP

::
So far the input matches the hypothesis.
::

**** reading word: THE

**** reading word: PROBLEM

**** instantiating case frame:
^WEB.3719:
((CONCEPT ((CLASS GOAL)
 (STATUS ACTIVE)

```
(GOAL ((HEAD ATRANS)
      (CLASS ACT)
      (OBJ ^WEB.2656)
      (TO ^WEB.1113)
      (ACTOR ^WEB.1113)))
(ACTOR ((CLASS PERSON)
      (NAME JENNY)
      (FIRST-NAME JENNIFER)
      (AGE N-)
      (GENDER FEMALE))))
```

```
(ROOT PROBLEM)
(MARKER NONE)
(DETERMINER THE)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
The reference "the problem" is matched with Jenny's problem
which is found in the context (a problem is an active goal).
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
**** reading word: UP
```

```
**** reading word: WITH
```

```
**** reading word: HER
```

```
**** reading word: DAD
```

```
**** instantiating case frame:
```

```
^WEB.3764:
```

```
((CONCEPT ((TITLE FATHER)
            (CLASS PERSON)
            (GENDER MALE)))
```

```
(ROOT DAD)
```

```
(MARKER WITH)
```

```
(DETERMINER ((HEAD POSSESSIVE)
```

```
(ROOT SHE)
```

```
(CONCEPT ((CLASS PERSON)
```

```
(NAME JENNY)
```

```
(FIRST-NAME JENNIFER)
```

```
(AGE N-)
```

```
(GENDER FEMALE))))))
```

**** compare hypothesis with input phrase

**** found discrepancy:

^WEB.3784

((HEAD DISCREPANCY)
 (EXPECTED ((CLASS PHYS-OBJECT)))
 (RECEIVED ((CLASS GOAL)))
 (CASE OBJECT1))

::
The class of "the problem" does not match the class expected
in the hypothesis above.
::

**** forming a new hypothesis

**** 1. forming the new meaning

**** the context:

^WEB.3650

((HEAD PLANNING-SCRIPT)
 (NAME PROBLEM-SOLVING)
 (PROBLEM ((HEAD GOAL)
 (STATUS ACTIVE)
 (VAR ?X)))
 (SOLUTION ((HEAD PLAN)
 (FOR-GOAL ((HEAD GOAL)
 (STATUS ACTIVE)
 (VAR ?X)))
 (STATUS UNKNOWN)
 (VAR ?Y)))
 (EVENT1 ((CLASS EVENT)
 (HEAD FIND-SOLUTION)))
 (EVENT2 ((CLASS EVENT)
 (HEAD APPLY-SOLUTION)))
 (EVENT3 ((CLASS EVENT)
 (HEAD RESOLVE-PROBLEM))))

////////////////////////////////////
Within the global context, the search is in the problem-
solving script (Jenny tries to solve her problem), due
to the reference "the problem".
////////////////////////////////////

**** select event in script: PROBLEM-SOLVING

**** the clues:

X:PERSON V:ACT WITH Y:PERSON
^WEB.1034
((CLASS RELATION)
 (HEAD ASSISTANCE)
 (ACTOR ?X)
 (ACT ?V)
 (ASSISTED-BY ?Y))

////////////////////////////////////
First linguistic clue: the word "with" means assistance.
////////////////////////////////////

X:PERSON V:ACT UP
^WEB.1046
((CLASS STATE)
 (STATUS POSITIVE-OUTCOME)
 (ACTOR ?X)
 (PLAN ?V))

////////////////////////////////////
Second linguistic clue: the word "up" can mean a positive
outcome for a plan.
////////////////////////////////////

**** selected event:

^WEB.3787
((HEAD FIND-SOLUTION)
 (CLASS EVENT)
 (ACT ((HEAD GOAL)
 (NAME D-KNOW))))

```

(ACTOR ((CLASS PERSON)
        (NAME JENNY)
        (FIRST-NAME JENNIFER)
        (AGE N-)
        (GENDER FEMALE)))
(PROBLEM ((CLASS GOAL)
          (STATUS ACTIVE)
          (GOAL ((HEAD ATRANS)
                (CLASS ACT)
                (OBJ ^WEB.2656)
                (TO ^WEB.1113)
                (ACTOR ^WEB.1113)))
          (ACTOR ((CLASS PERSON)
                  (NAME JENNY)
                  (FIRST-NAME JENNIFER)
                  (AGE N-)
                  (GENDER FEMALE))))))

```

```

/////////////////////////////////////////////////////////////////
The first event of the script is attempted first.
/////////////////////////////////////////////////////////////////

```

**** the hypothesis:

```

^WEB.3792
((HEAD ASSISTANCE)
 (CLASS RELAIION)
 (ACTOR ^WEB.2726)
 (ASSISTED-BY ^WEB.1113)
 (OBJ ((HEAD EVENT)
        (NAME FIND-SOLUTION)
        (ACT ^WEB.2754)
        (ACTOR ^WEB.1113)
        (PROBLEM ((CLASS GOAL)
                  (STATUS ACTIVE)
                  (GOAL ^WEB.2213)))
                  (ACTOR ^WEB.1113))))

```

```

/////////////////////////////////////////////////////////////////
this hypothesis indicates that the father helped Jenny find

```

a solution.

;;

**** prove the hypothesis:

PROVE #(^WEB.3792)

;;

The hypothesis is proved by the following chain:

1. Jenny's father possesses money.
2. A father agrees to give money to a daughter.
3. Giving money to Jenny can solve her problem.

;;

PROVED #(^WEB.3792) 1 WAY(S)

**** 2. constructing the new pattern:

**** including phrase:

(X:PERSON V:ACT WITH Y:PERSON)

**** excluding phrase

(X:PERSON V:ACT UP)

**** the new hypothesis:

current-hypothesis

^WEB.3706:

((HEAD PHRASE)

(PATTERN ((SUBJECT ((VAR ?X)
(MARKER NONE)))
(VERB ((ROOT TAKE)
(MODIFIER NONE)))
(OBJECT1 ((VAR ?Y)
(MARKER NONE)))
(OBJECT2 ((VAR ?Z)
(MARKER WITH))))))

(CONCEPT ((CLASS RELATION)
(HEAD ASSISTANCE)
(ACTOR ?X)
(ASSISTED-BY ?Z)
(OBJ ((HEAD EVENT)
(NAME FIND-SOLUTION))

(ACTOR ?X)
(PROBLEM ((CLASS GOAL)
(STATUS ACTIVE)
(GOAL ?Y))))))

**** generating text:
JENNIFER TOOK IT WITH HER FATHER.
HE HELPED HER FIND A SOLUTION.

////////////////////////////////////
The user provides another example to correct the pattern.
////////////////////////////////////
>>> (parse '(no))

**** reading word: NO

**** ready to accept another example.

>>> (parse '(she took up it with him.))

