PARSING PARADIGMS AND LANGUAGE LEARNING

Michael G. Dyer
Uri Zernik

April 1986
CSD-860079

Parsing Paradigms and Language Learning

Michael G. Dyer
Uri Zernik

Artificial Intelligence Laboratory
Computer Science Department
University of California
Los Angeles, CA 90024

1. Introduction

In many artificial intelligence applications, where it is advantageous to use natural language input, text must first be converted into an internal representation, suitable for further manipulation by inference and search processes within the specified task domain. 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:

(a) In **database** applications, 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.

(b) 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.

(c) 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 use this to guide comprehension.

(d) **Information-gathering and analysis** systems 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 failed. 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.

1.1 Problems in Language Comprehension

Some of the problems encountered in parsing are described below:

In Proceedings of First Annual Artificial Intelligence & Advanced Computer Technology Conference, Long Beach, CA 1985.

(a) **Reference resolution**: For example, in: `John got hit by Bill. He fell down`, which of the two characters does `he` refer to? The solution to this problem lies in understanding the causality of actions.

(b) **Word-sense disambiguation**: For example, in: `She looked up the word`, as opposed to: `She looked up when he entered`, two separate word-senses are involved. Ambiguous word-senses must be resolved using the context as well as semantic and syntactic clues.

(c) **Idioms**: A sentence such as: `Her father put his foot down when he heard her plans` should not be interpreted literally as a movement of a body-part down toward the floor, but rather that the father expressed an objection. A program must be able to distinguish figurative from literal phrases in text.

(d) **Ill-formed input**: In people's communication, most sentences do not perfectly conform to textbook grammar. Ill-formedness appears through ellipsis (e.g., continuing the sentences and thoughts of others), corrections (e.g., changing utterance plans while in the middle of a sentence) and style/space circumstances (e.g., telegraphic navy messages).

(e) **Feature-interaction**: Linguistic features are easy to account for when they appear in isolation. For example, one feature is *recipient movement* in English, which causes `give` verbs to appear in two different forms, as in: `I lent Mary the book` and `I lent the book to Mary`. Another English feature is *passive-voice*. Now what is the correct interaction of these two features? Clearly, the following is incorrect: `To Mary was lent the book`.

In all these cases, features which appear natural and pass unnoticed by a human listener pose non-trivial problems for a computer-programmed analyzer.

1.2 Problems in Language Learning

Learning is called for when the system fails to analyze input text using the knowledge it has been set up with. This is similar to learning as experienced by humans. Naturally, two groups experiencing extensive learning are children and foreign-language speakers, who face similar problems in these areas:

(a) **Novel phrases**: For example, suppose that both `look` and `up` are familiar when the program receives the input: `How do I look up the word "text-editor" in the help system?` A human can learn the new phrase, `look up`, from the context. Since phrases and word senses are ever-changing, programs must also be able to acquire the meanings of new lexical items from the context.

(b) **Grammar-idiom interaction**: The interaction of grammar rules with idioms of various types has not been addressed by theories of grammar. Can the idiom: `I gave her a piece of my mind` interact with recipient movement and thus appear as: `I gave a piece of my mind to her`? Or can the idiom: `Her father put his foot down` interact with the passive voice: `The foot was put down by her father`? Since these phrases appear to be idiosyncratic, an expensive solution would be to include all possible correct interactions in the lexicon.

In order to communicate effectively in natural language, parsers must address the problems listed in Section 1.1. However, in order to appear robust and to face new situations, a program must learn new phrases and acquire the correct interaction between idioms and grammar.

2. Three Paradigms in Parsing

In the following sections we introduce three trends in parsing. These approaches do not really contradict one another, but rather each emphasize different aspects of language processing. Syntax-based parsing emphasizes aspects of feature interaction, lexicon-based parsing focuses on word-sense disambiguation, while expectation-based parsing highlights interaction with the context.

2.1 Syntax-Based Parsers

Syntax-based parsers as opposed to *semantics-based* parsers proceed in two stages. First, input tokens are anticipated and accepted according to syntactic rules. Second, the semantic structure for the entire sentence is constructed from the semantics of its constituents.

