In *Proceedings of the First Annual Workshop on Theoretical Issues in Conceptual Information Processing.* TICIP-1, March, 1984.

An Approach to Natural Language Understanding in Rule-based Systems

Robert E. Frederking

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213 U.S.A.

Phone: (412) 578-3048
ARPAnet address: Frederking@CMUA

Abstract

As part of a long-term project to build an intelligent natural language interface for naive users, a parser is being constructed which is designed to integrate all the various knowledge sources necessary for natural language understanding within a single, unified framework, and to tolerate ambiguity and some forms of ungrammaticality via a general mechanism that is part of that framework. The design attempts to achieve this by embedding a rule-based parser in a user-interface production system, and using an adaptation of chart parsing as its basic mechanism. The method used is discussed, and a currently working example is presented. In the example, the system handles ambiguity arising from three different sources, including an ambiguous spelling correction. The production system is being implemented in the OPS5 language.

1. Project Objectives

The long-term goal of this project is to construct a natural language dialogue system which integrates knowledge about syntax, semantics, pragmatics (real-world knowledge), dialogue conventions, and human goals in an elegant and natural way, in order to allow natural English conversations with naive users of knowledge-based systems. In addition, knowledgeable users who are not intimately familiar with natural language systems should be able to add new nouns and verbs to the system. This requires handling ambiguity via a general mechanism built into the architecture of the system, rather than having the definitions of ambiguous words handle the problem internally.

2. System Environment and History

The system serving as the initial testbed for this user interface is the Intelligent Management System [4] being built in the Intelligent Systems Laboratory of the Robotics Institute at Carnegie-Mellon University. The need for natural language communication arises in this setting because many of its users will be managers in a business or factory environment, most of whom will only use the system a few times a week. A manager who uses the system in this fashion would prefer to hold a conversation with it, and not be forced to learn and remember a complex command language, or go through a time-consuming pre-programmed interaction much of which may be irrelevant to his or her particular needs.

In order to achieve this capability, a User Interface Process is being implemented as a production system using the OPS5 [3] production system language. The UIP will have four main components: an English parser, a system interface which can reason about its capabilities, an English generator, and a mechanism for modelling user's goals. It is not intrinsically linked to any one application or database. A production system, in the sense used here, refers to a set of condition-action rules, a working memory which the rules match against and modify, and a built-in conflict resolution mechanism, which selects a rule to fire when many rules match working memory. The decision to use OPS5 was driven by the main goal of integrating different sources of knowledge as smoothly and gracefully as possible; the advantages of a production system approach will be discussed further below, following an example of its use. Construction of this system has begun with the parser and closely related parts of the system interface.

The current version of the parser, known as Psli3 (pronounced "sly three"), was preceded by two earlier attempts (as one might suspect). Each of the three versions was designed to intermingle its use of syntactic and semantic knowledge, integrating the two types of knowledge both in its production rules and in the structures it builds in working memory. Once the decision to use a production system architecture was made, the next question to be answered was what parsing strategy would be best given this architecture. The main consideration, in addition to retaining the capability for knowledge integration provided by the architecture, was to keep the knowledge in the rules as self-contained as possible, in order to make human modification of the system feasible. Reasonable efficiency was only a secondary criterion, since this is a long-range effort not intended for short-term actual use.

The first attempt, Psli1, was designed to be a production system implementation of an expectation-based parser like CA [1]. There are two approaches which have traditionally been used to handle ambiguity within this type of parser. The one approach is to create an expectation for every possibility, each of which can contain an arbitrary set of syntactic and semantic tests to determine

whether its choice is the correct one. The other approach is to do "intelligent error correction", in which the rule in question guesses which choice is correct, and later retracts its guess if another choice proves to be the right one. Both of these methods require hand-crafting each rule associated with a local ambiguity. This is incompatible with the goal of easy extensibility, since the builder of a rule must anticipate all possible conflicts.