**** reading word: SHE

**** instantiating case frame:
^WEB.3713:
((CONCEPT ((CLASS PERSON)
(NAME JENNY)
(FIRST-NAME JENNIFER)
(AGE N-)
(GENDER FEMALE)))
(ROOT SHE)
(MARKER NONE)
(DETERMINER NONE))

**** reading word: TOOK

**** triggering phrase: (X TAKE Y)

case frame:
^WEB.3716:
((TENSE PAST)

(ROOT TAKE)
(MODIFIER NONE)
(POLARITY POS)

**** reading word: IT

**** instantiating case frame:

^WEB.3719:

((CONCEPT ((CLASS GOAL)
 (STATUS ACTIVE)
 (GOAL ((HEAD ATRANS)
 (CLASS ACT)
 (OBJ ^WEB.2656)
 (TO ^WEB.1113)
 (ACTOR ^WEB.1113)))
 (ACTOR ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE))))))

(ROOT PROBLEM)
(MARKER NONE)
(DETERMINER THE))

**** reading word: UP

**** reading word: WITH

**** reading word: HIM

**** instantiating case frame:

^WEB.3764:

((CONCEPT ((TITLE FATHER)
 (CLASS PERSON)
 (GENDER MALE)))

(ROOT HE)
(MARKER WITH))

**** compare hypothesis with input phrase

**** found discrepancy:

^WEB.3784
((HEAD DISCREPANCY)
 (EXPECTED ((MODIFIER NONE)))
 (RECEIVED ((MODIFIER UP)))
 (CASE VERB))

////////////////////////////////////
The particle "up" is not accounted for.
////////////////////////////////////

**** constructing the new pattern:

**** including phrase
 (X:PERSON V:ACT UP)

**** the new hypothesis:

current-hypothesis

^WEB.3706:
((HEAD PHRASE)
 (PATTERN ((SUBJECT ((VAR ?X)
 (MARKER NONE)))
 (VERB ((ROOT TAKE)
 (MODIFIER UP)))
 (OBJECT1 ((VAR ?Y)
 (MARKER NONE)))
 (OBJECT2 ((VAR ?Z)
 (MARKER WITH))))))
 (CONCEPT ((HEAD ASSISTANCE)
 (CLASS RELATION)
 (ASSISTED-BY ?Z)
 (ACTOR ?X)
 (INTENTION ((HEAD EVENT)
 (NAME FIND-SOLUTION)
 (ACTOR ?X)
 (PROBLEM ((CLASS GOAL)
 (STATUS ACTIVE)
 (GOAL ?Y))))))))))

**** generating text:
 JENNIFER TOOK IT UP WITH HER FATHER.

HE HELPED HER FIND A SOLUTION.

////////////////////////////////////
The user provides another example to correct the meaning of the
phrase.

////////////////////////////////////
>>> (parse ' (no)

**** reading word: NO

**** ready to accept another example.

>>> (parse ' (she discussed it with him))

**** reading word: SHE

**** instantiating case frame:

^WEB.3713:

((CONCEPT ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE)))
(ROOT SHE)
(MARKER NONE)
(DETERMINER NONE))

**** reading word: DISCUSSED

**** triggering phrase: (X DISCUSS Y WITH Z)

case frame:

^WEB.3716:

((TENSE PAST)
(ROOT DISCUSS)
(MODIFIER NONE)
(POLARITY POS))

**** reading word: IT

**** instantiating case frame:

^WEB.3719:
((CONCEPT ((CLASS GOAL)
 (STATUS ACTIVE)
 (GOAL ((HEAD ATRANS)
 (CLASS ACT)
 (OBJ ^WEB.2656)
 (TO ^WEB.1113)
 (ACTOR ^WEB.1113)))
 (ACTOR ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE))))))

(ROOT IT)
(MARKER NONE))

**** reading word: WITH

**** reading word: HIM

**** instantiating case frame:

^WEB.3764:
((CONCEPT ((TITLE FATHER)
 (CLASS PERSON)
 (GENDER MALE)))

(ROOT HE)
(MARKER WITH))

**** phrase: (X DISCUSS Y WITH Z)
is matched in the sentence

**** instantiating phrase meaning:

^WEB.2715:
((FROM ((CLASS PERSON)
 (NAME JENNY)
 (FIRST-NAME JENNIFER)
 (AGE N-)
 (GENDER FEMALE)))

(OBJ ((CLASS GOAL)
 (STATUS ACTIVE)
 (OBJ ((HEAD ATRANS)

```
(CLASS ACT)
(OBJ ^WEB.2656)
(TO ^WEB.1113)
(ACTOR ^WEB.1113)))
(ACTOR ((CLASS PERSON)
(NAME JENNY)
(FIRST-NAME JENNIFER)
(AGE N-)
(GENDER FEMALE))))))
(TO ((TITLE FATHER)
(CLASS PERSON)
(GENDER MALE)))
(CLASS ACT)
(HEAD MTRANS))
```

**** found discrepancy:

^WEB.3784

```
((HEAD DISCREPANCY)
(EXPECTED ((CLASS RELATION)
(HEAD ASSISTANCE)
(ASSISTED-BY ?Z)))
(RECEIVED ((CLASS ACT)
(HEAD MTRANS)
(TO ?Z)))
```

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
Instead of the expected assistance relation where the father
helps, the new example means an mtrans where the father is talked
to.
```

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

**** constructing the new meaning:

```
**** including phrase
(X:PERSON V:ACT UP)
```

**** the new hypothesis:

current-hypothesis

^WEB.3706:

```

((HEAD PHRASE)
 (PATTERN ((SUBJECT ((VAR ?X)
                    (MARKER NONE)))
 (VERB ((ROOT TAKE)
        (MODIFIER UP)))
 (OBJECT1 ((VAR ?Y)
           (MARKER NONE)))
 (OBJECT2 ((VAR ?Z)
           (MARKER WITH))))))
(CONCEPT ((HEAD MTRANS)
 (CLASS ACT)
 (ACTOR1 ?X)
 (TO ?Z)
 (OBJ ((HEAD EVENT)
       (NAME FIND-SOLUTION)
       (ACTOR ?X)
       (PROBLEM ((CLASS GOAL)
                (STATUS ACTIVE)
                (GOAL ?Y))))))))

```

**** generating text:

```

JENNIFER TALKED TO HER FATHER
ABOUT HER ACTIVE GOAL
TO POSSESS A MOTOR-VEHICLE.

```

>>> (show *context*)

In conclusion, we showed how the a phrase is extracted from a given episode. The pattern was extracted from the given sentences, and the context was extracted from the context.

Appendix B: MCRINA

The program MCRINA presents a basic phrasal parser, which is a stripped down version of RINA. MCRINA is intended to serve as a text analyzer for AI applications, by converting text into conceptual representation. MCRINA can be extended in several ways:

- (1) The sample lexicon provided in Section B.4 can be extended to include other phrases as needed for the application.
- (2) Conceptual representation, which is up to the application, can be constructed as the system is developed and added on to the file called rules.t.
- (3) Linguistic features can be added according to the guidelines in Chapter 11.
- (4) Acquisition functions can be implemented by adding on the learning rules in Chapter 8.

In general, Chapters 8 and 11 should be used as the background for understanding the program, in addition to the old and the new GATE manuals [Mueller84, Mueller87].

B.1 Getting Started

In order to use the parser, the following steps must be taken:

- (1) Get into the MCRINA directory on the APOLLO system (at UCLA):

```
cd /ucla/ai_research/uri/MCRINA
```

- (2) Load GATE (see GATE Manual [Mueller87])

- (3) Load MCRINA.

```
(load 'load.t)
```

- (4) Call the parsing function:

```
(parse '(david took on goliath))
```

B.2 Adding Lexical Entries

In order to add entries to the lexicon, use the format in the sample lexicon.

- (1) Place your lexicon in the file lexicon.t

- (2) Call the loading function:

```
(load-from-file *lex* 'lexicon.t)
```

- (3) Add new single words into the file world.t

- (4) Call the loading function:

```
(load-from-file wl 'world.t)
```


B.3 Sample Session

Next, we show a session with the program MCRINA, demonstrating how the program processes the following input text:*

```
Al went on trial.  
The judge threw the book at him.
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;  
This shows the initial context:  
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
>>> (mem-get-all *context*)  
(#{OB.1199: (PERSON name 'JENNY)})  
(#{OB.1197: (PERSON name 'DAVID)})  
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;  
This is the call to the parser  
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
>>> (parse '(david went on trial))
```

```
**** reading word: DAVID  
(WORD-SPEC root 'DAVID  
           number 'NIL  
           word-type 'NAME  
           lex 'DAVID  
           form 'NOUN  
           function 'NONE)
```

