Syntax and Semantics in Natural Language Parsers

Robert E. Frederking

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

May 23, 1985

Abstract

The relationship between syntax and semantics is discussed, as an approach to thinking about different types of natural language parsers. Several types are then reviewed, and difficult natural language phenomena are discussed. The Psli3 semantic chart parser is presented as a possible solution to the problem of syntax, and its theoretical capabilities are discussed. An example of the current capabilities of the parser is presented.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-84-K-1520.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

Table of Contents

1. What is syntax anyway?

In recent years, the question of what role syntax should have in natural language understanding has been quite controversial. There have been arguments for using syntax first, for integrating it with other knowledge, for using it only when absolutely necessary, and even for not worrying about it at all, at least in detail, as a way of achieving robustness. This paper is the result of an attempt to sort out the different views, and will compare the ways different parsers use different knowledge sources, and present a new approach (or at least a new combination of approaches), which integrates syntactic and semantic knowledge in a tractable way.

But first we will try to determine what syntax really is. Traditionally, it has been defined either as the structure of a sentence, with semantics being the actual content, or else as the "parts of speech" and the rules which act on them, determining whether a sentence is grammatical or ungrammatical. The idea of structure versus content seems strongly related to formal logic, where the semantics of certain logical operators and their syntactic arrangement are defined independently from any real-world knowledge (except of course for the knowledge of the way logic works). All real-world knowledge is carried by various functions, predicates, variables, and constants, which are defined in terms of a model of the world one is talking about. Rather than argue about the usefulness of formal logic, or whether perhaps this sort of syntax is only the semantics of a set of patterns, it should be sufficient to note that, when used in a computer program, all semantics must be represented by structures as well, so the division between structure and content becomes less meaningful[1]. One could try to differentiate surface syntactic structures from deeper semantic structures, but then problems arise in differentiating deep syntactic structures from semantic structures.

Following the grammaticality approach, the concept of a sentence being ungrammatical seems to be intuitively clear, but also seems to be hard to define formally. A common definition is that syntax is the set of rules that determine which orderings of words are allowed in a given language, plus which ways the morphemes of the language can be combined into words[2]. Unfortunately, this does not seem to capture what people really mean by syntax, since this definition could require the use of any of our real-world knowledge in its application. For instance, by rigidly applying this definition, Chomsky's classic example of a syntactically correct but meaningless sentence, ***Colorless green ideas sleep furiously***, would be ruled "syntactically" incorrect, since no one would claim that this sequence of words is allowed in English. A related definition, that it is the existence of a small number of parts of speech which are abstracted from the words of a language, and can be used to determine which combinations of words are legal without referring to the rest of one's knowledge about the words, will be discussed in a moment, after our intuitions about syntax are looked into.

As it is intuitively used, syntax seems to mean that portion of a language user's knowledge that indicates how to combine the meanings of the morphemes, words, and phrases he knows using positional information to get the meanings of larger units: words, phrases, and sentences, respectively. The requirements of syntactic agreement, such as between the number of the subject and of the verb in an English sentence, are part of the conditions that must be satisfied for these rules to apply. (Understanding syntactically incorrect sentences will be discussed shortly.) Thus, each word in the previous example is a word of English, and they can be combined by using the rules of English for combining meanings. The problem is that the meaning produced is internally inconsistent, and would be rejected in favor of a different interpretation, if one were available.

One way to approach the concept of syntax, and yet remain intellectually independent from traditional notions, is to think of it as that part of our linguistic knowledge which is used in understanding new words and

---

[1] In fact, [Woods 75] argues that semantic representations should not be called that, since they deal only in structures, rather than in "real" semantics, that is, the relationship between symbolic structures and what they denote.

[2] For a similar definition, see [Winograd 83], p. 35.

novel metaphors. So if one heard ***Your green ideas will get us killed***, or ***That is a very colorless idea***, or ***Last night I slept furiously***, one could get a good idea of what the sentence meant. Chomsky's example simply has so many anomalies in it that it is hard to derive any consistent interpretation from it. Similarly, if one heard ***Joe frobbed the foo with a bar***, one would know that ***foo*** was a noun, and ***frob*** a verb. This could explain where our intuitive notions of syntax rules come from, without requiring us to posit a separate syntactic process. Normal processing could proceed in an integrated fashion, either with syntax integrated into the individual rules, or with interleaved execution of syntactic and semantic rules, while purely syntactic processing would be used when no current rules containing non-syntactic knowledge apply, to derive a meaning for a new word or phrase. Similarly, phrases with syntax errors which are still understandable to people could be interpreted by applying purely non-syntactic processing, and correcting the grammar of the original internally once a reasonable meaning has been found. (This explicit corrective action is necessary because grammar errors can provide useful information on social, educational, and regional differences.)

Looked at in this way, syntax *is* somewhat different from semantics and real-world knowledge, but may not be categorically different. There may not be any sharp dividing line between these two different types of knowledge when they are used by humans in understanding a sentence. The rules and word classes, or "parts of speech", that seem to generalize across many examples, without relying (too heavily) on the meanings of the words used[3] or requiring detailed hypotheses about the mental processes involved in understanding, are seen *by us* (linguists, AI researchers, and other humans using introspection) as syntax. The parts of speech which are sometimes used as evidence for syntax are thus simply the sets of words that are manipulated in similar ways by those rules which do not rely too heavily on the meanings of words, and can not in fact be as cleanly defined as one is lead to believe in grammar school. In other words, in this view there are just rules, some of which are more specialized; some more generalized; some more syntactic, relying more on word classes and structures relating only to language; and some more semantic, relying more on the real-world properties of the things referred to. The division between syntax and the rest is in the eye of the beholder.

As an example of the way syntax and meaning can run together, let us consider the differences between verbs and nouns. Word classes are usually defined either by the distribution of their member words in sentences, by their morphology, or by their meaning. The first two methods are syntactic, and appear to be straightforward, although there are real difficulties in using them objectively in a systematic way. The distribution method defines word classes by putting two words in the same class if they occur in the same contexts. But the distribution of any one word will generally be different from any one other word, unless one starts by making some decisions about when these differences are "really semantic" and when they are "really syntactic". Also, many words must be assigned to several word classes, while others do not really fit into any general word class, and are traditionally lumped together as "adverbs". As for morphology, while this seems to cleanly divide verbs and nouns, most word classes in English are not clearly marked, and even verbs present problems, since some of the forms they take on act like nouns: ***Running is good for you.*** (Semantically oriented parsers can avoid problems like these in many cases by not making a syntactic decision at all, whereas syntactic ones must always make a decision as to whether it is a noun or a verb.) On the other hand, a purely semantic approach which attempts to derive word classes from the semantic properties of the objects they refer to cannot deal with the fact that in many cases the class memberships of particular words seem to be fairly arbitrary: ***return*** is both a noun and a verb, whereas ***go*** is only a verb (in its root form)[4]. The correct way to deal with this issue seems to be a combination of the two approaches, acknowledging that many concepts can be expressed either as a noun or a verb, but also noting that the form used is not actually an arbitrary choice—the choice carries meaning. Expressing a concept as the head noun in a noun phrase *refers*

---

[3] I.e., the interactions of the real-world properties of the objects the words refer to.