The second version, Psli2, attempted to use a search-tree technique, sprouting a leaf from the tree for each possible decision at each point in the utterance, and using best-first search to find a globally consistent meaning. This version was somewhat awkward for a system builder to program, and resulted in much inefficiency, especially when several correct choices were available at the same time (such as two expectations being satisfied simultaneously). Any correct state not chosen for further growth had to be replicated. Moreover, correctly constructed constituents embedded in larger structures that later proved incorrect might have to be reconstructed several times.

Neither of these parsing strategies was sufficient for the stated purposes. A technique was needed which would handle all kinds of ambiguity in a simple, general fashion, without unreasonable inefficiency. It also had to be adaptable to a production system architecture, so that knowledge could be applied in an integrated and intuitively appealing fashion. Starting with chart parsing [7] (explained cogently by Winograd in [13]) and modifying it to meet these goals lead to the development of "semantic chart parsing", so named to indicate the use of semantic information in the chart and in the parser's rules. A technique reported recently by Jardine and Shebs [6] appears to function in a similar fashion, although it was developed by adding parallelism to PHRAN [12], and is not integrated into an overall production system.

3. Example: Parsing a Sentence with Local Ambiguities

Each rule in a chart parser checks for the starting and ending lexical positions of each constituent state it matches, and indicates the starting and ending positions of the state it builds. A constituent is built only once on a given interval, and only those constituents that are part of the interpretation of the whole input are seen in the final result. In a *semantic* chart parser, such as Psli3, a constituent is built only if it is semantically as well as syntactically consistent.

This approach will be demonstrated using the sentence "Show me xn [sic] axle order". This sentence contains three types of local ambiguity. The most noticeable is that there is a spelling mistake, where "xn" could be corrected to be either "an", "in", or "on", using the current dictionary and spelling corrector (a FranzLisp version of [2]). The second local ambiguity is the case relationship of "me" to "show". In this sentence, "me" is the recipient of the action, whereas in "Show me to Mark", it is the object being shown. Thus in a left-to-right parse, the case of "me" is locally ambiguous. The third local ambiguity is whether "axle order" is a noun-noun compound, or two separate nouns whose juxtaposition is a coincidence. Axle orders, axles, and orders are all objects which can be referred to in the current application[1].

Figure 3-1 shows the chart built during the parse of this sentence. Each line segment in the diagram represents a state built by a rule, usually defining a newly found constituent of the sentence. For clarity, the only backlinks shown are those for states which are part of the correct interpretation. The actual representation used for a state is shown in figure 3-2, along with an English translation. This

[1] Actually, "axle" has been substituted for "blade" throughout this example, in order to show an interesting spelling correction using *xn*. The current application is actually a turbine blade factory information system.