```
**** instantiating case frame:  
(NOUN-CASE marker 'NONE  
           determiner 'NONE  
           root 'DAVID  
           concept (PERSON name 'DAVID)  
           lex 'DAVID)
```

* Differences in the input and output formats between this trace and the trace in the previous appendix are due to the use of an upgraded version of GATE. This version has two new features: (a) I/O is conceptually clearer, and (b) it uses only minimal object-oriented features of the language T, and thus it runs much faster.

**** reading word: WENT
(WORD-SPEC root 'GO
lex 'WENT
form 'VERB
suff 2)

**** instantiating case frame:
(VERB-CASE tense 'PAST
modifier 'NONE
lex 'NONE
root 'GO)

***** triggering phrase: (HE WENT ON TRIAL)

**** reading word: ON
(WORD-SPEC root 'ON
form 'PREP)

**** reading word: TRIAL
(WORD-SPEC root 'TRIAL
number 'SINGLE
word-type 'NONE
lex 'TRIAL
form 'NOUN
function 'NONE)

**** instantiating case frame:
(NOUN-CASE marker 'ON
determiner 'NONE
root 'TRIAL
concept (SOC-SCRIPT name 'TRIAL)
lex 'TRIAL)

***** modified concept: #{OB.1222: (SOC-SCRIPT name 'TRIAL)}
()

////////////////////////////////////
This is the trial script which is applied as a result
of reading the text. David is inserted as the
defendant in that script (since he went on trial).
////////////////////////////////////

```
>>> (px ^ob.1222)
((TYPE #{SOC-SCRIPT} ())
 (NAME TRIAL ())
 (DEFENDANT #{OB.1197: (PERSON name 'DAVID)} ()))
```

```
;;;;;;;;;;;;;
Parsing the second sentence:
;;;;;;;;;;;;;
```

```
>>> (parse '(the judge threw the book at him))
```

```
**** reading word: THE
(WORD-SPEC root 'THE
           form 'DETERMINER)
```

```
**** reading word: JUDGE
(WORD-SPEC root 'JUDGE
           number 'SINGLE
           word-type 'NONE
           lex 'JUDGE
           form 'NOUN
           function 'NONE)
```

```
**** instantiating case frame:
(NOUN-CASE marker 'NONE
           determiner 'THE
           root 'JUDGE
           concept (PERSON)
           lex 'JUDGE)
```

```
**** reading word: THREW
(WORD-SPEC root 'THROW
           lex 'THREW
           form 'VERB
           suff 2)
```

```
**** instantiating case frame:
(VERB-CASE tense 'PAST
           modifier 'NONE
           lex 'NONE)
```

root 'THROW)

***** triggering phrase: (THE JUDGE THREW THE BOOK AT HIM)

***** triggering phrase: (HE THREW A PHYS-OBJ)

**** reading word: THE
(WORD-SPEC root 'THE
 form 'DETERMINER)

**** reading word: BOOK
(WORD-SPEC root 'BOOK
 number 'SINGLE
 word-type 'NONE
 lex 'BOOK
 form 'NOUN
 function 'NONE)

**** instantiating case frame:
(NOUN-CASE marker 'NONE
 determiner 'THE
 root 'BOOK
 concept 'NIL
 lex 'BOOK)

**** failed case
#{OB.1244: (NOUN-CASE marker 'NONE concept)}

Failed case frames are instrumental in error analysis for learning.

**** reading word: AT
(WORD-SPEC root 'AT
 form 'PREP)

***** triggering phrase: (HE PROPELLED A PHYS-OBJ AT THING)

**** failed case
#{OB.1254: (NOUN-CASE marker 'NONE concept)}

**** reading word: HIM
(WORD-SPEC root 'HE
 number 'NIL
 word-type 'PRONOUN
 lex 'HIM
 form 'NOUN
 function 'OBJECTIVE)

**** instantiating case frame:
(NOUN-CASE marker 'AT
 determiner 'NONE
 root 'HE
 concept (PERSON name 'DAVID)
 lex 'HIM)

**** failed case
#{OB.1255: (NOUN-CASE marker 'AT concept)}

***** proved presupposition of phrase: (THE JUDGE THREW THE
BOOK AT HIM)

***** instantiated concept
#{OB.1262: (AUTH-PUNISH from (PERSON) to ...)}
of phrase (THE JUDGE THREW THE BOOK AT HIM)

:::
This is the context after parsing
:::

>>> (mem-get-all *context*)
(#{OB.1262: (AUTH-PUNISH from (PERSON) to ...)})
(#{OB.1222: (SOC-SCRIPT name 'TRIAL)})
(#{OB.1197: (PERSON name 'DAVID)})
(#{OB.1228: (PERSON)})
(#{OB.1199: (PERSON name 'JENNY)})

:::
In particular we expand the following two concepts:
:::

>>> (px `ob.1222)
((TYPE #{SOC-SCRIPT} ()))

```
(NAME TRIAL ())
(DEFENDANT #{OB.1197: (PERSON name 'DAVID)} ())
(JUDGE #{OB.1228: (PERSON)} ()))
```

```
>>> (px ^ob.1262)
((TYPE #{AUTH-PUNISH} ())
 (FROM #{OB.1228: (PERSON)} ())
 (TO #{OB.1197: (PERSON name 'DAVID')} ()))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

B.4 A Sample Lexicon

The following entries reside in the file called lexicon.t

```
;lexicon for DAVID AND THE JUDGE-
(PHRASE
  comment '(he took it up with his dad)
  pattern (PATTERN
    subject (CASE marker 'none
              concept ?x:PERSON)
    verb (CASE root 'take
           modifier 'up)
    object1 (CASE marker 'none
              concept ?z:ACTIVE-GOAL)
    object2 (CASE marker 'with
              concept ?y:PERSON))
  pres (AUTHORITY high ?y
         low ?x)
  concept (AUTH-APPEAL from ?x
           to ?y))

(PHRASE
  comment '(he took up jogging)
  pattern (PATTERN
    subject (CASE marker 'none
              concept ?x)
    verb (CASE
           root 'take
           modifier 'up)
```

```

        object1 (CASE
                marker 'none
                concept ?z:ACTIVITY-THEME))
modify
  (MODIFIER
   concept ?z
   path '(actor)
   val ?x))

(PHRASE
 comment '(he took a ball)
 pattern (PATTERN
         subject (CASE marker 'none
                    concept ?x:PERSON)
         verb (CASE
              root 'take
              modifier 'none)
         object1 (CASE
                 marker 'none
                 concept ?z:PHYS-OBJECT)) .
 concept
   (ATRANS
    actor ?x
    to ?x
    obj ?z))

(PHRASE
 comment '(he threw a phys-obj)
 pattern (PATTERN
         subject (CASE marker 'none
                    concept ?x:PERSON)
         verb (CASE root 'throw
                  modifier 'none)
         object1 (CASE marker 'none
                   concept ?z:PHYS-OBJECT))
 concept (PROPEL
         actor ?x
         obj ?z
         instrument 'hand))

(PHRASE

```

```

comment '(he propelled a phys-obj AT thing)
pattern (PATTERN
  subject (CASE marker 'none
           concept ?x:PERSON)
  verb (CASE root ?v
        modifier 'none)
  object1 (CASE marker 'none
           concept ?z:PHYS-OBJECT)
  object2 (CASE marker 'AT
           concept ?y:PERSON))
concept (PROPEL actor ?x
        obj ?z
        direction ?y
        mode 'no-acknowledge))

```

(PHRASE

```

comment '(the judge threw the book at him)
pattern (PATTERN
  subject (CASE marker 'none
           concept ?x:PERSON)
  verb (CASE root 'throw
        modifier 'none)
  object1 (CASE marker 'none
           determiner 'the
           root 'book)
  object2 (CASE marker 'at
           concept ?z:PERSON))
pres (SOC-SCRIPT name 'trial
      judge ?x
      defendant ?z)

concept (AUTH-PUNISH from ?x
        to ?z))

```

(PHRASE

```

comment '(he went on trial)
pattern (PATTERN

```



```

subject (CASE marker 'none
          concept ?x:PERSON)
verb (CASE root 'go
      modifier 'none)
object1 (CASE marker 'on
         determiner 'none
         concept ?y:SOC-SCRIPT
         root 'trial))
modify (CONC-MODIFIER
       concept ?y
       path '(defendant)
       val ?x))

```

B.5 Single-Word Definitions

Single words are encoded separately. However, this system is not different from definitions of phrases. The entries below reside in the file called world.t.