[4] Example taken from [Winograd 83], p. 51.

to it as an object ***(the marriage of John and Sue last Sunday)***, whereas expressing it as the main verb in a clause *describes it as an event or state in the train of world events*, and uses it to convey a message of some kind ***(John and Sue were married last Sunday)***. This message may be a question, an assertion, a request, etc., but in any case says something using the concept in question as the main concept that the message is built around. It is interesting to note in this regard that a verb phrase has a required tense and modality, while a noun phrase does not. As a message, the verb phrase must indicate whether you are asserting the event or state, and the time period covered by it. Also worth noting is that although you can include new information in a noun phrase ***(the former marriage of John and Sue)***, the complete phrase still refers to the object or concept named, and is not in isolation a complete utterance (i.e., a message).

This account of syntax would explain the extreme difficulty linguists have had in defining word classes and rules that successfully capture the generalizations of English[5]. They are attempting to find general rules which simply are not present in the actual natural language understanding systems they are studying. The apparent generalizations found are true for large subsets of the language, but almost always exceptions exist, which are often simply special cases that defy inclusion in general rules. Some of these apparent generalizations may be due to the way our learning processes form rules[6], while others may be due to historical accidents, such as the importation of words with Latin morphology, or the fact that the allowable ways to form new words have changed over the centuries, but the words formed by the old rules have remained in use.

According to this view, it would not be unreasonable to use syntax explicitly in a natural language understanding program. What would be considered unreasonable is the use of syntax as a control scheme. Most parsers which explicitly deal with syntax use the syntactic structure of the sentence to drive the parse, building a syntactic tree top-down, bottom-up, or using some other strategy. Generally, semantics is used only after a syntactic parse is completed, to check whether the proposed structure has a reasonable interpretation, or are limited to simple checking of semantic types during the parse. Since the boundary between syntax and the other knowledge sources used seems to be indistinct in nature, the use of syntax as a separate structure, radically different from semantics of various kinds, might be expected to produce a boundary effect, similar to those in any discrete representation of a continuous phenomenon. In fact, such effects do seem to occur, in that phenomena that are naturally more "syntactic" become easy to handle, while those which are naturally more "semantic" become quite difficult to incorporate. Some examples in English include the attachment of prepositional phrases to noun phrases, and the understanding of noun-noun compounds, both of which are ambiguous with regard to syntax, and require semantic judgements for their resolution.

There seem to be two main arguments in favor of using a separate syntactically-controlled phase in parsing. The first of these is that syntactic parsing is much more efficient than semantic parsing, since the simple uniform algorithms that syntactic parsing makes possible can be made to run very quickly, producing a parse of a sentence in milliseconds. Semantics can then be used at the end, to verify that the parse found can be interpreted. Unfortunately, this comparison leaves out the amount of time that is spent by the syntactic parser in interpreting the parse, which can be much larger than the comparable portion of the semantic parser's time, since postponing all interpretation until after all syntactic parsing is finished can result in many spurious possible parses, each of which must be considered, and many of which would have been filtered out much earlier in the parsing process if the parser had used semantics in an integrated fashion. This is an example of

---

[5] As noted in [Charniak 76], p. 23.

[6] See [Anderson 83] for a detailed psychological theory in which these generalizations are the result of specialized rules being created by "compiling" sets of more general ones via a general learning mechanism.

the general rule in artificial intelligence search problems which says that moving some of a program's testing into its generator almost always increases efficiency.

The other major argument is that the rules of syntax are more general, and therefore more domain-independent, than semantic knowledge, and that therefore a syntactic parser is much more portable between different domains. While this is true to a degree, there are in fact variations between different domains in terms of the portions of English syntax that they generally use, so until a parser actually covers all of English, it will need to be augmented when it is moved to a new domain. This argument also ignores the fact that the underlying semantics of the new domain will have to be developed in any event for the program which will make use of the parses produced by the parser. If a semantic parser can take advantage of the semantic routines that the application program will need anyway, the difficulty of moving it to a new domain may not be much greater than that for a syntactic parser. While this does not appear to be possible for such things as semantic grammars (discussed below), the project described in the final two sections has this as one of its long term goals.

2. Different kinds of parsers

As mentioned above, one dimension on which parsers differ is the extent to which they use syntactic knowledge, and the form which this knowledge takes. Other structural dimensions on which parsers may differ include:

- ***Modularity*** — the degree of internal organization. While systematic internal structure is desirable in any program for software engineering reasons, in parsing programs it has often had the unfortunate side-effect of causing them to become inflexible. It becomes very difficult to make the parser change with its dialogue (or other) context. This appears to be due to the integrated nature of natural language processing, as noted above.

- ***Control strategy*** — what kind of control strategy they use in attempting to build hierarchical structures which match their input. The two most common strategies in explicitly syntactic parsers are top-down, where the parser starts with the top-level structure it wants to find and fills in successive parts until it has accounted for all the words in the input, and bottom-up, where it starts with the actual words and builds up constituents until it finds a single structure that accounts for the whole input. Most semantically oriented parsers use a mixed strategy, using bottom-up methods to choose a high-level structure, and then using that structure in a top-down fashion on the rest of the input. This sort of hypothesize-and-test strategy, which is common to many types of frame-based reasoning, has been called "expectation-based parsing" by Schank [Schank 80], since reading the beginning of a sentence sets up expectations as to what will be seen in the rest of the input. (The typical implementation of this concept will be described shortly.)

- ***Handling ambiguity*** — when and how they make decisions between conflicting choices. The various possible ways to handle these decisions include picking one and backtracking on error, picking one and doing "intelligent error correction" [Birnbaum 81], postponing decisions as long as possible (referred to also as "wait and see"),[7] and maintaining the set of all currently plausible options in parallel.

Six parsers will be described along these dimensions:

---

[7] Also from [Birnbaum 81], quoting Marcus.

- the LUNAR ATN [Woods 72]
- chart parsers as described in [Winograd 83]
- the LIFER semantic grammar approach [Hendrix 77]
- Wilks' preference semantics parser [Charniak 76]
- the expectation-based conceptual analyzer CA [Birnbaum 81]
- the IPP program [Schank 80]

The first two of these are highly syntactic. In the ATN (augmented transition network) approach, the grammar is specified as a list of states, each with an ordered list of transitions to other states. The transitions can depend on the semantic type properties of the current word, and can set and use the results of a set of registers (the augmentation that gives this approach its name). States can be pushed and popped, giving the overall system Turing machine power. In chart parsers, the grammar is specified as a set of syntactic productions, applied by a procedure which only uses the syntactic categories mentioned in the grammar. Semantic testing, if done at all, is used at the end to select among the syntactic parses generated, when there is more than one. The semantic grammar approach blends the two kinds of knowledge more evenly, using a grammar based on domain-dependent categories which embody semantic as well as syntactic knowledge, specified by rewrite rules which use LISP action side-effects to build internal structure. The last three (Wilks', CA, and IPP) all take a dim view of syntax, utilizing it as little as possible, and not building any syntactic structures at all, but rather building a semantic representation immediately from the surface string. They still use syntax in some sense, since they use word order information and syntactic case markers such as prepositions. In CA and IPP, the use of word order information is generally implicit, in that one word will set up structures and expectations which affect how later words are processed.