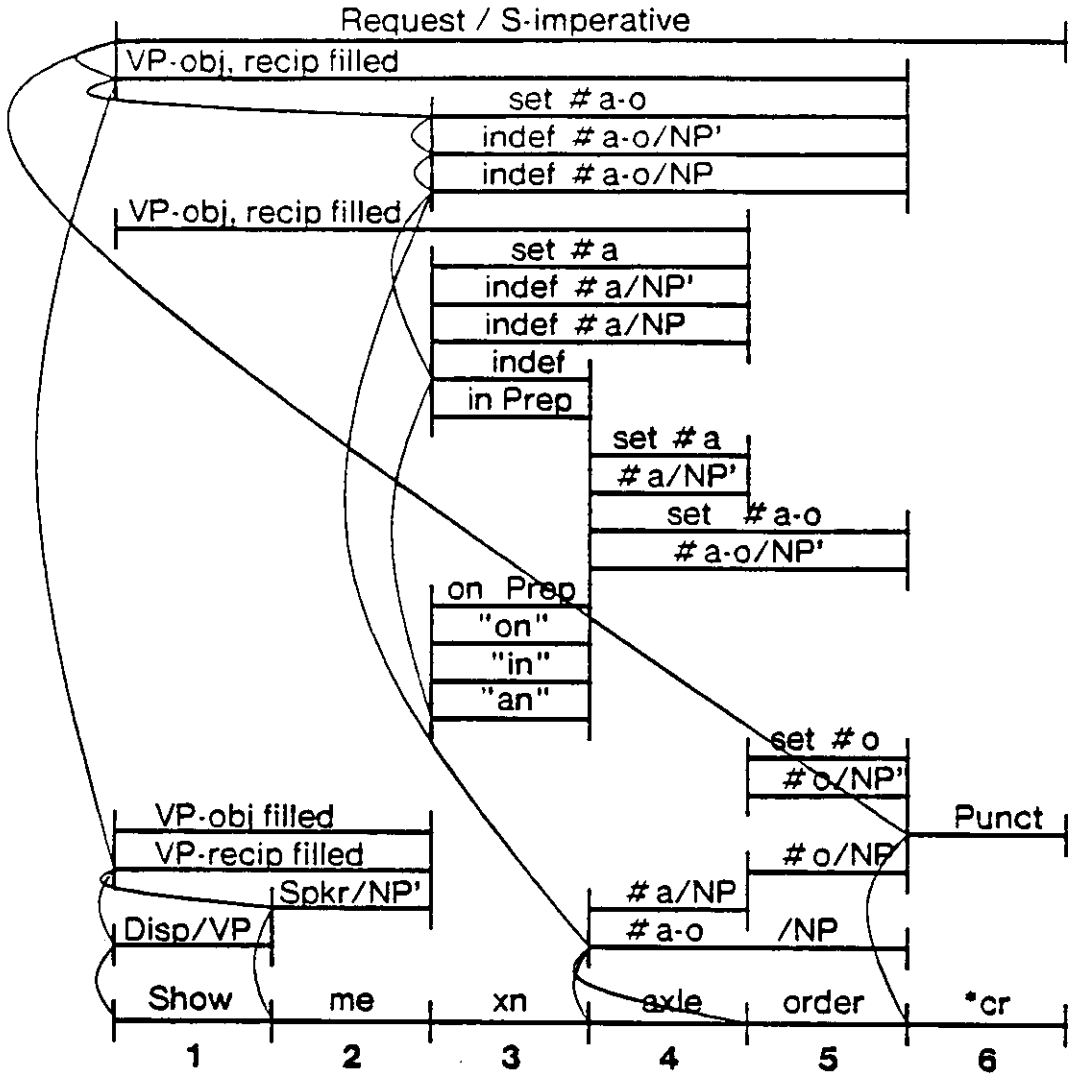

Figure 3-1: Chart built for "Show me xn [sic] axle order"

representation consists of a set of working memory elements, each containing the unique name of this state in its "state" field. A typical production rule and its English translation is shown in figure 3-3.

The first rule that fires in our example places the interpretation of "show" into working memory. This interpretation; shown in figure 3-2, indicates that it is a display action, a verb and a verb phrase, and that it spans the interval [1:1]. Several expectations are also placed into this state, indicating which syntactic and semantic markers correspond to which cases of this verb.

The rule for "me" then fires, placing a description of the current speaker into working memory, and noting that this is a syntactically complete noun phrase located on the interval [2:2]. At this point, two rules fire, one creating a verb phrase on the interval [1:2] with "me" as the recipient of "show", the other with "me" as the object. These can both fire because "me" has a known referent (from pragmatic knowledge), and fits the syntactic and semantic requirements of both cases. These incompatible choices are both entered into the chart, and are both available to higher-level rules looking for a verb phrase on the interval [1:2].

```
(state ↑status posted ↑state g00011)
(act ↑token g00012 ↑action display ↑state g00011 ↑utt 1)
(expect ↑token g00015 ↑con g00012 ↑type e-unmarked ↑marker 1st
        ↑slot recip ↑isa #human ↑default ref ↑state g00011)
(expect ↑token g00015 ↑con g00012 ↑type e-marked ↑marker to
        ↑slot recip ↑isa #human ↑default ref ↑state g00011)
(expect ↑token g00014 ↑con g00012 ↑type e-unmarked ↑marker 2nd
        ↑slot object ↑state g00011)
(expect ↑token g00013 ↑con g00012 ↑type e-marked ↑marker for
        ↑slot bene ↑isa #human ↑default ref ↑state g00011)
(POS ↑type verb ↑state g00011 ↑utt 1)
(POS ↑type verb-phrase ↑state g00011 ↑utt 1)
(word-seq ↑first 1 ↑last 1 ↑prev nil ↑next 2 ↑state g00011 ↑utt 1)
(ancestor ↑ancestor i00010 ↑state g00011 ↑utt 1)
```