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;The words for DAVID AND THE JUDGE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(PHRASE
 pattern 'ball
 concept (PHYS-OBJECT
         name 'ball
         shape 'round
         function 'toy))

```

```

(PHRASE
 pattern 'book
 concept (PHYS-OBJECT
         name 'book
         function 'container
         contents 'information))

```

```

(PHRASE
 pattern 'boy

```

concept (PERSON
 age 'N-
 gender 'male))

(PHRASE
pattern 'woman
concept (PERSON
 age 'N
 gender 'female))

(PHRASE
pattern 'car
concept VEHICLE
 function 'transportation)

(PHRASE
pattern 'dad
concept (PERSON
 gender 'male
 title 'father)
pres (SOC-STRUCTURE
 name 'family
 parent ?concept))

(PHRASE
pattern 'defendant
concept (PERSON)
pres (SOC-SCRIPT
 name 'trial
 defendant ?concept))

(PHRASE
pattern 'jogging
concept (ACTIVITY-THEME
 name 'running-for-fitness))

(PHRASE
pattern 'judge
concept (PERSON
 title 'judge)
pres (SOC-SCRIPT

```
name 'trial
judge ?concept))
```

```
(PHRASE
pattern 'law
concept (ABS-OBJECT
      name 'rule)
pres   (SOC-SCRIPT
      name 'trial
      code ?concept))
```

```
(PHRASE
pattern 'money
concept (SOC-POSSESSION
      function 'currency))
```

```
(PHRASE
pattern 'problem
concept (ACTIVE-GOAL)
pres   (GOAL-SITUATION))
```

```
(PHRASE
pattern 'school
concept (LOCATION
      name 'school
      function 'education))
```

```
(PHRASE
pattern 'son
concept (PERSON
      gender 'male
      title 'son)
pres   (SOC-OBJECT
      name 'family
      child ?concept))
```

```
(PHRASE
pattern 'trial
concept (SOC-SCRIPT
      name 'trial))
```

```
(PHRASE
  pattern 'trip
  concept (SOC-SCRIPT
           name 'trip))

(PHRASE
  pattern 'John
  concept (PERSON
           name 'john
           gender 'male))

(PHRASE
  pattern 'Mary
  concept (PERSON
           name 'mary
           gender 'female))

(PHRASE
  pattern 'David
  concept (PERSON
           name 'david
           gender 'male))

(PHRASE
  pattern 'Jenny
  concept (PERSON
           name 'jenny
           gender 'female))

(PHRASE
  pattern 'Goliath
  concept (PERSON
           name 'goliath
           gender 'male))

(PHRASE
  pattern 'ucla
  concept (LOCATION
           head 'ucla))
```

```
(PHRASE
  pattern 'he
  concept (PERSON
          gender 'male))
```

```
(PHRASE
  pattern 'she
  concept (PERSON
          gender 'female))
```

```
(PHRASE
  pattern 'i
  concept (PERSON
          gender 'male
          name 'uri
          degree 'MSc))
```

```
(PHRASE
  pattern 'you
  concept (PERSON
          gender 'male
          name 'erik
          age 'N+
          degree 'MSc))
```

```
(PHRASE
  pattern 'uri
  concept (PERSON
          gender 'male
          name 'uri))
```

```
(PHRASE
  pattern 'erik
  concept (PERSON
          gender 'male
          name 'erik))
```

B.6 Verb Inflections

In MCRINA there is no morphological analysis, and verb inflections are given explicitly. These entries reside in the file verb-inflections.t.

```
////////////////////////////////////  
(ask asked asked asking asks)  
(bog bogged bogged bogging bogs)  
(bring brought brought bringing brings)  
(buy bought bought buying buys)  
(call called called calling calls)  
(decide decided decided deciding decides)  
(discuss discussed discussed discussing discusses)  
(dress dressed dressed dressing dresses)  
(explain explained explained explaining explains)  
(fall fell fallen falling falls)  
(get got got getting gets)  
(give gave given giving gives)  
(go went gone going goes)  
(make made made making makes)  
(need needed needed needing needs)  
(propel propelled propelled propelling propels)  
(put put put putting puts)  
(run ran run running runs)  
(sell sold sold selling sells)  
(show showed showed showing shows)  
(stash stashed stashed stashing stashes)  
(store stored stored storing stores)  
(take took taken taking takes)  
(think thought thought thinking thinks)  
(throw threw thrown throwing throws)  
(toss tossed tossed tossing tosses)  
(walk walked walked walking walks)  
(want wanted wanted wanting wants)
```

B.7 Interfacing with GATE

The interface of MCRINA with GATE is by several function calls, which are described here briefly. This section should not replace reading the GATE manual itself [Mueller87]. In GATE, *web* is the basic slot-filler representation object. Pattern matching, unification and instantiation are all in terms of webs.

(1) WEB\$CREATE

This function takes a web definition and returns an instantiated web. For example,

```
> (web$create '(PERSON name 'erik degree 'phd))
```

(2) WEB\$SET-SLOT

This function sets the designated slot by a new value:

```
> (web$set-slot ^web39 age 28)
```

(3) WEB\$GET-SLOT

This function returns the value of the designated slot:

```
> (web$get-slot ^web39 age)
```

(4) WEB\$UNIFY

This function unifies two webs and returns a binding list.

```
> (web$unify ^web1 ^web2 '(t (x 6)))
```

(5) WEB\$INSTANTIATE

This function takes a web template and instantiates it relative to a given binding list:

```
> (web$instantiate web^2 '(t (x 6)))
```

(6) WEBS\$PROVE

This function takes a web template and proves it relative to (a) a binding list, (b) a database of facts, (the **context**, in our case), and (c) a database of rules.

```
> (web$prove ^web25 '(t (x 6)) *context* *prules*)
```

In the code below there few are other calls to GATE, which are variations of the calls above.

B.8 The Code

The code itself is given below. This code resides in several files. Each file can be recognized here by the "herald" form at its head.

```
*****
;
; MCRINA
; Version 1.3
;
; ** Copyright (c) 1985 by Uri Zernik. All Rights Reserved.
;
; Originally written: April 1985
; Latest version: January 1987
;
*****

(herald parser (read-table *gate-read-table*))
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;PAR
;top level parser
;1. read new word (into *word*)
;2. convert a word to its corresponding web (into *web*)
;3. activate phrases triggered by a word
;   (into Active-Phrase-Stack)
;4. package the current case frame (into *case*)
;5. check active case frames (on Active-Case-Stack)
;6. check active phrases (on Active-Phrase-Stack)
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
(define (par sentence)
  (loop (for word in (append sentence ' ( ( )))
        (do (read-word word)
            (wordTOweb *word*)
            (webTOcase *web*)
            (phrase-trigger (index-of *case*))
            (scan-cases *case*)
            (scan-phrases)
        )))
)))
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```

;PARSE, CPARSE
;the actual calls to the parser
;cparse (continue parse) enables continuing
;parsing the same sentece
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (parse s) (init-parser) (par s))
(define (cparse s) (par s))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;INIT-PARSE
;initialize structures
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (init-parser)
  (init noun-morph)
  (init verb-morph)
  (lset *subject* nil)
  (lset *verb* nil)
  (lset *word* nil)
  (lset *next-word* nil)
  (lset *case* nil)
  (lset *web* nil)
  (lset *next-web* nil)
  (init case-frame-input-stream)
  (init active-phrase-stack))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;READ-WORD
;process a new word
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (read-word word)
  (set *prev-word* *word*)
  (set *word* *next-word*)
  (set *next-word* word)
  (and *word*
    (format *gate-output*
      " &**** reading word: A &" *word*))
  *word*)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;wordTOweb

```