In terms of system structure, the highly syntactic ones are also highly structured, since they use the structure provided by syntax as the representation for the linguistic knowledge they have. The simplicity of this approach is one of its chief practical attractions, aside from any questions of theoretical correctness. Any detailed semantic processing in either ATNs or chart parsers is relegated to unspecified later processing. Semantic grammars are also strongly structured, since they follow essentially the same scheme, except that they include domain-specific semantic knowledge in the categories and rules they use. Of the non-syntactic parsers, the most cleanly structured is Wilks', in that he uses various "formulas" and "paraplates" to represent the linguistic knowledge in his system, and has a uniform matching procedure which applies them to the input[8]. Both CA and IPP use frames with packets of LISP code embedded in them for their word definitions. These packets of code are called "requests", and they serve to encode and proceduralize the expectations that reading these words generates. This use of unrestricted code in each word definition leads to the kind of extremely complex interactions typical of unstructured systems, and the programmer must know a great deal about the definitions of previous words in order to successfully add a new word. This is often justified by claims that it is the result of the inherent complexity of the task. These two systems differ mainly in the level of representation used. CA builds a detailed semantic representation of its input using Conceptual Dependency notation [Schank 75], while IPP tries to get at the meat of the story it reads directly, building high level knowledge structures [Schank 77] as its representation, and not even attending to all the words in

[8]Unfortunately the value of this clean structure is diminished somewhat by the impenetrability of the representation used.

the input. In some sense, it is almost not a parser, but more a program for loosely interpreting a whole story.[9]

A somewhat separate issue from the amount of structure in a system is the control strategy it uses in analyzing an input and building its internal representation. ATNs and semantic grammars both use a top-down approach, starting with a set of top-level structures that represent the allowable kinds of sentences, and, for each of them, trying to find each of its subparts in turn. The problem with this technique is that the parser will generally try to build many structures which do not actually exist in the sentence. There are ways of reducing this inefficiency, but it cannot be completely prevented. Pure bottom-up techniques are not used in any of the parsers in question, possibly due to the large number of false starts produced when a word is taken in isolation as possibly being part of any constituent in the grammar, and all of the ones it fits are built. The distinguishing feature of chart parsers is their control strategy, which involves the use of a *chart*, or well-formed substring table. This holds every constituent which has been found so far in the current input, and can be used in conjunction with top-down, bottom-up, or some other strategy. The presence of constituents in a table prevents any constituent from being built more than once, since every rule that would build a new structure checks the chart for it first. This mechanism greatly increases efficiency, combining the selectivity of top-down and bottom-up processing, in much the same way as dynamic programming techniques increase the efficiency of recursive algorithms. As for the non-syntactic parsers studied, all three use the expectation-based hypothesize-and-test paradigm mentioned earlier. The implementations of this scheme vary, however, with Wilks instantiating a declarative frame and then using a uniform procedure to try to fill it in, whereas CA and IPP activate procedural rules attached to the frame chosen, which then carry out the actions necessary to fill in the frame. CA and IPP differ, again, in the size of frame used, CA using primitive CD ACTs, and IPP working with scripts and scenes.

The final dimension on which these systems will be compared is the question of when decisions are made. The problem this addresses is that natural languages contain many local ambiguities, so that decision points arise where the parser must decide which path to follow, i.e., which local parse to use, or else maintain several parses in parallel. The possibilities used include:

- backtracking on error
- intelligent error correction
- only deciding when you have to
- maintaining all options in parallel
- avoiding the issue

ATNs and semantic grammars pick one path, either based on a programmer-supplied ordering of paths or nondeterministically. If the chosen path fails, they backtrack and try other possibilities. To avoid this highly inefficient process, a chart parser uses its chart to maintain all the options in parallel without incurring too much overhead. It thus does not actually need to make a choice unless the entire sentence is ambiguous, in which case any parser must make a decision, either taking the first result, returning all of them, or rating them somehow and taking the best. In Wilks' system, the different possibilities are generated by a simple match of "bare templates", and then the preferences each object has for its subparts are used to pick between them, producing a single reading for each phrase. This would seem to make it an early decision system, with no mention of any backtrack or error repair mechanisms. Since all of a word's semantics are used in this

[9] It is called, after all, the "Integrated Partial Parser".

decision, presumably most ambiguities are sorted out correctly. It is interesting to note, however, that, as presented by Wilks, this decision is made before his "paraplates" are applied to find the deep case relations between templates. Thus if there are ambiguities present which depend on preferences for objects in a different template for their resolution, Wilks' system may not get them right. While there is a discussion in the CA paper on the relative merits of wait-and-see versus early decisions combined with intelligent error correction, this choice is left up to the programmer, depending on how he writes his rules. The built-in decision mechanisms include examples of both strategies: a word sense disambiguation scheme which keeps requests for conflicting definitions from firing until one is clearly preferred (wait-and-see), and a mechanism that gives marked cases priority over unmarked ones, so that ***John gave Mary to the Sheik of X***[10] will be correctly parsed, even though ***Mary*** is at first taken to be the recipient (decide and intelligently backtrack). In the case of IPP, decisions are made immediately, since it has very strong expectations as to what the stories it reads are going to say, and would not notice a mistake if it made one. Problems such as multiple word senses simply are not handled[11], as it will ignore words that it is not interested in, while assuming that the words it recognizes have the meanings that it expects them to have.

An interesting correlation should be noted here, in that the various choices made by the designers of these systems seem to depend to a large degree on whether they use an explicit syntactic structure or not. The connection with the level of internal system structure is fairly obvious, since the matching of syntactic structures allows a relatively simple, well-defined overall scheme, while matching semantic structures can be a good deal more complex. Somewhat less obvious is the tendency for the decision to emphasize semantics to lead to a hypothesize-and-test scheme. This can be partially explained by the difficulties one has with either maintaining parallel alternatives or backtracking in a semantic scheme. In addition, there is a need to have some top-down semantic context to work with, since bottom-up generation of all possible semantic constituents is clearly not feasible. Finally, the designers of these systems usually want them to work in an intuitively appealing fashion. These factors also seem to be the reason that Wilks' and IPP make final decisions fairly early in the process, and that CA proposes either wait-and-see or intelligent error correction or both, implemented partly as general schemes, and partly as special-case LISP code.