State g00011 has been posted (it is active).
This state contains action g00012, which is a "display" action.
This state expects an indirect object for action g00012, which will be the
    human recipient of the action, and defaults to "ref" (the user),
or a human recipient contained in a prepositional phrase marked by "to".
This state expects a direct object for action g00012, which will be the
    object acted on by the action.
This state expects a prepositional phrase marked by "for", which will be the
    human benefactor (who the action is done for), default "ref".
This state represents a verb part-of-speech.
This state also represents a verb phrase part-of-speech.
This state represents the first word of the input.
This state is backlinked to state i00010 (an input word).

Figure 3-2: Actual representation of state g00011 in working memory

Since no rule can fire on "xn", the rule for the fixed noun phrase "axle order" now fires, creating a description of the database item corresponding to the phrase, and noting that it is an incomplete noun phrase on the interval [4:5]. Incomplete noun phrases are those which have not yet had their syntactic boundaries confirmed, and so should not initiate a search for a referent. The rule for "axle" then fires similarly, followed by "order" and the punctuation (the carriage return is currently regarded as an end-of-sentence punctuation mark).

Since the incomplete noun phrase recorded for "order" now has a punctuation mark at one end, and a word that might not be part of the same noun phrase at the other end ("axle"), a syntactically complete noun phrase for it is created on [5:5]. This allows the referent finding rules to act on it. Currently, these are very primitive, and merely create a description of the set of all database objects which are instances of this type. Such a description is therefore created on [5:5].

Due to the misspelling of "an", no other linguistic rules can fire. The rule which would take "order" as a second object of "show", for instance, cannot fire because no state for "order" is on an interval adjacent to a state for the verb phrase containing "show". Since there is no state spanning the whole input, the rule for assembling a completed parse cannot fire, and a goal requesting error correction is placed into working memory. This activates a number of rules, including one which attempts to correct the spelling of a word if there is nothing covering its interval in the chart, as is the case with "xn". The corrections "in", "on", and "an" are each placed into the chart as a simple lexical item on [3:3], just as the original word at that location was.

The goal of error correction is now removed, allowing the linguistic rules to act. The first rule to fire

```
(p dict-show
        (goal ↑status active ↑name cp-grow-tree)
        (input ↑word << show name present tell give >> ↑utt nil
                ↑state <s> ↑position <pos> ↑prev <prev> ↑next <next>)
        (speaker ↑userID <speaker>)
        -->
;;; Build a new state.
        (bind <new-state>)
        (make state ↑state <new-state> ↑status posted)
        (make ancestor ↑state <new-state> ↑ancestor <s>)
;;; Build the display concept
        (bind <name>)
        (make POS ↑type verb ↑state <new-state>)
        (make POS ↑type verb-phrase ↑state <new-state>)
        (make word-seq ↑first <pos> ↑last <pos> ↑prev <prev>
                                ↑next <next> ↑state <new-state>)
        (make act ↑action display ↑token <name> ↑state <new-state>)
;;; Start up the display expectations.
        (bind <e2>)(bind <e3>)(bind <e4>)
        (make expect ↑type e-marked ↑con <name> ↑marker for
                ↑state <new-state> ↑token <e2> ↑isa #human
                ↑slot bene ↑default <speaker>)
        (make expect ↑type e-unmarked ↑con <name> ↑token <e3>
                ↑marker 2nd ↑state <new-state> ↑slot object)
        (make expect ↑type e-marked ↑con <name> ↑marker to
                ↑state <new-state> ↑token <e4> ↑isa #human
                ↑slot recip ↑default <speaker>)
        (make expect ↑type e-unmarked ↑con <name> ↑isa #human
                ↑state <new-state> ↑token <e4> ↑marker 1st
                ↑slot recip ↑default <speaker>)
        )
```