```

;fetch the mini-concept for the word
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (wordTOweb word)
  (set *prev-web* *web*)
  (set *web* *next-web*)
  (set *next-web* (table-lookup *next-word*))
  (if *web* (po *web*))
  *web*)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;TABLE-LOOKUP
;find word in tables
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (table-lookup word)
  (if (or (null? word)
        (memq? word '(*break* *period*)))
      *nil-web*
      (or
       (get-entry vl word)
       (get-entry avl word)
       (get-entry nl word)
       (get-entry cl word)
       (get-entry pl word)
       (get-entry dl word)
       (error
        "undeveloped 4 in functions: unknown word: A" word))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;webTOcase
;construct the case
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (webTOcase web)
  (and web
       (test-new-web noun-morph *prev-web* *web* *next-web*)
       (test-new-web verb-morph *prev-web* *web* *next-web*)
       (if (eq? (get-web web 'form) 'adverb)
           (handle-adverb web))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

;PHRASE-TRIGGER
;determine which phrases in the lexicon to access at each point.
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

(define (index-of tcase)
  (and tcase
    (let ((root (get-web tcase 'root))
          (marker (get-web tcase 'marker))
          (modifier (get-web tcase 'modifier)))
      (if (var? root)
          modifier
          (if (memq? modifier '(none nil))
              root
              (concatenate-symbol root modifier))))))

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;PHRASE-TRIGGER
;bring a phrase into Active Phrase Stack
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

(define (phrase-trigger word)
  (and word (add-to-phrases (get-entry *lex* word))))

(define (add-to-phrases entries)
  (map (lambda (x)
        (format *gate-output*
          " &**** triggering phrase: A &"
          (get-web x 'comment))
        (oadd active-phrase-stack (copy-deep x)))
    entries))

(lset *subject* nil)
(lset *verb* nil)
(lset *cur-slot* nil)
(lset *word* nil)
(lset *next-word* nil)
(lset *prev-word* nil)
(lset *case* nil)
(lset *web* nil)
(lset *next-web* nil)

```

```

(lset *prev-web* nil)
(lset *no* 0)

(herald structures (read-table *gate-read-table*))

(define-operation (check-list self))
(define-operation (get-top self))
(define-operation (get-all self))
(define-operation (get-no self))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MAKE-LEX
;make the container of lexical phrases
;interesting operations:
;      oadd: add a new phrase
;      get-entry: get a list of phrases through indices
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (make-lex)
  (labels ((found nil)
           (index nil)
           (entry nil)
           (structure nil))
    (object nil
      -----
      ((INIT self) (set structure nil))
      -----
      ((ODELETE self key)
       (set structure (delq (assq key structure) structure)))
      -----
      ((OADD self phrase)
       (set entry (message-phrase phrase))
       (set index
        (index-of
         (get-path entry '(pattern verb))))
       (if (set found (assq index structure))
           (push (cdr found) entry)
           (push structure (list index entry)))
       (format t " & A" (get-web entry 'comment))
       entry)
      -----
      ((GET-ENTRY self root)

```

```

(loop
  (initial (body nil))
  (for entry in structure)
  (do (and (eq? (car entry) root)
          (set body (cdr entry))))
  (result body))
;-----
((PRINT self stream)
 (format stream "phrasal-lexicon"))
;-----
((PRETTY-PRINT self stream)
 (loop (for entry in structure)
       (do (format stream " &** A " (car entry))
           (loop (for phrase in (cdr entry))
                 (do (newline *gate-output*)
                     (wpl phrase))))))))
;-----
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;MESSAGE-PHRASE
;reorganize the restrictions on case-concepts
;(to make them usefull for the unifier)
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

(define (message-phrase phrase)
  (labels ((phr (makewb phrase)
            (pattern (get-web phr 'pattern)))
           (loop (for slot in (get-slots pattern))
                 (do (set-web (get-web pattern slot) 'type
                              (case slot
                                ((verb) ^verb-case)
                                ((object1 subject object2) ^noun-case))))
                    phr))
    phr))

(set *lex* (make-lex))

(define-operation (check-stack self))

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;MAKE-CASE-STREAM
;make the container of lexical phrases

```

```

;interesting operations:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (make-case-stream)
  (labels ((structure nil)
           (no -1)
           (object nil)
           ;-----
           ((INIT self)
            (set structure nil))
           ;-----
           ((OADD self entry)
            (increment *no*)
            (push structure entry)
            (set-web entry 'no *no*)
            (if (eq? (get-web entry 'type) ^verb-case)
                (set no *no*))
            entry)
           ;-----
           ((GET-NO self) no)
           ;-----
           ((GET-ENTRY self number)
            (loop (for kase in structure)
                  (until (= number (get-web kase 'no)))
                  (result kase)))
           ;-----
           ((GET-ALL self) structure)
           ;-----
           ((GET-TOP self)
            (car structure))
           ;-----
           ((PRINT self stream)
            (format stream "#case-frame-input-stream"))
           ;-----
           ((PRETTY-PRINT self stream)
            (loop (for entry in structure)
                  (do (newline *gate-output*)
                      (wpl entry))))))
           ;-----

(set case-frame-input-stream (make-case-stream))
(set cfis case-frame-input-stream)

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MAKE-PHRASE-STACK
;make the container of lexical phrases
;interesting operations:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (make-phrase-stack)
  (labels ((structure nil))
    (object nil
-----
      ((INIT self)
       (set structure nil))
-----
      ((OADD self entry)
       (push structure entry)
       (phrase-unify entry)
       entry)
-----
      ((GET-ALL self) structure)
-----
      ((CHECK-STACK self)
       (check-active-phrases structure))
-----
      ((GET-TOP self)
       (car structure))
-----
      ((PRINT self stream)
       (format stream "#phrase-stack"))
-----
      ((PRETTY-PRINT self stream)
       (loop (for entry in structure)
             (do (newline *gate-output*)
                 (wpl entry))))))
-----

(set active-phrase-stack (make-phrase-stack))
(set aps active-phrase-stack)

(define (change-slot-name web n1 n2)
  web)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```



```

;MAKE-CASE-STACK
;make the container of lexical cases
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (make-case-stack)
  (labels ((structure nil)
           (object nil)
           ;-----
           ((INIT self)
            (set structure nil))
           ;-----
           ((OADD self entry)
            (push structure entry)
            entry)
           ;-----
           ((ODELETE self entry)
            (set structure (delq entry structure))
            entry)
           ;-----
           ((GET-ALL self) structure)
           ;-----
           ((GET-TOP self)
            (car structure))
           ;-----
           ((PRINT self stream)
            (format stream "#active-case-stack"))
           ;-----
           ((PRETTY-PRINT self stream)
            (loop (for entry in structure)
                  (do (newline *gate-output*)
                      (wpl entry))))))
           ;-----

(set active-case-stack (make-case-stack))
(set acs active-case-stack)

(herald memory (read-table *gate-read-table*))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; INITIALIZE:
; *context* and *ltm* (long term memory) are both defined
; as GATE contexts. *context* is used as working memory

```

```

; in parsing.
; in MCRINA contexts are organized as stacks:
; retrieval is by recency, from the "top" of the context.
; BY DEFAULT, in all functions, WEB is the returned value.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(set *context* (cx$create))
(set *ltm* (cx$create))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MEM-AUGMENT
; input: context, web
; adds web as an assertion to context
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (mem-augment mem obj)
  (cx$assert mem obj))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MEM-UPDATE
; input: context, web
; reorganize the context:
; IF web exists in context:
; THEN move web to top of context.
; ELSE add web at top of context.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (mem-update mem obj)
  (if (cx$retrieve mem obj)
      (cx$retract mem obj)
      (cx$assert mem obj)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MEM-RETRIEVE
; input: context, web
; look for an object matching web in the context
; search by recency
; IF matching object exists in context:
; THEN move object to top of context.
; RETURN object