2.1.1 Augmented Transition Networks

Traditionally, Augmented Transition Networks (ATNs) [Woods70] have provided the mechanism for syntax-based parsers. In parsing, a sentence is accepted or rejected according to grammar rules and a semantic structure may be generated as a by-product. A typical use for this approach is in the generation of predicate-calculus expressions, applicable for querying a database. For example, in the figure below we show a sentence and the logical expression constructed by the parser.

```
Every car that moves is driven by a driver

all(X) : (car(X) & moves(X) => exists(Y) : (driver(Y) & drive(Y,X)))
```

Figure 1: **Sentence and Constructed Expression**

ATNs may best be viewed as state machines (which by their nature can express only context-free grammars (CFGs)), augmented in two ways:

(a) *Agreement rules* (which generally require context-sensitivity) can be implemented by the use of *registers*. For example, number and tense agreement along verb phrases can be maintained in dedicated registers and tested in ATN transitions.

(b) The construction of the required logical structure is accomplished by *actions* which are associated with the transitions.

2.1.2 Logic Parsing

With the advent of PROLOG and logic programming, the focus in syntax-based parsing has shifted to Definite Clause Grammars (DCGs) [Pereira80]. Since DCGs supersede ATNs in both theoretical interest and potential application, we will describe parsing using DCGs. Again, it is helpful to think of DCGs as extensions to CFGs.

Definite Clause Grammars as Extended Context-Free Grammars

The simplified set of rules below demonstrates the ease of writing a parser, simply by specifying a number of context-free rules. (To be readable by PROLOG, these rules need to be translated into PROLOG clauses. This translation can be performed by a simple preprocessor.) The parsing itself is carried out by any standard PROLOG interpreter. For example, the sentence: `John loves Mary` can be parsed by a set of context-free rules such as:

```
sentence --> noun-phrase, verb-phrase
noun-phrase --> name
verb-phrase --> trans-verb, noun-phrase
name --> Mary
name --> John
verb-phrase --> loves
```

Figure 2: **Context-Free Rules**

Such grammars are denoted by a set of non-terminals (e.g., noun-phrase), a set of terminals (e.g., John), and a set of rules which allow one non-terminal on the right-hand side and a list of terminals and non-terminals on the left-hand side. DCGs extend CFGs in two ways.

First Extension: Parameters in Non-Terminals

A DCG non-terminal may be any PROLOG term (rather than a simple atom as in CFGs). For example, the first rule in the grammar in Figure 2 is extended as shown in the figure below..

```
sentence --> noun-phrase, verb-phrase

sentence( s(NP,VP) ) --> noun-phrase(N,NP), verb-phrase(N,VP)
```

Figure 3: **Allowing Arguments in DCG Non-Terminals**

This extension enables the following two features:

Structure-construction: The argument to **sentence** (i.e., s(NP,VP)) specifies the construction of the entire parse tree from the structures of its two constituents.

Agreement: The variable N guarantees the "number" agreement between the noun-phrase and the verb-phrase. This prevents sentences such as `We loves Mary`.

Second Extension: Procedure Calls

The right-hand side of a rule may also include transparent procedure calls. For example, in the grammar for the generation of dates as month-day pairs (shown in Figure 4), the single rule is mutated to constrain the days of the month.

```
date --> month, day

date (D,M) --> month(M), [D], {integer(D), 0<D, D<32}
```

Figure 4: **Allowing Transparent Procedures in DCG Rules**

The *transparent* procedure call delimited by { } above, which checks that D is within range, is not reflected as a token in the generated sentence.

Evaluation

Logic-programming has assumed a central role in language processing, especially in implementing syntactic-based parsers, its main advantages being (a) No need to write a parser (PROLOG backtrack and unification provide the mechanism), (b) Rules are declarative, thus perspicuous, (c) DCGs may emulate a variety of linguistic grammars such as case grammar [Fillmore68], and transformational grammar [Akmajian75],

Current problems with DCGs are: (a) Top-down parsing, making use of knowledge structures, is not yet possible. This should go beyond a *syntactic parsing with semantic interpretation* as implemented by McCord [McCord82]. (b) Since the process is controlled by PROLOG itself, it is difficult to handle ill-formed input, or perform relaxed parsing which could be done naturally by semantic parsers. (c) Since syntax and form restrict the allowed sentences, human behavior such as ellipsis and correction is difficult to handle.

2.2 Expectation-Based Parsing

Expectation-Based parsing was developed in ELI [Riesbeck74], SAM [Cullingford78] and PAM [Wilensky83] which viewed comprehension in a broader scope than just determining the syntactic components of sentences. Parsing proceeds relative to a dynamically updated context, and text is said to be understood when the structures brought to mind are linked appropriately. For example, in understanding the following text: `Wilma was hungry. She looked up a restaurant in the Michelin Guide` the first sentence instantiates the satisfy-hunger *goal*. The realization of this goal involves three *subgoals*: find-food, get-food, and eat-food, where the guide-consulting *act* in the second sentence realizes the find-food subgoal. Understanding is accomplished by linking the top-level goal through the subgoal to the action.