RULE dict-show:
IF there is an active goal to parse input
and an input word that is one of: show, name, present, tell, give
and a speaker
THEN obtain a unique name
make a posted state indicator
make a backlink to the input word
obtain another unique name
indicate that this state is a verb part-of-speech
indicate that it is also a verb phrase part-of-speech
indicate the part of the input it represents
indicate that it is a display action
obtain three more unique names
indicate an expectation for a human "benefactor" (defaults to user) marked by "for"
indicate an expectation for an "object" as a direct object
indicate an expectation for a human "recipient" (defaults to user) marked by "to"
or as an indirect object.

Figure 3-3: Actual production rule defining ***show***

defines "on" as a preposition on [3:3]. In this system, prepositions have no semantics except in context. Although "on" cannot be used as part of a larger constituent in the current sentence, it does allow a complete noun phrase to be built for "axle order" on [4:5], since it provides it with a left boundary. A referent finding rule then creates a description of the set of all axle orders, again on [4:5]. The same two rules build a complete noun phrase and a referent on [4:4] for "axle".

Once this takes place, the same rule that defined "on" as a preposition defines "in". This again cannot be used as part of any larger structure[2], and this time nothing happens, since the noun phrases adjoining it are already complete. Finally, "an" is defined as an indefinite determiner on [3:3]. It can combine with either of the incomplete noun phrases starting at 4, namely "axle" or "axle order". This is possible even though these have already both been used to build complete noun phrases; an incorrect decision cannot prevent a correct one from being made later. The first of these two incomplete, indefinitely determined noun phrases to be built is for "axle", on [3:4], which in turn causes a complete noun phrase and a referent to be built on [3:4].

At this point, the two incompatible verb phrases built for "show me" on [1:2] are both adjacent to the referent and noun phrase for "an axle" on [3:4]. This, and the expectation that a second object will occur, allow a verb phrase spanning the interval [1:4] to be built, with its recipient and object cases both filled. Only one new verb phrase is built, because a special mechanism prevents any case in a verb phrase from being filled more than once, and "an axle" can only be the object case, due to the restrictions on the recipient case. The "special mechanism" is simply a rule which adds to any verb phrase a list of all cases which are already filled in any underlying verb phrase. Here we see a local ambiguity being decided, since the larger structure cannot incorporate the incorrect verb phrase. Unfortunately, this verb phrase is a dead end, because it only covers [1:4], and cannot be extended to include "order" at 5. However, the determiner "an" has yet to be combined with "axle order". When this occurs a globally consistent parse is found up to the punctuation mark. The verb phrase built on [1:5] is essentially identical to the one built for [1:4], except that "axle order" is the object instead of "axle".

Finally, because there is a verb phrase extending from the beginning of the utterance to an end-of-sentence punctuation mark, a rule fires which builds a request for the described action, and indicates that it is an imperative sentence spanning the whole utterance [1:6]. The system currently only handles surface imperatives, but is easily extensible to other types of sentences and sentence fragments, such as noun phrases. In order to capture the completed parse, the "ancestor" elements of states that are part of the correct interpretation are traced back, their states are marked as being part of this utterance, and the result is assembled into a compact form for further processing. Before the next utterance is parsed, the states not included in the correct parse will be deleted from working memory.

An annotated listing of a short user interaction with this parser is shown in figure 3-4. The first user request is the one discussed above. The second illustrates a slightly different verb form, and the last demonstrates a somewhat more complex noun phrase.