```

```

; ELSE return NIL.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (mem-retrieve mem obj)
  (let ((obj1 (caar (reverse (cx$retrieve mem obj)))))
    (and obj1 (mem-update mem obj1))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MEM-RETRIEVE-BD
; input: context, web, bd
; retrieve relative to a binding list, by recency
; IF matching object exists in context:
; THEN move web to top of context.
; RETURN the binding list
; ELSE return NIL.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (mem-retrieve-bd mem obj bd)
  (let ((obj1 (car (reverse (cx$retrieve-bd mem obj bd)))))
    (and (car obj1)
         (mem-update mem (car obj1))
         (append '(t) (cdr obj1)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MEM-RETRIEVE-OBJ
; input: context, web, bd
; retrieve relative to a binding list, by recency
; IF object exists in context:
; THEN move object to top of context.
; RETURN the object itself.
; ELSE return NIL.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (mem-retrieve-obj mem obj bd)
  (let ((obj1 (car (reverse (cx$retrieve-bd mem obj bd)))))
    (and (car obj1)
         (mem-update mem (car obj1)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MEM-RETRIEVE-ALL
; input: context, web

```

```

; RETURN all matching objects from the context
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

(define (mem-retrieve-all mem obj)
  (map (lambda (x) (car (del alikeq? '(t) x)))
       (cx$retrieve mem obj)))

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;MEM-GET-ALL
; input: context, web
; RETURN a list of objects
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

(define (mem-get-all mem)
  (cx$retrieve mem *xx*))

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;MEM-INIT
; input: context
; RETURN an empty context
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

(define (mem-init mem)
  (loop
   (for x in (mem-retrieve-all mem *dummy*))
   (do (cx$retract mem x))))

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; CALLS to GATE:
;   cx$create
;   cx$assert
;   cx$retrieve
;   cx$retrieve-bd
;   cx$init
;   cx$retract
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

(herald reference (read-table *gate-read-table*))

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

```

```

;REFERENCE-RESOLVE
;find the appropriate referent for a reference
;1. find concept for reference in lexicon
;2. find object for concept in memory
;TCASE is the current case-frame
;SORT is determiner or ()
;CONCEPTS is a list of retrieved concepts
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

```

```

(define (reference-resolve tcase tsort)
  (labels
    ((concepts (get-concept tcase))
     (inst1 nil))
    (or concepts
      (error "undeveloped 8 (in reference):
              definition not found for word A"
              (root-of tcase)))
    (set inst1 (case (get-web tcase 'word-type)
                  ((name)
                   (list (mem-retrieve *context*
                                       (get-web (car concepts) 'concept))))
                  ((pronoun)
                   (pronoun-resolve tcase concepts))
                  (else
                   (case (get-web tcase 'determiner)
                       ((some this)
                        (list (augment *context* (car concepts))))
                       ((the)
                        (list (the-retrieve concepts)))
                       ((none () a)
                        (list (a-retrieve concepts)))
                       (else
                        (if (web? (get-web tcase 'determiner))
                            (list (possessive-retrieve tcase (car concepts)))
                            (error "undeveloped 5 (in reference):
                                    strange reference"))))))))
    (splice-web tcase 'concept inst1)
  tcase))

```

```

(define (splice-web web slot con)
  (remove-slot-web web slot)

```

```

(loop (for elem in con)
      (do (add-web web slot elem))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;PRONOUN-RESOLVE
;find referent for a pronoun (must be in WM)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (pronoun-resolve tcase concepts)
  (case (function-of tcase)
    ((subjective)
     (list (mem-retrieve *context*
                       (get-web (car concepts) 'concept))))
    ((objective)
     (if (eq? (root-of tcase) 'it)
         (mem-retrieve-all *context*
                           (get-web (car concepts) 'concept))
         (list (retrieve-not-subject
               (get-web (car concepts) 'concept)))))
    ((possessive)
     (list (mem-retrieve *context*
                       (get-web (car concepts) 'concept)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;RETRIEVE-not-SUBJECT
;in case of "he took him" (as opposed to "he took himself")
;find referent (for "him") in WM which is different than
;the referent for "he".
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (retrieve-not-subject con)
  (let ((pat (ob$fccreate '(UAND obj
                          (UNOT obj
                            ,(get-web *subject* 'concept)) obj ,con))))
    (mem-retrieve *context* pat)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;GET_CONCEPT
;find the concept for the reference in the lexicon
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(define (get-concept tcase)
  (let ((root (root-of tcase)))
    (map (lambda (x)
          (copy-deep x ))
         (get-entry wl root))))

```

```

;A-RETRIEVE
;instantiate a referent for a NEW reference

```

```

;account for multiple meanings
(define (a-retrieve concepts)
  (mem-augment *context*
    (get-web (car concepts) 'concept)))

```

```

;THE-RETRIEVE
;instantiate a referent for a KNOWN reference
;either:
;1. find it in WM
;2. find it as a role in an existing frame

```

```

;account for one meaning only
;retrieve the frame of that concept if specified
(define (the-retrieve concepts)
  (if (get-web (car concepts) 'pres)
      (mem-update *context* (frame-retrieve (car concepts)))
      (mem-retrieve *context*
        (get-web (car concepts) 'concept))))

```

```

;FRAME-RETRIEVE
;find the presupposition of the concept and hook it up
;i.e: the judge: find a trial script and instantiate a
;           judge in that script
;(must be in WM)

```

```

(set *con-var* (ob$fccreate '(uvar name 'concept)))

(define (frame-retrieve con)
  (labels ((frame (get-web con 'pres))
            (concept (get-web con 'concept))
            (path (find-path-by-value frame *con-var*))
            (new-frame (remove-path (copy-deep frame) path))
            (inst-frame (mem-retrieve *context* new-frame))
            (bdg (unify1 frame inst-frame '(t) '())))
    (if inst-frame
        (if bdg
            (bd-lookup 'concept bdg)
            (set-path inst-frame path concept))
        (error "no frame A found in the context" new-frame))
    ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;POSSESSIVE-RETRIEVE
;find referent in case of a possessive "his brother"
;three cases:
;1. his trip: a new frame where possessor "has" the entire frame
;2. his father: a new frame where the possessor has-a role
;3. his money: if the object is a soc-posses then
;   instantiate
;           ((head possess)
;           (actor possessor)
;           (obj possessed))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (possessive-retrieve tcase defl)
  (labels
    ((con (car (get-concept tcase)))
     (possessor (get-web defl 'concept))
     (possessed nil))
    (set-path tcase '(determiner concept) possessor)
    (if (set possessed (get-web possessor (get-web con 'name)))
        (error: "error 8: a role holder A of A not found"
                (get-web con 'name)
                possessor))
    (mem-update *context* possessor)
    (mem-update *context* possessed)))

```


////////////////////////////////////

(herald phrase-unify (read-table *gate-read-table*))

////////////////////////////////////

;PHRASE-UNIFY

;

;spawns demons for each one of the pattern cases

;each demon is matched against the current case in the text

////////////////////////////////////

(define (phrase-unify phrase)

(labels

((pattern (get-web phrase 'pattern))

(verb-no (get-no cfis))

(slots (get-slots pattern)))

(set-web phrase 'unmatched-cases slots)

(set-web phrase 'bd '(t))

(loop

(initial (case-frame nil))

(for slot in slots)

(do (set case-frame (get-web pattern slot))

(set-web case-frame 'phrase phrase)

(set-web case-frame 'slot slot)

(set-web case-frame 'no

(case slot

((subject) (~ verb-no 1))

((verb) verb-no)

((object1) (+ verb-no 1))

((object2) (+ verb-no 2))))

(oadd active-case-stack case-frame)

(if (< (get-web case-frame 'no) *no*)

(case-unify-prev case-frame)))

(result phrase)))