3. Why natural language understanding is hard

Now that we have discussed the relationship between syntax and semantics, and looked at a number of parsers that take different approaches to some of the problems of understanding natural language, we are ready to discuss which phenomena in English seem to cause the most difficulties. Some of these cause more trouble for syntactic parsers, whereas others are harder for non-syntactic parsers. Some are simply not well understood, and pose difficulties in any formalism. It should be mentioned here that all of these methods are equivalent in formal power, which is to say, any of them can in principle be made to handle any situation that any of the others can. The problem with this notion is that each method is good at handling certain things, and very bad at handling others. Although it is possible to get any of these methods to do anything, some constructs are simply not feasible for some parsers.

The phenomena that seem to be the hardest for ***syntactic*** parsers to cope with include:

- word-sense ambiguity
- prepositional phrase attachment

---

[10] Example taken from [Birnbaum 81] p.336, quoting Wilks.

[11] It is claimed that this does not cause any problems in the domain of skimming newspaper stories.

- ill-formed input
- integration with other processing

The problem with ambiguity is that most words in English have more than one meaning, and it is often not possible to resolve this ambiguity without understanding the meaning of the words in the sentence. This may not cause a problem for the parser itself, if all the senses are in the same word class, since it can leave the disambiguation up to the interpreter. Unfortunately, these different senses are often in different word classes, forcing the parser to make a decision or generate several parses, one using each sense. The problems that arise with prepositional phrase attachment and other types of *structural* ambiguity are similar, in that there are generally many places in the syntactic structure that a prepositional phrase would fit, and the only way to tell which is correct is by the semantic plausibility of the attachment.

Attempts have been made to remedy this situation through the addition of semantic restrictions to syntactic parsers. As mentioned earlier in connection with ATNs, these work by allowing only those word senses which have the correct semantic type to be combined with words that have semantic restrictions on their connections, such as verbs. This has proven to be inadequate, since not all semantic knowledge can be encapsulated in this way, and any piece of knowledge may be necessary for a disambiguation. An additional approach for handling prepositional phrases involves incorporating case frame information as a test on prepositional attachment, so that only those cases a verb allows can be attached to it.

Handling ill-formed input and coping with dialogue, script, and other context effects are difficult because of the highly structured framework that syntactic parsers use. It is not clear how to incorporate other mechanisms into a syntactic parsing mechanism, since it uses a simple procedure that only looks at the syntactic class of a word. Such things might be added as separate routines, called either when an error is encountered or at the end of a parse, but appear to have had very limited success. Examples of this kind of problem are ungrammaticalities, spelling errors, pronoun interpretation, and dialogue expectations, such as expecting a noun phrase as an answer after a question.

Semantic parsers on the other hand have the most difficulty with structures that depend on fine grain syntactic analysis, and those that just seem to be naturally syntactic. An example of this type of syntactic detail is given in [Winograd 83], p. 51:

> ***The man who knew him was going left*** and ***The man who knew he was going left*** are both meaningful and contain the same words, differing only in a subtle syntactic marker. But one is interpreted as ***the man who was acquainted with him was going leftward*** and the other as ***the man who knew that he was going went out.*** If the message is to be deciphered, the complexities of the syntactic structure need to be accounted for.

Of course, there is no inherent reason why such things could not be addressed within this framework, but by and large they are not. The reason for this seems to be largely idealogical, in that the proponents of non-syntactic parsing wish to do absolutely as much as possible without explicit syntax as they can. Examples of constructs that have been claimed to be inherently syntactic include raised and passive sentences, where the surface subject can be almost any noun phrase which would have followed the verb, and non-subject relative clauses, where the constituent the clause describes is left out of its normal place in the clause.

Several constructs are just hard, and are not handled well by any current parser. These include anaphoric references, such as pronouns with antecedents in previous utterances; extraposed constituents, as in ***The man came in wearing a red hat***; and various combinations of conjunction, comparatives, and ellipsis, which often occur together: ***John took the red ball, and Mary the green***; ***John is taller than Mary (is tall)***; and ***John is taller, and Mary shorter, than Ruth.*** Ellipsis also occurs in dialogue settings: ***Show me all my files. Now my***

directories. Oh, I meant print out. While no current parsers can handle these constructs in general, those which do not rely solely on syntax would seem to be the better candidates for eventually handling them, since many of the clues as to which structure is correct seem to come from a thorough knowledge of the semantics of the situation described.

4. A different approach

The most important lesson from the preceding discussion seems to be that explicitly using syntax really is worthwhile, but that using it as a simple, clean control structure might not be as helpful, due to its rigidity. Another way to look at this is that one should try to incorporate as much knowledge as one can, both syntactic and semantic, in an integrated fashion, since any of it may be necessary to understand a sentence. All previous attempts to leave some knowledge out of the interpretation process have failed, due to examples that just could not be handled without the missing knowledge. At the same time, some structure is necessary for human engineering reasons, if the parser is to become large enough to cover a reasonably large subset of English. In this section, an attempt to develop a parser which can take advantage of the best features of the different kinds of systems will be discussed. These features include the chart parser's parallel control and decision strategy, the semantic grammar's mixture of semantic and syntactic information, Wilks' use of a mostly declarative representation for linguistic knowledge, and an adaptation of CA's and IPP's use of expectations.

The parser is called Psli3 (pronounced "sly three"), and is implemented as a part of an OPS5 [Forgy 81] production system natural language interface to the Intelligent Management System [Fox 81] being built in the Intelligent Systems Laboratory of the Robotics Institute at Carnegie-Mellon University. A detailed description of Psli3 parsing a sample sentence is given in [Frederking 84], included here as an appendix. The use of a production system provides the structure needed to make extension to a large system feasible, without predetermining either the control strategy or the amount of explicit syntax present. OPS5 productions are composed of a condition side, which consists of patterns matching declarative objects built from working memory elements, and an action side, which involves primarily the placing of objects into working memory. Therefore the rules are much more declarative in nature than LISP code. So declarative, in fact, that there are plans to eventually produce the productions automatically from SRL[12] schemata describing the phrases the system knows. Another advantage of using a production system is that all of the semantic representations in the system are available to the rules' condition sides in working memory, facilitating the inclusion of any semantics present anywhere in the system as part of a phrase's description. (That the rules used are not simply syntactic, but include semantics, should be no surprise at this point.) This ability, which gives the technique its name, should be helpful in dealing with dialogue phenomena and real-world knowledge.

Within the general structure of OPS5, the chart parsing mechanism is implemented by having each rule check for the starting and ending lexical positions of each constituent state it matches, and having it indicate the starting and ending positions of the state it builds. A state represents an interpretation of a word or phrase, and includes data structures representing both syntactic and semantic facets of the interpretation, separated so that other rules can check only those aspects of the constituent that they are interested in. This is in contrast to semantic grammars, where this information is all convolved into a single, atomic category name. The chart produced for the example sentence from the appendix, ***Show me xn*** [sic] ***axle order***[13], is shown here in figure 4-1. Here, ***xn*** is a misspelling of ***an***, which the system corrects. Each line segment in the diagram

---

[12] SRL [Wright 82] is the semantic network database used by the Intelligent Management System.