4. Strengths of the Approach

There are a number of ways in which Psli3 is superior to earlier versions of this parser. Most significantly, the chart parsing strategy is tolerant of ambiguity in exactly the way desired. The individual rules need not know whether there are other interpretations for the constituents they use. Because ambiguity is handled globally by the chart mechanism, the rules do not need to handle ambiguity internally, and they are thus much simpler, and easier to write and understand than typical expectation-based parser rules. Because they do not keep track of the ancestry of a constituent, but only know that it is a possible alternative on a particular interval of the utterance, they are simpler and

---

[2] As the coverage increases, the constituent "in axle order" may actually be built (even if it is syntactically imperfect), but it could not lead to a globally consistent parse, and would not interfere with the correct interpretation.

```
Script started on Thu Apr 28 10:51:05 1983
Warning: no access to tty; thus no job control in this shell...
% isis

ISIS, from Franz Lisp OPUS 36 12-21-81.
        [Diagnostic output editted out.]
1.load psli3.0.init
        [Diagnostic output editted out.]
3.run

+show me xn axle order
        ["blade" has been changed to "axle" throughout, for clarity.]

Ambiguous spelling mistake: xn

Attempting (ambiguous) spelling correction: xn
Words tried: an in on

Ready to assemble a meaning for state g00177
                        [This indicates that a single overall meaning was found.]
                axle-order-BAE95036-T1-0012R41W
                        .       [The system currently displays all
                        .         Xs if you ask for "an X".]
                axle-order-907482-09-5092000W

+Display the axles to Mark for me
        [The system understands this, but cannot really redirect output.]
Ready to assemble a meaning for state g00218

                #731J735001
                    .
                    .
                #752J351005

+list the finish-dates of the axle orders
        [Here the user is asking for an attribute of an object.]
Ready to assemble a meaning for state g00395

                axle-order-BAE95036-T1-0012R41W         031282
                        .       [This display routine broke recently.]
                        .
                axle-order-907482-09-5092000W           032682

+bye

Ready to assemble a meaning for state g00409

end -- explicit halt
 97 productions (853 // 1958 nodes)
 669 firings (1722 rhs actions)
 355 mean working memory size (551 maximum)
 24 mean conflict set size (125 maximum)
 586 mean token memory size (1263 maximum)
    ,   [Diagnostic output editted out.]
script done on Thu Apr 28 11:03:26 1983
```

Figure 3-4: A sample Psli3/user interaction

faster than the rules in Psli2. The fact that states are not replicated needlessly reduces the number of working memory elements that need to be matched against, again promoting ease of understanding and acceptable efficiency.

In the initial, small system typical sentences are parsed in 30 to 45 seconds in FranzLisp OPS5 on a lightly loaded Vax Unix™ system. If run under the current Bliss implementation of OPS5, this would be cut by a factor of five, making it almost real-time. Using OPS83, a new Pascal implementation of OPS5, would yield another factor of three, giving parses in a second or so. Finally, work is under way on a special-purpose architecture production system machine, which would add another factor of 100. It should be realized that these times include all processing of the input, including disambiguation and interpretation, not just the time required to produce one of possibly many syntactic structures. Since the time complexity of OPS5 production systems tends to be proportional to the logarithm of the number of rules, adding a large number of rules to this system should not cause significant speed problems.

This system also has a number of advantages over other rule-based approaches to parsing. The use of semantics within the rules which build constituents reduces the number of states built which cannot lead to a complete interpretation, compared to a system which uses a separate syntactic phase. Only those constituents which are at least locally plausible are built. In addition, Psli3 works in a much simpler fashion than other "wait-and-see" systems—notably those of Marcus [8] and Ginsparg [5], which wait as long as possible, and CA [1], which waits as long as it must. Psli3 can be thought of as tentatively making a decision when it uses a constituent in a larger structure, but not finalizing the decision until its global consistency is confirmed after the parse.