How is the verb `look` analyzed in the presence of many different meanings (e.g., **`look at her`**, **`look up the restaurant`**, **`look up the monument`**)? Two types of processes take part in the analysis of `look` in the example above. First, *bottom-up* processes use input clues to distinguish a set of candidate-senses by syntactic and semantic expectations. Two senses of the verb `look` are distinguished: to `look up` in the sense of searching for an item, and to `look up` in the sense of lifting the face vertically. Since there are two candidates, further disambiguation is required. A *top-down* process is used to determine the appropriate meaning by *expectations* stemming from a central structure such as a goal or a script. The expectation here is find-food which is satisfied by the guide-consulting act.

BORIS [Dyer83] implements such expectations by *demons*. A demon can be viewed as an extended procedure. Like a procedure call, a demon can be *spawned* with a set of input arguments. However, unlike a procedure which is executed as soon as it is called, a demon *fires* only when its *test* is satisfied. The test may be any event observed in the system or the appearance of a certain token in the input. A word-sense is associated with a *demon-template*. Figure 5 shows the demon-template for the phrase `look up a guide`. When the word `look` is encountered, the demons in the template are spawned. Each demon expects a set of events. When all the demons have fired, the template is instantiated yielding the appropriate concept.

```
(MTRANS
     actor  ?x    (demon-find-concept person before)
     from   ?y    (demon-find-concept thing after)
                  (demon-find-word 'up)
     to     ?x
     object ?z    (contents ?y)
     instrument
          (ATTEND
               actor  ?x
               object ?o    (default 'eyes)
               to     ?i    (information-source)
     goal
          (D-KNOW
               owner  ?x
               fact   ?z)))
```

Figure 5: **Lexical Entry For "Look up a Guide"**

The entire template denotes an MTRANS (mental-transfer) act, intended to satisfy a goal of obtaining information (D-KNOW) through attending a sense organ (probably the eyes) to an information source. Each variable in the template body obtains its value by the return value of its associated demon. For example, (demon-find-concept person before) is a call for a demon which anticipates a person-type before the verb in the sentence. The concept retrieved by this demon gets bound to the variable ?x.

Two drawbacks characterize this approach, as implemented in BORIS: (a) Lexical expectations are procedural, and thus not perspicuous. (b) Idioms and figurative phrases are difficult to encode. On the other hand, using semantic expectations rather than syntax to direct parsing, has two advantages: (1) Input text can be interpreted if it still makes sense conceptually, even if it does not conform to textbook grammar. (2) Disambiguation is performed conceptually using top-down as well as bottom-up conceptual clues, thus yielding the meaning intended by people.

2.3 Phrase-Based Parsing

It turns out that idioms and figurative phrases are ubiquitous in human communication (e.g., `turn out` itself is a figurative phrase). A *figurative phrase* is a linguistic pattern whose associated meaning cannot be produced from the composition of its constituents. Indeed, an interpretation of the phrase based on the meanings of its constituents often exists, but it carries a different meaning. The fact that this literal interpretation of the figurative phrase exists is a misleading clue in learning and in comprehension. Becker [Becker75] has described a space of phrases ranging in their generality from fixed proverbs: `charity begins at home` through idioms: `lay down the law`, and phrasal verbs: `put up with one's spouse`, `look up the name`, to literal verb phrases, such as: `sit on the chair`. He suggested employing a *phrasal lexicon* to capture this entire range of language structures.

Wilensky [Wilensky81] suggested a knowledge-based approach to the lexicon representation, which was employed in PHRAN [Arens82], using pattern-concept pairs for phrasal entries in the lexicon, as shown in Figure 6.

```
(1)   pattern:    ?x:person look:verb <at ?y:thing>
      concept:    ?x ATTEND eyes to object ?y

(2)   pattern:    ?x:person look:verb up ?y:concept in ?z:ref-book
      concept:    ?x FIND ?y in ?z
```

Figure 6: **Two Phrases in the Lexicon**

The pattern captures the linguistic form of the phrase (the way it appears in text) and the concept encodes its associated conceptual meaning. Parsing is done by matching the pattern with the input sentence, and consequently instantiating the structures in the concept.