[13] 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.

represents a state built by a rule, usually defining a newly found constituent of the sentence. Each state has backlinks to its constituent states, generally used only to assemble the completed parse. For clarity, the only backlinks shown are those for the states which are part of the correct interpretation.

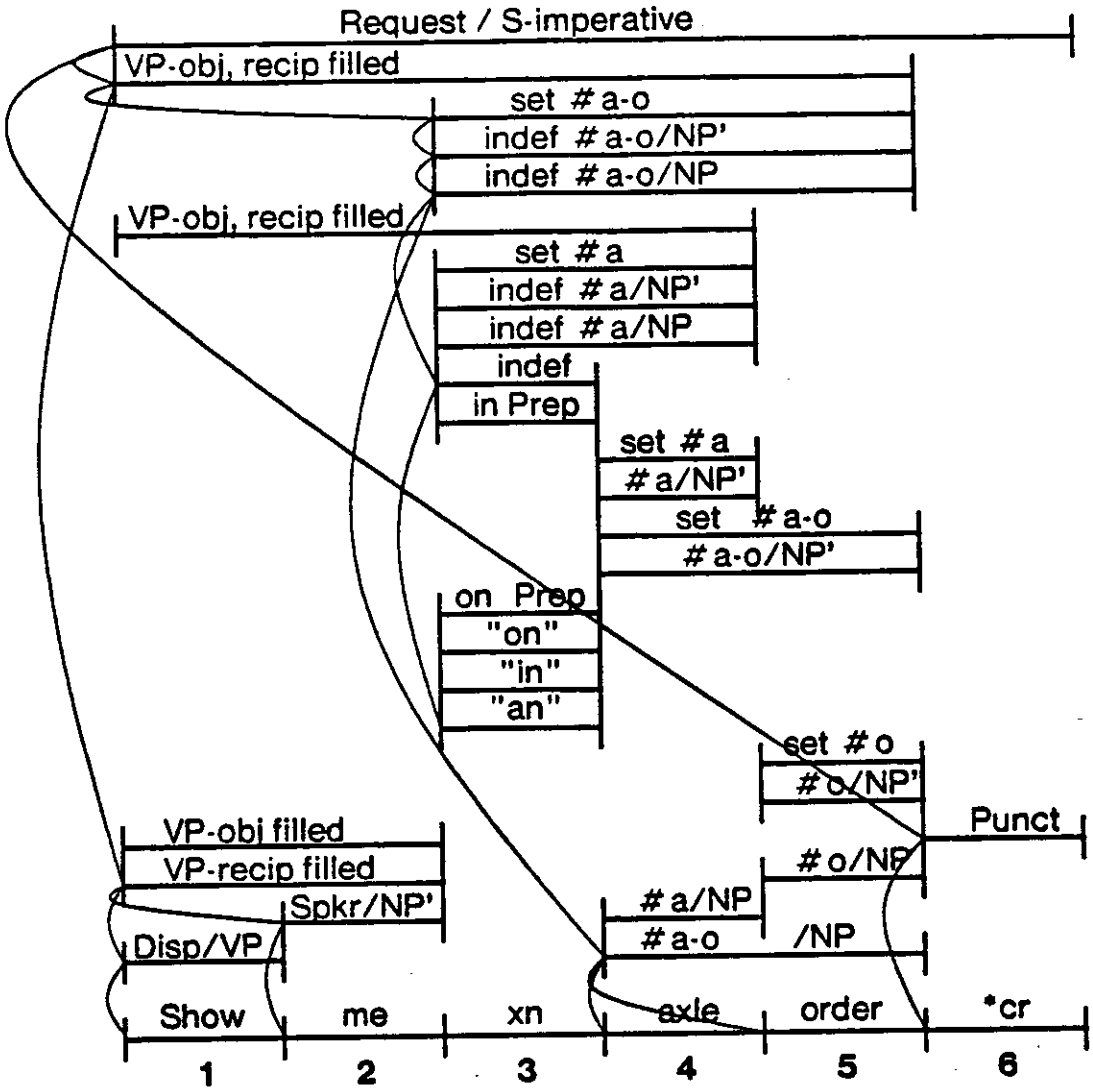

Figure 4-1: Chart built for "Show me xn [sic] axle order".

A locally ambiguous word or phrase, such as ***Show me...***[14], produces several interpretations spanning its part of the sentence, each of which is available to any rule trying to build a constituent incorporating that section, all of which coexist without interfering with each other. This flexibility is due to the use of lexical positions when looking for smaller constituents to incorporate into larger ones. One way to think of this is as indexing the interpretations of a constituent by attaching them to the actual words used in the surface level representation. As mentioned earlier, a chart is used in conjunction with some other control scheme. In this case, the scheme used is mostly of the hypothesize-and-test variety, since expectations are used to control the

[14]This phrase has a case ambiguity, since *me* can be either the recipient, as here, or the object, as in ***Show me to Mark.***

building of the verb phrases in the system, but the expectations are the result of the bottom-up construction of the original constituents. This top-down control is exerted by only allowing the rules to attach noun phrases or prepositional phrases to a verb phrase if declarative "expectations" included in the description of the verb indicate that the phrases fit the cases the verb allows. The use of declarative expectations differs from the CA or IPP approach, in which the expectations are embodied as procedural requests. Declarative expectations are required by the production system philosophy that rules reside permanently in production memory, and are only added by learning processes. Not only does the decision to build a chart provide a solution to the problem of which control strategy to use, but it also handles ambiguity in a simple way, without complicating the individual rules. The mechanism is general enough to include all forms of ambiguity, even those resulting from ambiguous error corrections, such as the spelling correction in the example. The only ambiguity question not addressed so far is how the parser will handle sentences which are globally ambiguous, but which have a clearly preferred meaning. It is currently expected that syntactic and semantic clues, such as parallelism between members of a conjunction, and dialogue conventions will allow the construction of rules which will prefer one interpretation to another. It is also possible that with some care in the design of the rules the OPS5 conflict resolution mechanism will be able to cause the best interpretation to be generated first, so that taking the first complete parse would be sufficient.

The other major feature of the design of Psli3 is the inclusion of rules which define entire phrases as units, rather than always building phrases up from the individual words in them. Despite the widespread tendency to design parsers that always build up meanings from individual words[15], there seem to be important reasons to have a "phrasal lexicon" [Becker 75], in which whole phrases and their meanings are stored. The main reason that parsers do not usually follow this path seems to be that a word-by-word interpretation ability must be present, if the system is to understand novel sentences. What is often ignored is the fact that understanding a phrase which one is familiar with is noticeably easier than understanding a truly novel phrase, and the possibility that this might be a desirable property for computer systems to emulate, for efficiency reasons. In addition, there are many phrases which apparently *must* be understood as a whole, rather than being built up from their constituents. Such phrases include not only obvious idioms, which are not at all interpretable from their parts (***to pull someone's leg***), but even phrases which might be derivable from their parts, but which lack a single, clear interpretation (such as ***fire station*** versus ***gas station***)[16], and more general patterns, as might be found in the rules of a semantic grammar (such as ***Now there's a (noun phrase) if I ever (perceive)-ed one!***).