////////////////////////////////////

;CASE-UNIFY

; matches a lexical case frame with its designated

; one on the input stream

////////////////////////////////////

```

(define (case-unify lcase icase)
  (labels ((phrase (get-web lcase 'phrase))
            (umc (get-web phrase 'unmatched-cases))
            (slot (get-web lcase 'slot))
            (bd (get-web phrase 'bd)))
    (if (= (get-web lcase 'no) (get-web icase 'no))
        (block
         (set bd (unify1 lcase icase bd '(phrase slot)))
         (if bd
              (set-web phrase 'unmatched-cases (delq slot umc))
              (block
               (set-web phrase 'failed-case slot)
               (format t " & *** failed case. phrase: A lcase: A icase A"
                       phrase lcase icase)))
         (odelete active-case-stack lcase)
         (set-web phrase 'bd bd)))
        bd))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;CASE-UNIFY-PREV
; matches a lexical case frame with previous one (for subject)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(define (case-unify-prev lcase)
  (labels ((phrase (get-web lcase 'phrase))
            (umc (get-web phrase 'unmatched-cases))
            (slot (get-web lcase 'slot))
            (bd (get-web phrase 'bd))
            (icases (get-all case-frame-input-stream)))
    (loop
     (for icase in icases)
     (do
      (if (= (get-web lcase 'no) (get-web icase 'no))
          (block
           (set bd (unify1 lcase icase bd '(phrase slot)))
           (if bd
                (set-web phrase 'unmatched-cases (delq slot umc))
                (block
                 (set-web phrase 'failed-case slot)
                 (format t " & *** failed case. phrase: A lcase: A icase A"
                         phrase lcase icase)))
           (odelete active-case-stack lcase)
           (set-web phrase 'bd bd))))
      bd)))

```

```

        (delete active-case-stack lcase)
        (set-web phrase 'bd bd)))
(until
  (= (get-web lcase 'no) (get-web icase 'no)))
  bd))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SCAN-CASES
;the top-level call for phrase unification
;spawns demons for each one of the pattern cases
;each demon is matched against the current case in the text
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (scan-cases icase)
  (if icase
    (labels
      ((structure (get-all active-case-stack)))
      (loop (for lcase in structure)
            (do (case-unify lcase icase))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SCAN-PHRASES
;the top-level call for phrase unification
;spawns demons for each one of the pattern cases
;each demon is matched against the current case in the text
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (scan-phrases)
  (loop
    (for phrase in (get-all active-phrase-stack))
    (initial (bd nil) (umc nil) (result nil))
    (do
      (set bd (get-web phrase 'bd))
      (set pres (get-web phrase 'pres))
      (set umc (get-web phrase 'unmatched-cases))
      (and (null? umc) bd
           (if pres
              (set-web phrase 'bd (phrase-prove phrase))))

      (block
        (push result phrase)

```

```

      (if (get-web phrase 'concept)
          (phrase-instantiate phrase)
          (phrase-modify phrase))))
    (result result))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;PHRASE-INSTANTIATE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(define (phrase-instantiate phrase)
  (let ((bd (get-web phrase 'bd))
        (con (get-web phrase 'concept))
        (con1 nil))
    (set con1 (ob$instantiate con bd))
    (mem-augment *context* con1)
    (format t " & instantiated concept  A of phrase  A  &"
            con1 phrase)

    (po con1)
    con1))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;PHRASE-MODIFY
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(define (phrase-modify phrase)
  (labels
    ((bd (get-web phrase 'bd))
     (modify (get-web phrase 'modify))
     (modify1 (block
                (format t " &*****"
                        bd A modify A" bd modify)

                        (ob$instantiate modify bd)))
     (con1 (get-web modify1 'concept))
     (path (get-web modify1 'path))
     (val (get-web modify1 'val)))
    (format t " &*****"
            path A con1 A val A mod A" path con1 val modify1)
    (set-path con1 path val)

```



```

        (word-analysis five-tuple 1 2)
        (word-analysis five-tuple 2 3))))))
    (word-analysis five-tuple 3 4)
    (word-analysis five-tuple 4 5)
clist))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; FUNC-AV
; the function given to the aux-verb-list
; INPUT: a list of this form:
;         (be was been being is)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(define (func-av five-tuple)
  (labels ((avlist nil)
            ((word-analysis five-tuple sn suff)
             (push avlist
                   (list (nth five-tuple sn)
                         (ob$create
                          `(WORD-SPEC
                            root ',(nth five-tuple 0)
                            form 'aux-verb
                            suff ',suff)))))))
    (if
     (eq? (nth five-tuple 0)
           (nth five-tuple 1))
     (word-analysis five-tuple 0 123)
     (block
      (word-analysis five-tuple 0 1)
      (if
       (eq? (nth five-tuple 1)
             (nth five-tuple 2))
       (word-analysis five-tuple 1 23)
       (block
        (word-analysis five-tuple 1 2)
        (word-analysis five-tuple 2 3))))))
     (word-analysis five-tuple 3 4)
     (word-analysis five-tuple 4 5)avlist))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; FUNC-AV

```



```

                '(subjective) three-tuple)
(push-new (nth three-tuple 4) nil
          (nth three-tuple 0)
          '(possessive) three-tuple)
(push-new (nth three-tuple 3) nil
          (nth three-tuple 0)
          '(objective) three-tuple))))
(else ; not a pronoun or a name
(format t " &hi three A" three-tuple)
(if (eq? (car three-tuple) (cadr three-tuple))
    (push-new (nth three-tuple 0) 'common
              (nth three-tuple 0)
              '(() three-tuple)

              (block
                (push-new (nth three-tuple 0) 'single
                          (nth three-tuple 0)
                          '(() three-tuple)

                          (push-new (nth three-tuple 1) 'plural
                                    (nth three-tuple 0)
                                    '(() three-tuple))))))
    alist))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; FUNC-A
; the function given to the a-list
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (func-a two-tuple)
  (labels ((alist nil))
    (push alist (list (car two-tuple) (ob$create
                               '(WORD-SPEC
                                 form ', (cadr two-tuple)
                                 root ', (car two-tuple))))))alist))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; FUNC-d
; the function given to the d-list
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (func-d four-tuple)

```

```

(labels ((alist nil)
  ((word-analysis four-tuple sn suff)
    (push alist
      (list (nth four-tuple sn) (ob$fcreeate
        `(WORD-SPEC
          root ',(nth four-tuple 0)
          form 'description
          suff ',suff))))))
  (if
    (eq? (nth four-tuple 0)
      (nth four-tuple 1))
    (word-analysis four-tuple 0 12)
    (block
      (word-analysis four-tuple 0 1)
      (word-analysis four-tuple 1 2)))

    (word-analysis four-tuple 2 3)
    (word-analysis four-tuple 3 4)
  alist))

```

```

(set vl (make-word-list 'vl func-v verbs))
(set avl (make-word-list 'avl func-av ftp-list))
(set nl (make-word-list 'nl func-n nouns))
(set dl (make-word-list 'dl func-d ftp-list))
(set pl (make-word-list 'al func-a ftp-list))
(set cl (make-word-list 'al func-a ftp-list))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MAKE-WORLD-LIST
; A T object which defines access operations.
; world list is used to define semantics of lexical words.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(define (make-world-list name)
  (labels ((blist nil)
    (tuple nil)
    (pat nil))
    (object nil
      ((init self)

```

```

      (set blist nil))
((delete self word)
 (set blist (delq (assq word blist) blist)))
((oadd self phr)
 (set tuple (ob$create phr))
 (make-name-entry tuple)
 (if (assq (set pat
           (get-web tuple 'pattern)) blist)
     (push (cdr (assq pat blist)) tuple)
     (push blist (list pat tuple)))
 (format *gate-output* " A " pat))
((print self stream)
 (format stream "{ A}" name))
((pretty-print self stream)
 (loop (for entry in blist)
       (do (format t " % A A"
                   (car entry) (cdr entry))))))
((get-entry self word)
 (or (cdr (assq word blist)) nil))))

(define (make-name-entry phr)
  (if (and (get-path phr '(concept name))
          (eq? (get-path phr '(concept type)) ^person))
      (push *name-list* (list (get-path phr '(concept name))
                              nil
                              'name))))

(define (make-item-list)
  (let ((clist nil))
    (object nil
      ((init self) (set clist nil))
      ((oadd self item) (set clist (cons item clist)))
      ((get-entry self word) (assq word clist))
      ((pretty-print self stream)
       (map (lambda (x) (wp x)) clist))))))

(set verbs (make-item-list))
(set nouns (make-item-list))
(set ftp-list (make-item-list))

(set *name-list* nil)