The primary advantage derived from the choice of a production system architecture is that rules which can act whenever a certain pattern occurs in working memory greatly facilitate the integration of various types of knowledge, as has been demonstrated in expert system architectures. In fact, OPS5 has been used to construct various expert systems [9] [10]. The integration of non-linguistic knowledge sources has begun with the factory database system, written in SRL [14]. No special dictionary is used to interpret the names of any database items mentioned. When an unknown word occurs in the input (such as "xn"), it is looked up in the database. If it is found, it is treated in the same way during parsing as a noun that is defined as a word. Examples of other rules that can use database knowledge are the set/instance noun phrase, such as "order a-o-31212", or the attribute/object noun phrase, such as "the due date of a-o-41511". Rules of this nature were present in Psli2, and will soon be added to Psli3.

5. Plans for Further Development

In addition to extending the linguistic coverage of the system, a general scheme is being investigated which should allow the system to handle pragmatics (the influence of general knowledge and reasoning on the understanding of a sentence) in a clean fashion. The basic requirement for this is that the system be able to verify the reasonableness of a construct before building it. The natural way to do this in a production system is to build a goal to verify whether the construct in question makes sense. If it does, other rules, outside of the parser, will indicate that the goal succeeded. If there is a problem, the outside rules will indicate a failure. Since any knowledge that the system has will be available for use by the production rules, this should allow the entire knowledge of the system to be applied to the understanding of natural language inputs, without any adaptation of the non-linguistic knowledge.

Several dialogue effects should prove easy to incorporate as well. A mechanism for analyzing a user's goals will be added, as well as anaphoric reference rules similar to Sidner's [11]. In a more unusual vein, it should be relatively easy to incorporate dialogue expectations into this parser's flexible structure, so that dialogue context can influence judgements about how to interpret a fragmentary constituent (e.g., as an answer to a question, or as an expected ellipsis). Also, it may be possible to handle a truly ambiguous sentence by leaving each global possibility in the chart, and allowing the dialogue context to disambiguate the sentence, in effect extending the notion of a chart to cover the structure of a conversation.

References

1. Birnbaum, L. and Selfridge, M. Conceptual Analysis of Natural Language. In *Inside Computer Understanding*, Schank, R. C. and Riesbeck, C. K., Eds., Lawrence Erlbaum Associates, 1981, ch. 13.

2. Durham, I., Lamb, A. L., and Saxe, J. B. "Spelling Correction in User Interfaces." *CACM 26*, 10 (October 1983), 764-773.

3. Forgy, C. L. OPS5 User's Manual. Tech. Rept. CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University, July, 1981.

4. Fox, M. S. The Intelligent Management System: An Overview. Tech. Rept. CMU-RI-81-4, Robotics Institute, Carnegie-Mellon University, July, 1981.

5. Ginsparg, J. Natural Language Processing in an Automatic Programming Domain. Tech. Rept. AIM-316, Stanford AI Laboratory, June, 1978.

6. Jardine, T. and Shebs, S. Knowledge Representation in Automated Mission Planning. To appear in AIAA Computers in Aerospace

7. Kay, M. The Mind System. In *Natural Language Processing*, Rustin, R., Ed.,Algorithmics Press, 1973.

8. Marcus, M.. *A Theory of Syntactic Recognition for Natural Language.* MIT Press, 1980.

9. McDermott, J. "R1: A Rule-Based Configurer of Computer Systems." *Artificial Intelligence 19*, 1 (September 1982).

10. McDermott, J. XSEL: A computer sales person's assistant. In *Machine Intelligence 10*, Hayes, J. E., Michie, D., Pao, Y-H, Eds., John Wiley & Sons, 1982.

11. Sidner, C. L. "Focusing for Interpretation of Pronouns." *American Journal of Computational Linguistics 7*, 4 (October-December 1981).

12. Wilensky, R. and Arens, Y. PHRAN: A Knowledge-based Approach to Natural Language Analysis. Tech. Rept. UCB/ERL M80/34, University of California, Berkeley, 1980.

13. Winograd, T.. *Language as a Cognitive Process.* Volume 1: *Syntax.* Addison-Wesley, 1983.

14. Wright, J.M., and Fox, M.S. *SRL User's Manual.* Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, 1982.