The inclusion of phrases such as these in Psli3 is very easy, due to the inclusion of semantics in the rules, and the graceful handling of ambiguity. In fact, the inclusion of semantics in the rules by its nature replaces single rules for various syntactic constructs with sets of rules which carry semantic information and constraints with them. (Of course, where a truly syntactic rule seems to be operating, a general syntactic rule can be used.) These semantically oriented rules can be thought of as specializations of the general syntactic rules, where additional meaning has been attached to them as they were learned. Thus the existence of a phrasal lexicon in this system is quite natural, and could be taken as evidence for its psychological plausibility. One can include verb case frames in this scheme, viewing them as verb-specific specializations of the general syntactic rule that a verb phrase contains zero, one, or two unmarked objects, plus prepositional phrases. The verb phrase is an example of a general syntactic rule that is clearly far too general for the real linguistic facts.

---

[15] An exception is the PARRY system, as described by Faught in [Waterman 78], which appears to use a phrasal approach, taking it even a step further, in that whole interactions are viewed as single patterns, although they admit that they break these patterns up into a set of rules "for ease of implementation".

[16] The article in [Waterman 78] by McDonald and Hayes-Roth describes some of the difficulties that can arise in trying to infer the meanings of such phrases.

The existence of general rules that are useful for learning and inference, but too general to account for all the phenomena in their domain, is not restricted to language use. As an example, human reasoning about numbers must be productive in the same way as language, since we can talk about numbers such as 39,423,929, yet there are some numbers such as 7 and 13 which clearly have additional cultural meaning that cannot be derived from their numerical qualities, but must simply be learned. While Psli3 does not learn new phrases currently, it can go into an inference mode, when the rules for words and phrases it knows do not lead it to an interpretation for the complete utterance. This general inference mode, which may in the future be broken up into bounded and unbounded inference, is where such things as spelling correction and database lookup of undefined names occur. It could also include overly general syntactic or semantic rules that could propose possible interpretations of difficult or ill-formed inputs, and other heuristics for handling such things as novel noun-noun compounds. This is similar to a suggestion in [Newell 78] that most human parsing is done in a compiled fashion, with only complex novel constructions pieced together in an interpretive fashion.

5. Theoretical coverage of the Psli approach

As is the case with the other formalisms discussed, the mechanisms in Psli3 can handle any phenomena that any computer program can. However, as noted in section 3, this says very little, since in actuality some things might be hard enough to formulate that they would not actually be handled in practice. In this section, we will examine how Psli3 handles, or will handle, the various constructs that pose problems for the other parsers. We will also take a look at its efficiency compared to earlier versions.

5.1. Current capabilities

First come the constructs that are difficult for syntactic parsers: ambiguity, prepositional phrase attachment, ill-formed input, and integration with dialogue and general real-world knowledge. Ambiguity and some ill-formed inputs are already handled, using the mechanisms discussed above. One interesting point which should be made here is that when chart parsing is applied to grammars that are not context-free, one runs into problems with incorporating the context of a constituent into the indexing scheme. That is, two constituents might look identical locally, but rely on different interpretations of another part of the input. The options available include entering each such constituent into the chart separately, and checking later as to whether the non-local context matches, or somehow including the non-local context in the indexing scheme. In Psli, rules which can be affected by non-local context create a "bindings" working memory element in the states they produce, and will only fire if no state exists on the interval in question with a "bindings" element identical to the one it would produce. This element indicates which rule is operating, and what its variable bindings are, thereby including any non-local context in the index.

Prepositional phrase attachment is currently done using case frame constraints, as described in the previous section. General real-world semantics may eventually be included in the attachment process, by having the appropriate rules create subgoals to evaluate the reasonableness of an attachment before doing it. The subgoal would cause the activation of error-checking rules which presumably already would exist for any kind of request that the system "understands" how to carry out. Another interesting difficulty resolved in the design of the Psli parser was the question of how to incorporate case-frame constraints into the production system/chart parsing environment, in that the marked cases of a verb can occur in any order, but each case is only allowed to occur once per verb phrase. After some thought, the answer appeared to be to represent a whole clause, including the subject, as a verb phrase, with the attachment of each noun phrase or prepositional phrase extending the lexical interval covered by the verb phrase to include the new constituent's lexical interval. This raised a further problem, in that the syntax of OPS5 rules[17] appeared to prevent a single

[17] The infamous "no variable slots allowed" problem.

rule from handling all marked cases if this were done. (In fact, it appeared that a combinatorial number of rules might be needed.) Fortunately, the solution to this proved to be simple. Each state representing a verb phrase has "inhibit" elements in it, indicating which cases have already been filled, and are therefore not allowed to be filled again. A single "inhibition" rule propagates all the "inhibit" elements found in the original verb phrase prior to attachment into the new larger one which includes the newly attached case. This appears to allow a small number of attachment rules to handle the problem.

5.2. Future capabilities

As for future developments, dialogue effects and general real-world knowledge are not yet handled by this system, but there does not appear to be any theoretical difficulty with introducing mechanisms such as the subgoaling scheme described above to deal with such issues. In particular, dialogue rules such as the anaphoric reference rules described in [Sidner 81] should be quite easy to integrate with the Psli approach, since they are already described as condition-action rules.

As for the phenomena that give semantic parsers trouble, fine grain syntax should not be an issue with this approach, since the parser explicitly keeps track of the syntactic aspects of the sentence it is parsing. However, since the coverage of the syntactic forms implemented so far is still quite small, this cannot as yet be demonstrated. Sentences exhibiting syntactic phenomena that are not yet covered will of course cause problems, but this is clearly the case with any parser. As for truly syntactic constructs, these may be fewer in number than sometimes claimed. For instance, raising, as in ***They believed Patty to have robbed the bank***, can probably be handled quite easily by assuming that the raised form is handled by a slightly different set of rules than the unraised form, ***They believed that Patty robbed the bank***, although proof of this will also await the results of further implementation. That the two forms mean the same thing can be handled by having the two derivations produce the same semantic representation. The assumption here would be that any generalization that can be drawn deriving one of these two forms syntactically from the other is the result of external factors, as described in the first section, and is not directly relevant to the design of a parser's actual performance routines. As another example, a passive sentence can be treated as a sentence containing a different form of the main verb, which is the semantic inverse of the active form of the verb, rather than being looked at as a transformation of a complete active sentence. In fact, most current transformational grammar approaches treat passives similarly, based on examples such as ***Few people read many books*** versus ***Many books are read by few people***[18], where a passive sentence has a different meaning from its active form, due to quantifier scoping.