The Pattern Notation

In the two patterns above, there are elements which are "fixed", such as the word `up`, while there are elements which are left as variables, such as ?x:person, and ?y:ref-book, which stand for elements of the designated semantic types (person-type and reference-book-type). Patterns do not dictate any specific order, unless the delimiters < and > appear, enforcing an order on the enclosed elements. In principle, pattern (1) could be either one of: `he looked up the guide` or: `he looked it up`. Four questions are raised by this lexical representation:

(a) Token order is not completely specified. What would prevent the generation of undesired sentences such as: `looked he up the Michelin Guide`?

(b) What would account for correct *feature interaction*? How does passive voice interact with lexical patterns? What is the interaction with verbs taking an infinitive form, such as: `Mark decided to look at Mary`? We certainly do not want to enumerate all possible interactions of such pairs of verbs (decide and look, decide and walk, etc.).

(c) What is the selection principle when two or more patterns match the input sentence? How are they disambiguated?

(d) The lexicon holds many finely distinguished phrases. What efficiency measures could be taken to perform pattern matching in a reasonable time?

The solution for the first two problems lies with the grammatical foundations given by *functional grammars* [Kay79] and [Kaplan75] which enable the definition of lexical as well as grammatical patterns in a uniform way. The operation of such a system is described below.

Lexical Patterns and Ordering Patterns

Some lexical patterns for the verb `look`, for example, are given in Figure 7.

(1) ?x:(person,actor) look:verb <at ?y:thing>
(he looked at the wall)

(2) ?x:(person,actor) look:verb <up ?y:dimension>
(he looked up the wall)

(3) ?x:(person,actor) look:verb up ?z:(word-token) in ?y:ref-book
(he looked up the word in the dictionary)

(4) < ?x:thing look:verb ?y:description >
(it looks shabby)

(5) < ?x:thing look:verb like ?y:thing >
(she looked like her sister)

Figure 7: **Lexical Patterns of "Look" Phrases**

In parsing, these patterns interact with a set of *ordering patterns* to form correctly ordered sentences according to English language conventions. Some sample ordering patterns are given in Figure 8.

(6) < ?u:actor ?v:(verb,active-form) >
(7) < ?u:recipient ?v:(verb,passive-form) > by ?w:actor
(8) < ?u:object ?v:(verb,passive-form) > by ?w:actor

Figure 8: **Some Basic Ordering Rules**

These rules encode our knowledge about sentence structure. In many cases, parsing could be based on semantic expectations only. For example, in the sentence: `John took the book` it is clear that the book did not take John, thus the first element is the actor. But in the sentence: `He took Mary` where both actor and object are persons, only sentence-structure rules along with context reveal the appropriate roles.

How Does It All Work?

For example, assume the input sentences are:

(9) `John needed a car.` (10) `He looked up "car" in the Yellow Pages.`

The process, in parsing (10) above, proceeds as follows:

input: **he** Concept: (john1 name:John class:person)

input: looked — Word: look, voice:active
Apply all the lexical patterns of look.
Apply ordering pattern (6) (active-voice, in Figure 8)
Unify the *actor* variable with *john1*.
input: up — Matched in lexical patterns (2) and (3).
input: "car" — Concept: (word1 class:word-token spelling:"car")
input: in the Yellow Pages
Concept: (yel1 name:Yellow Pages class:ref-book)
Satisfy the expectation of pattern (3).
All the expectations of pattern (3) are satisfied,
therefore instantiate the concept of phrase (3).

Disambiguation

Disambiguation is required when two or more phrases match the input. For example, the sentence (10) matches both lexical phrases (7) and (8). The preference principles used in PHRAN [Arens82] are:

(a) Select the phrase with the longer pattern.
(b) Select the phrase with the more specific pattern.

Since patterns (2) and (3) have the same length, pattern (3) is selected by its specificity: *ref-book* is a sub-class of *thing*.

Phrase Interaction

Phrase interaction is required in sentences such as: `Wilma decided to look up the Michelin Guide.` Notice that two lexical patterns are involved. The pattern for `decide`: `?x:(person,actor) decide:verb <to ?y:verb>` and pattern (3) for `look`: `?x:(person,actor) look:verb up ?y:ref-book` The resulting concept combines the concepts of both patterns.

3. Learning Phrases from the Context

Currently, no computer program is capable of acquiring new phrases from context and automatically augmenting its own lexicon. The computer program RINA [Zernik85] presents a first attempt at learning English phrases by modeling a second language speaker. Some examples of learning dialogues are given below. In the first dialogue the program is introduced to an unknown phrase: `take on`. The words `take` and `on` in isolation are familiar to RINA, who also remembers the biblical story. The program interacts with a user who is a native speaker.