```

```

(set wl (make-world-list 'wl))

(herald morphology (read-table *gate-read-table*))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MAKE-RULE-LIST
;converts rules into internal representation (webs)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (make-rule-list rule-list)
  (loop (initial (body nil))
        (for rule in rule-list)
        (do (push body
                (ob$create '(MORPH-RULE prev ,(car rule)
                           curr ,(cadr rule)
                           next ,(caddr rule)
                           thus ,(caddr rule)
                           done ,(caddr (cdr rule))))))
          (result body)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MATCH-RULES
;try out matching all the rules on the list
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (match-rules prev curr next form rule-list mo)
  (loop (for rule in rule-list)
        (initial (temp nil))
        (do ;(px rule)
            (set temp (match-rule prev curr next rule form)))
        (until temp)
        (result temp)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MATCH-RULE
;try matching a single rule
;return a binding list if matched
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(lset *match-rule-ob* (ob$create '(MORPH-RULE)))

(define (match-rule prev curr next rule form)
  (labels ((wb *match-rule-ob*))

```

```

        (bd nil))
      (ob$set wb 'prev prev)
      (ob$set wb 'curr curr)
      (ob$set wb 'next next)
      (set bd (unify1 rule wb '(t) '(thus done)))
      (and bd (loop (for slot in (get-slots (get-web rule 'thus)))
                    (do (set-web form slot
                            (get-path rule (list 'thus slot))))
                        (result (cons (inst! form bd)
                                      (get-path1 rule '(done done))))))))

(define-operation (oform self))
(define-operation (test-new-web self prev curr next))
(define-settable-operation (drules self))

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;MAKE-MORPH
;the object containing the rules
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
(define (make-morph init-form)
  (labels ((rules nil)
            (temp nil)
            (form (ob$create init-form)))
    (object nil
      ((init self)
       (set form (ob$create init-form)))
      ((test-new-web self prev curr next)
       (set temp (match-rules prev curr next form rules self))
       (if (eq? (cdr temp) 'ref)
           (set-ref form self)
           (if (eq? (cdr temp) 't)
               (set-case form self)
               nil)))
      form)
    ((drules self) rules)
    (((setter drules) self entries)
     (set rules entries)nil)
    ((oform self) form))))

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

```

```

;VERB
;setting up rules for verb morphology
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(set init-verb '(VERB-CASE
                form 'verb
                root 'none
                lex 'none
                voice 'active
                polarity 'pos
                tense 'present
                word-type nil
                modifier 'none))

(set verb-morph (make-morph init-verb))

(set (drules verb-morph) (make-rule-list '(
  (?? (WORD-SPEC form 'noun)
      (WORD-SPEC form (UOR obj 'aux-verb obj 'verb) suff ?x)
      (CASE-MODIFIER suff ?x))

  (?? (WORD-SPEC root 'have)
      (WORD-SPEC suff (UOR obj '3 obj '23))
      (CASE-MODIFIER aspect 'perfect))

  (?? (WORD-SPEC root 'be)
      (WORD-SPEC form (UNOT obj (UOR obj 'aux-verb obj 'verb)))
      (CASE-MODIFIER root 'isa)
      (SIGNAL done 't))

  (?? (WORD-SPEC root 'be)
      (WORD-SPEC suff (UOR obj '3 obj '23))
      (CASE-MODIFIER voice 'passive))

  (?? (WORD-SPEC root 'be) (WORD-SPEC form (UOR verb 'aux-verb)
      suff '4)
      (CASE-MODIFIER mode 'continuous))

  (?? (WORD-SPEC form 'verb root ?x) ??
      (CASE-MODIFIER root ?x)
      (SIGNAL done 't))

```

)))

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;NOUN
;setting up rules for noun morphology
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(set init-noun '(NOUN-CASE
                form 'noun
                determiner 'none ; the a some
                marker 'none; to from by
                description 'none; small yellow
                lex 'none; boys, boy, men, man
                done nil
                function nil; subjective
                word-type nil)); name pronoun

(set noun-morph (make-morph init-noun))

(set (drules noun-morph) (make-rule-list '(

  (?? (WORD-SPEC form 'prep root ?x)
      (WORD-SPEC form (UOR obj 'determiner obj 'possessive
                      obj 'description obj 'noun obj 'pronoun))
      (CASE-MODIFIER marker ?x))

  (?? (WORD-SPEC form 'determiner root ?x)
      (WORD-SPEC form (UOR obj 'noun obj 'role))
      (CASE-MODIFIER determiner ?x))

  (?? (WORD-SPEC word-type 'pronoun lex ?1
      function ?p root ?x) ??
      (CASE-MODIFIER root ?x lex ?1
          function ?p word-type 'pronoun)
      (SIGNAL done 't))

  (?? (WORD-SPEC word-type 'pronoun
      function 'possessive lex ?1 root ?x)
      (WORD-SPEC form 'noun word-type 'none)
      (CASE-MODIFIER determiner
          (NOUN-CASE
```

```

        word-type 'pronoun
        function 'possessive
        root ?x
        lex ?l)
(SIGNAL done 'ref))

(?? (WORD-SPEC form 'noun root ?x lex ?l word-type ?y
      function (UAND obj (UNOT obj 'possessive)
                obj ?z)) ??
  (CASE-MODIFIER root ?x
    lex ?l
    function ?z
    word-type ?y)
  (SIGNAL done 't))
  )))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(herald reader (read-table *gate-read-table*))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; INITIALIZE: *last-open-file* can save you when you need
; to close a file
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(set *last-open-file* nil)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; LOAD-FROM-FILE
; input: structure, file
; initialize the structure (a lexicon) and load the phrases
; in the input into the structure.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (load-from-file lex file)
  (unwind-protect
    (let ((stream (open file '(in))))
      (set *last-open-file* stream)
      (init lex)
      (loop (initial (expr nil))
            (while (not (eq? (set expr

```



```

                (read-object stream *gate-read-table*) *eof*))
                (do (oadd lex expr)))
    (close *last-open-file*))

(define (load-phrase x y) (load-from-file x y))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; ADD-FROM-FILE
; input: structure, file
; do NOT initialize the structure (a lexicon)
; load the phrases in the input into the structure.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (add-from-file lex file)
  (unwind-protect
    (let ((stream (open file '(in))))
      (set *last-open-file* stream)
      (loop (initial (expr nil))
            (while (not (eq? (set expr
                                (read-object stream *gate-read-table*)) *eof*))
                    (do ;(format t " %- A" (cadr (assq 'comment expr)))
                        (oadd lex expr))))
            (close *last-open-file*)))
    ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; ADD-FROM-LIST
; input: structure, list
; do NOT initialize the structure (a lexicon)
; load the phrases in the list into the structure.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (add-from-list lex nlist)
  (let ((stream *gate-input*))
    (loop (for expr in nlist)
          (do (oadd lex expr)))
    ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; ADD-FROM-TEXT
; input: structure, file
; do NOT initialize the structure (a lexicon)

```

```
; load phrases as they are typed in the input-window
; into the structure. terminated by '()' (e.g., NIL)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (add-from-text lex)
  (let ((stream *gate-input*))
    (loop (initial (expr nil))
          (while (not (eq? (set expr
                              (read-object stream *gate-read-table*)) nil)))
          (do (oadd lex expr)
              )))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; CALLS
; all these utilities rely on OADD, an object-oriented
; operation, defined within each structure.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```