As a final and more difficult example of the syntax that this parser could handle, the probable form of the rules which will be needed for handling non-subject relative clauses will be considered in some detail. This type of relative clause can be difficult for non-syntactic parsers to handle, due to the fact that exactly one case filler will be omitted from somewhere in the case frame, indicating that that case is to be filled by the object that the clause is attached to. It is not clear how a parser could keep track of which place the filler is missing from without building a detailed syntactic structure. Several variants are possible:

The paperboy to who(m) I gave the tip quit.
The paperboy who(m) I gave the tip to quit.
The paperboy that I gave the tip to quit.
The paperboy I gave the tip to quit.
The paperboy who(m) I gave the tip quit.

The variability shown here is caused by the availability of several choices along a number of dimensions:

---

[18] [Winograd 83], p. 559.

which of the three relative pronouns[19] to use, whether to leave out the relative pronoun altogether, which of two possible locations for the case marker ***to*** to use, and whether to use the unmarked case instead, which corresponds to the difference between ***I gave the paperboy a tip*** and ***I gave a tip to the paperboy.*** In the current set of verb phrase rules, there is one rule for each type of positional (unmarked) case, and a rule for marked cases which compares the preposition in a prepositional phrase to the markers mentioned in the verb's "expectations". Schematically, the rule for marked cases looks like this:

<Verb-Phrase-missing-case-X>
<marker-for-case-X>
<Noun-Phrase-fitting-case-X>
-->
<Verb-Phrase-including-case-X>.

In order to handle the types of non-subject relative clauses in the examples, the most natural extension would involve adding two new rules for the unmarked object cases, and at most three new rules for marked cases. The need for more than one additional rule for each type of case arises from the OPS5 rule syntax. The most straightforward way to allow the relative pronoun to be absent is to have two rules, one with it and one without it. Similarly, it may be necessary to have an extra rule for the marked cases to allow the preposition to appear in either of the two places allowed when the relative pronoun is present[20]. The rule for incorporating a marked case relative clause into a noun phrase would look something like:

<Noun-Phrase-fitting-case-X>
<marker-for-case-X>
<relative-pronoun-fitting-case-X>
<Verb-Phrase-missing-only-case-X>
-->
<Noun-Phrase-including-Verb-Phrase-as-modifier>.

As for the two other rules for marked cases, both would require the case marker, with no object, to be contained within the verb phrase, with one rule including the relative pronoun and the other lacking it. It should be noted that in the rule shown above, the relative pronoun cannot be ***that***, unlike in the other two rules, and that the verb phrase must not be marked as a question or imperative form, since this would lead to sentences such as ****The paperboy to whom did I give the tip quit*** or ****The paperboy to whom give the tip quit.*** Also note that these rules will be written so that the missing case can belong to a verb case frame embedded any number of levels down in the verb phrase, thus achieving the unlimited motion that English syntax requires.

This approach to relative clauses is similar to a technique involving "derived categories"[21], in which the displacement of a constituent, such as a noun phrase, out of another constituent, such as a verb phrase, is handled by deriving a new category, VP/NP, which is defined as a verb phrase with exactly one noun phrase missing. The constituents of the verb phrase also have this gap, and the overall relative clause is represented by a syntactic production similar to the rule above, the main difference being that the Psli rule takes a case frame approach, rather than the usual phrase structure approach followed in the reference. The chart parsing scheme for handling ambiguity should mesh well with these new rules, since they are very close in form to the current method for extending verb phrases. The sorts of ambiguity that can occur when relative clauses such

---

[19] Although ***who*** is technically incorrect here, its use in informal communication is common enough that it will probably be included as a lower preference alternative.

[20] It would be necessary to actually write and test the rule to be sure, since it might be possible to adjust the rule's variables to achieve this effect. As of final writing, some of the relative clause rules have been implemented and tested, but not these.

[21] [Winograd 83], p. 345.

as these are present is illustrated by the second example sentence, ***The paperboy who(m) I gave the tip to quit***, which could be interpreted as a noun phrase describing a paperboy who was advised to resign. Such ambiguities will simply result in two interpretations which span the length of the utterance, one being a noun phrase and the other being a complete sentence. In isolation, the full sentence would be preferred. In dialogue situations where the noun phrase interpretation could make sense, such as perhaps after the question ***Who got fired?***, dialogue rules could rate each interpretation, and pick the more likely one, with the presence of the other interpretation causing no harm.

5.3. Efficiency

The efficiency of Psli3 compared to its earlier versions will be the last topic addressed in this paper. Each of the three versions uses a mixture of syntactic and semantic knowledge, which in itself should improve the parser's efficiency over purely syntactic schemes by cutting down on syntactic ambiguities, assuming that the semantics employed are not too expensive. Psli1 was designed as a production system implementation of an expectation-based parser such as CA. It handled local ambiguities by creating declarative "expectations" for the different possibilities, which would wait for further clues to be seen, and then create the correct choice. This is the most efficient approach of the three from a run-time point of view, since the only wasted effort is in the creation and matching of the expectation working memory elements for those paths which are not followed, but it requires each rule associated with a local ambiguity to be hand-crafted, resulting in an immensely harder programming task.

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 easier to program, but it resulted in much inefficiency, especially when several correct choices were available at the same time, such as two expectations being satisfied simultaneously. Since only one could be chosen for expansion, and any other correct states built could not be "moved over" to the chosen branch, any correct states not chosen for further growth had to be replicated. In addition, correctly constructed constituents embedded in larger structures that later proved incorrect might have to be reconstructed several times.

This problem with inefficiency was in large part what motivated the Psli3 design, which turned out to provide a much nicer representation for ambiguity as well as improving the efficiency problem. There were two other ways in which the Psli2 design was inefficient compared to Psli3, both stemming from the production system architecture. The first was that time was wasted not only in building the same structure over and over again, but in matching the rules against a much larger working memory, containing the many duplicates of these structures. This slowed down the matching, and therefore the execution, of all the rules. The other problem was that each state in Psli2 had to keep track of its ancestors, so that the rules matching against it could check the rest of its branch for its context. This resulted in the rules matching against a very large number of "ancestor" elements, all of which were very similar. This is a very slow process. In contrast, most of the rules in Psli3, and in particular all of the ones used in normal, error-free parsing, only need to match the lexical position of the states, which are much fewer in number than the "ancestor" elements. The resulting savings are roughly combinatorial in practice, resulting in a speed up of roughly an order of magnitude.

6. Conclusion

The Psli3 semantic chart parser has been presented as an attempted solution to the problem of incorporating syntax into a parser in a flexible fashion. Another goal of its design is to handle ambiguity in a clean manner within the architecture of the system, to avoid having to individually tailor knowledge regarding ambiguous words and structures. It is hoped that the use of a uniform production system architecture will aid in integrating the processing of natural language with other system functions, and that using a psychologically plausible model of cognition will be a useful design heuristic in the future extension of the system. Current extension efforts are directed toward enabling the system to handle inter-sentential dialogue ellipsis in a general fashion.

Bibliography and Reading List

[Anderson 83] Anderson, J. R.
The Architecture of Cognition.
Harvard Press, 1983.