David vs. Goliath

User: Remember the story of David and Goliath? David took on Goliath.
RINA: David took Goliath somewhere?
User: No. David took on Goliath.
RINA: He took on him. He won the fight?
User: No. He took him on. David attacked him.
RINA: He took him on. He accepted the challenge?
User: Right.

User: Here is another story. John took on the third exam question.
RINA: He took on a hard problem.

Another dialogue involves `put one's foot down`. Again, the phrase is unknown while its constituents are known:

Going Punk

User: Jenny wanted to go punk, but her father put his foot down.
RINA: He moved his foot down? It does not make sense.
User: No. He put his foot down.
RINA: He put his foot down. He refused to let her go punk.

3.1 Issues in Phrase Acquisition

Three issues must be addressed when learning phrases in context.

(a) **Detecting failures**: What are the indications that the initial interpretation of the phrase `take him on` as "to take a person to a location" is incorrect? Since all the words in the sentence are known, the problem is detected both as a *conceptual discrepancy* (why would he take his enemy anywhere?) and as a *syntactic failure* (the expected location of the assumed physical transfer is missing).

(b) **Determining scope and generality of patterns**: The linguistic pattern of a phrase may be perceived by the learner at various levels of *generality*. For example, in the second dialogue, incorrect generalizations could yield patterns accepting sentences such as:

Her boss put his left foot down.
He moved his foot down.
He put down his foot.
He put down his leg.

A decision is also required about the *scope* of the pattern (i.e., the tokens included in the pattern). For instance, the scope of the pattern in `John put up with Mary` could be (1) *?x:person put:verb up* where `with` is associated with `Mary` or (2) *?x:person put:verb up with ?y:person*, where `with` is associated with `put up`.

(c) **Finding appropriate meanings**: The conceptual meaning of the phrase must be extracted from the current context which may contain many concepts, both appropriate and inappropriate for hypothesis formation. Thus there must be strategies for focusing on appropriate elements in the context.

3.2 Learning by Failure-Analysis

Learning of phrases is an iterative process. The input is a sequence of sentence-context pairs, through which the program refines its current hypothesis about the new phrase. The hypothesis pertains to both the pattern and the concept of the phrase. The basic cycle in the process is:

(a) A sentence is parsed on the background of a conceptual context.
(b) Using the current hypothesis, either the sentence is comprehended smoothly, or a failure is detected.
(c) The analysis of a failure directs the update of the current hypothesis.

The crucial point in this scheme is to obtain from the parser an intelligible analysis of the failure. As an example consider this part of the first dialog:

(1) RINA: `He took on him. He won the fight?`
(2) User: `No. He took him on. David attacked him.`
(3) RINA: `He took him on. He accepted the challenge?`

The current hypothesis is shown in Figure 9.

```
pattern:    ?x:person take:verb <on ?y:person>
concept:    ?x win the conflict with ?y
```

Figure 9: **The First Hypothesis**

Notice that the preposition `on` is attached to the object ?y, thus assuming that the phrase is similar to `He looked at Mary` which cannot produce the following sentence: `He looked her at`. This hypothesis underlies sentence (1) which is erroneous in both its form and its meaning. Two observations should be made by comparing this pattern to sentence 2:

- The object is not preceded by the preposition **on**.
- The preposition **on** does not precede any object.

These comments direct the construction of the new hypothesis:

```
pattern:    ?x:person take:verb on ?y:person
concept:    ?x win the conflict with ?y
```

Figure 10: **The Second Hypothesis**

where the preposition **on** is taken as a modifier of the verb itself, thus correctly generating sentence 3. In Figure 10 the conceptual hypothesis is still incorrect and must itself be modified.

3.3 The Case-Frame Representation

The pattern representation discussed so far does not lend itself to error-analysis as given by the comments above. A pattern matcher matching the input and the pattern would state simply that they do not match. Therefore, RINA's pattern representation is *hierarchical*, given in terms of case-frames. Schematically, the hierarchy is:

```
pattern --> cases
case --> words and concepts
```

The entire pattern is constructed of frames where each frame itself is constructed of single tokens which are words and concepts. Each frame has several slots which contain information about the case pertaining to: (1) its syntactic appearance, (2) its semantic concept and (3) its role in the primary act of the sentence (actor, object, destination, etc.).