[Becker 75] Becker, J.
The Phrasal Lexicon.
Technical Report 3081, Bolt Beranek and Newman Inc., 1975.

[Birnbaum 81] Birnbaum, L. and Selfridge, M.
Conceptual Analysis of Natural Language.
In Schank, R. C. and Riesbeck, C. K. (editors), *Inside Computer Understanding*, chapter 13. Lawrence Erlbaum Associates, 1981.

[Charniak 76] Charniak, E. and Wilks, Y.
Computational Semantics.
North-Holland, 1976, , chapter 1-9.

[Forgy 81] Forgy, C. L.
OPS5 User's Manual.
Technical Report CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University, July, 1981.

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

[Frederking 84] Frederking, R. E.
An Approach to Natural Language Understanding in Rule-based Systems.
In *Proceedings of the First Annual Workshop on Theoretical Issues in Conceptual Information Processing.* TICIP-1, March, 1984.

[Hendrix 77] Hendrix, G., Sacerdoti, E., Sagalowicz, D., and Slocum, J.
Developing a Natural Language Interface to Complex Data.
Technical Report, SRI International, 1977.

[Moore 73] Moore, J. and Newell, A.
How Can Merlin Understand?
In Gregg, L. (editor), *Knowledge and Cognition*, chapter 9. Lawrence Erlbaum Associates, 1973.
Also CMU Tech Report, November, 1973.

[Newell 78] Newell, A.
Harpy, Production Systems and Human Cognition.
Technical Report, Department of Computer Science, Carnegie-Mellon University, September, 1978.

[Schank 75] Schank, R. C.
Conceptual Information Processing.
North-Holland, 1975, , chapter 3-4.

[Schank 77] Schank, R. C. and Abelson, R.
Scripts, Plans, Goals and Understanding.
Erlbaum, 1977, , chapter 3.

[Schank 80] Schank, R., Lebowitz, M., and Birnbaum, L.
An Integrated Understander.
American Journal of Computational Linguistics 6(1), January-March, 1980.

[Sidner 81] Sidner, C. L.
Focusing for Interpretation of Pronouns.
American Journal of Computational Linguistics 7(4), October-December, 1981.

[Tennant 81] Tennant, H.
Natural Language Processing.
PBI-Petrocelli, New York, 1981.

[Waterman 78] Waterman, D. and Hayes-Roth, F. (editors).
Pattern-Directed Inference Systems.
Academic Press, 1978.
First and last articles, plus Natural Language section.

[Winograd 83] Winograd, Terry.
Language as a Cognitive Process. Volume 1: *Syntax.*
Addison-Wesley, 1983, , chapter 1-4, 7, and Appendix B.

[Woods 72] Woods, W. A.
An Experimental Parsing System for Transition Network Grammars.
Technical Report 2362, Bolt Beranek and Newman Inc., 1972.

[Woods 75] Woods, W. A.
What's in a Link?
In Bobrow, D. G. and Collins, A. (editors), *Representation and Understanding*, chapter 2.
Academic Press, 1975.

[Wright 82] Wright J.M., and Fox M.S.
SRL User's Manual
Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, 1982.

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.

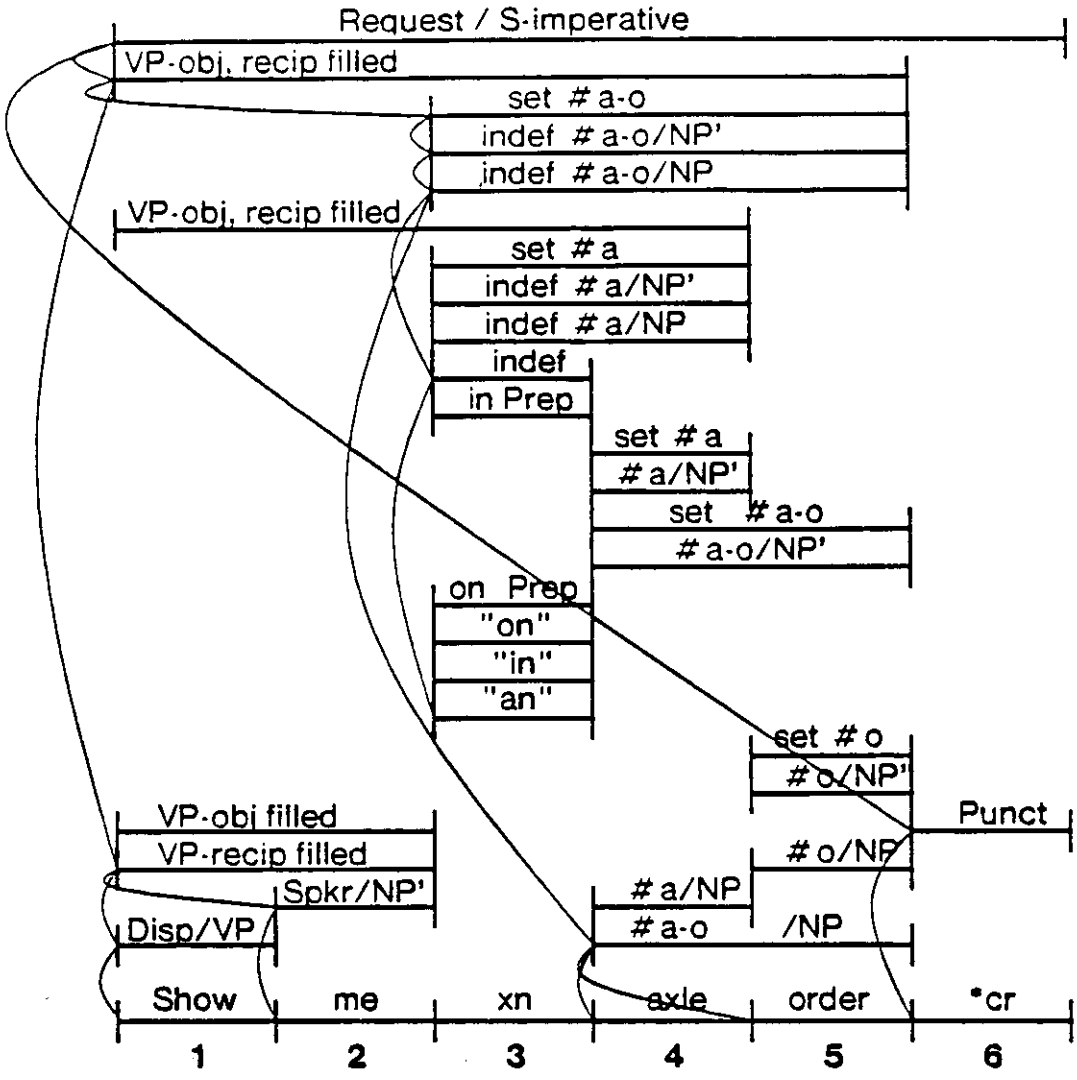

Figure 3-1: Chart built for "Show me xn [sic] axle order"

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.