3.4.1 Examples The first example-pattern denotes a simple literal verb phrase:

```
take:verb
{id:?x class:person role:actor}
{id:?y class:person role:object}
{id:?z class:location role:destination marker:to}
```

Figure 11: **Pattern for "He took her to school"**

Both the expected actor and object are people, the destination is a location, and the preposition **to** marks the case in the sentence. The second phrase is figurative:

```
take:verb
{id:?x class:person role:actor}
{marker:to determiner:the word:streets}
```

Figure 12: **Pattern for "He took to the streets"**

The second case does not have any standard role, and does not have a semantic concept. Rather, it is represented as a sequence of words. However, the words in the sequence are designated as the *marker*, the *determiner* and the *word* itself.

3.4.2 Advantages of Case-Frames

The advantages of hierarchical case-frame representation over "flat" pattern representation are:

(a) Case-frames provide clear and psychologically valid metrics for disambiguation. The length of a pattern is measured by the number of cases, and its specificity by the restrictions within cases.

(b) Case-frames support ill-formed parsing since pattern matching is done first on the basis of entire frames and even if a syntactic element is missing, the frame can be recovered.

(c) Case-frames support high-level error analysis, describing discrepancies in terms of frame parts (e.g., "the marker is missing").

4. Current Research and Conclusions

RINA maintains a declarative phrasal lexicon and performs demon-based pattern matching. Our implementation is directed towards integrating a PROLOG framework with demons which are used to implement expectations.

Our theoretical interests lie in three areas: (1) exposing learner's strategies in extracting phrase meanings from the context, (2) identifying strategies for error-analysis and error-recovery in forming pattern/concept pairs, and (3) maintaining *connotations* with each phrase. Connotations are represented by establishing associations between the current syntactic/semantic hypothesis and the episodic context within which the learning occurred. This context is later used by RINA in generating examples of phrases that have been learned.

RINA is written as an experimental model of parsing, designed to explore processes underlying abilities to learn the structure and meaning of new phrases automatically from context. RINA currently handles only a few examples (including the **David vs. Goliath** dialog in Section 3) and is undergoing continuing design and development. We hope that the insights being gained from this research will serve as the basis of future natural language systems which are both flexible and robust.

References

[Akmajian75] Akmajian, A. and F. Heny, *Introduction to the Principles of Transformational Syntax*, MIT Press, Cambridge, Mass. (1975).

[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).

[Becker75] Becker, Joseph D., "The Phrasal Lexicon," pp. 70-73 in *Proceedings Interdisciplinary Workshop on Theoretical Issues in Natural Language Processing*, Cambridge, Massachusets (June 1975).

[Cullingford78] Cullingford, R. E., "Script Application: Computer Understanding of Newspaper Stories," 116, Yale University, Department of Computer Science, New Haven, Connecticut (1978).

[Dyer83] Dyer, Michael G., *In-depth understanding: a computer model of integrated processing for narrative comprehension*, MIT Press, Cambridge, MA (1983).

[Fillmore68] Fillmore, C., "The Case for Case," pp. 1-90 in *Universals in Linguistic Theory*, ed. E. Bach R. Harms, Holt, Reinhart and Winston, Chicago (1968).

[Kaplan75] Kaplan, R. M., "On Process Models for Sentence Analysis," pp. 117-135 in *Explorations in Cognition*, ed. D. A. Norman D. E. Rumelhart, Freeman, San Francisco (1975).

[Kay79] Kay, Martin, "Functional Grammar," pp. 142-158 in *Proceedings 5th Annual Meeting of the Berkeley Linguistic Society*, Berkeley, California (1979).

[McCord82] McCord, M. C., "Using Slots and Modifiers in Logic Grammars for Natural Language," *Artificial Intelligence* **18**, pp.327-367 (1982).

[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).

[Riesbeck74] Riesbeck, C. K., "Computational Understanding: Analysis of Sentences and Context," Memo 238, AI Laboratory (1974).

[Shortliffe76] Shortliffe, E. H., *Computer Based Medical Consultation: MYCIN*, American Elsevier (1976).

[Wilensky83] Wilensky, Robert, *Planning and Understanding*, Addison-Wesley , Massachusetts (1983).

[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).

[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).

[Woods70] Woods, W. A., "Transition Network Grammars for Language Analysis," *Communications of the ACM* **13** (1970).

[Zernik85] Zernik, U. and M. G. Dyer, "Towards a Self-Extending Phrasal Lexicon," in *Proceedings 23rd Annual Meeting of the Association of Computational Linguistics*, Chicago, Ill. (1985). (to appear).