Responding Intelligently to Unparsable Inputs[1]

Ralph M. Weischedel
John E. Black[2]

Department of Computer and Information Sciences
University of Delaware
Newark, Delaware 19711

All natural language systems are likely to receive inputs for which they are unprepared. The system must be able to respond to such inputs by explicitly indicating the reasons the input could not be understood, so that the user will have precise information for trying to rephrase the input. If natural language communication to data bases, to expert consultant systems, or to any other practical system is to be accepted by other than computer personnel, this is an absolute necessity.

This paper presents several ideas for dealing with parts of this broad problem. One is the use of presupposition to detect user assumptions. The second is relaxation of tests while parsing. The third is a general technique for responding intelligently when no parse can be found. All of these ideas have been implemented and tested in one of two natural language systems. Some of the ideas are heuristics that might be employed by humans; others are engineering solutions for the problem of practical natural language systems.

1. Introduction

A truly *natural* language processing system does not have to have a perfect model of human language use, but it should have knowledge of whatever limitations its model has. Then, for a user who has exceeded these limitations, the system can interactively aid the user to rephrase the input in an acceptable way. This is a prerequisite to any practical application, whether it be natural language communication to a data base, a medical consultation system, or an office automation system. Users will not find such a system practical unless it gives helpful feedback when the system fails to understand an input.

As an example of how a user's input can exceed the system's model, we repeat an anecdote of Woods (1973b) about his system for answering natural language queries about lunar rock samples. One question asked was, "What is the average weight of all your samples?" This overstepped the system's model in at least three ways.

It surpassed the syntactic model, which did not provide for a predeterminer such as "all" preceding another determiner, such as "your" or "the". Therefore, the sentence could not be parsed, even though "What is the average weight of all samples" or "What is the average weight of your samples" could have been.

The semantic capabilities were also surpassed, because semantic rules for translating "weight of" to a functional representation had not been incorporated. Indeed, no data had been included for the weights of the samples.

The third problem was that no semantic translation rules for possession were present. The input violated the system's model of pragmatics, for the designers had not attributed possession of the samples to the machine.

This paper presents three ideas for giving useful feedback when a user exceeds the system's model. The ideas help to identify and explain the system's problem in processing an input in many cases, but do not perform the next step, which is suggesting how the user might rephrase the input.

These ideas have been tested in one of two systems: (1) an intelligent tutor for instruction in a foreign language and (2) a system which computes the

[1] This work was supported in part by the University of Delaware Research Foundation, Inc.

[2] Current address: W.L. Gore & Associates, Inc., Newark, Delaware 19711.

0362-613X/80/020097-13$01.00

presuppositions and entailments of a sentence. For each idea presented in the paper, we will indicate whether it pertains to systems in general or pertains specifically to the foreign language tutor system with its unique position of knowing more of the language than the user.

In Section 2 of this paper we offer a way to recognize that an input exceeds the *semantic* model. In general, the presuppositions or given information (defined later), of a user's input must be true in the system's model of context, for they represent facts that must be shared among the participants of a dialogue. For each presupposition not true in the machine's model, the system should print the false presupposition to identify an assumption that the user cannot make.

Section 3 presents a technique for relaxing constraints to accept sentences that would not parse otherwise. Frequently one wonders whether the *syntactic* component is responsible for much of the inability of previous systems to understand input partially, to isolate parts not understood, and to interpret ill-formed input. A top-down, left-right parser essentially cannot proceed to material to the right of a construction which the grammar is not prepared for. Yet, such a parser should have much predictive ability about what was expected when the block occurred. Section 4 describes a collection of heuristics that capitalize on the predictive abilities of a top-down, left-right parser to produce helpful messages when input is not understood.

Finally, Section 5 discusses related work, and Section 6 presents our conclusions.

2. Using Presuppositions

Semantic information in a sentence is commonly divided into two classes: *given* and *new* information. Given information, or *presupposition,* is that part of the meaning of a sentence which is presumed true in the context of a discourse. New information is the asserted part. For instance, "The defendant stopped beating his wife", has as given information that there is some defendant presumed in the context and that that person had been beating his wife. The new information is that the individual ceased that activity.

Some presuppositions are associated with the use of syntactic constructs. For instance, all noun phrases making definite reference presume that there is a referent in context. All "wh" questions request new information corresponding to the value of a variable and presuppose the set of constraints on the value of the variable. For instance, "Who is playing the tuba", presumes that someone is playing the tuba.

The meaning of particular words is the source of other examples. The use of certain verbs, such as "describe", conveys presuppositions, or given information. The question, "What books describe how President Truman died", has a presupposition that President Truman died. Certain quantifying phrases carry given information, as in "Only project J1 receives parts from New York companies", which presupposes that project J1 receives parts from New York companies.

An analogy can be drawn between given information and preconditions or "input assertions" on a procedure. Given information for definite noun phrases corresponds to predicates on the value of a variable. Given information from the meaning of predicates such as "describe" corresponds to assertions about the state on entry of a procedure. Therefore, given information includes preconditions on the execution of a user request. Furthermore, such preconditions are directly traceable to particular phrases in that request.

The psychological validity of given and new information has been demonstrated by Clark and Haviland (1977) and Haviland and Clark (1974). The psychological process they suggest is that (1) given and new information are first sorted in processing a sentence, (2) memory is then searched to establish that the given information holds in context, and (3) the new information is then asserted in memory.

We have modelled this process in natural language systems. Research reported in Joshi and Weischedel (1977) and Weischedel (1979) demonstrated how to organize an augmented transition network and lexicon to compute the given and new information of a sentence.

In another system, we implemented the second of the three parts of the psychological process suggested by Clark and Haviland. That system was an intelligent tutor which pinpointed errors a student makes while answering questions in German during a reading comprehension exercise (Weischedel, et.al., 1978). A text presented to English-speaking students in German provides a relatively closed world for the tutor system, since questions refer to entities presented in the text and facts about them. Therefore, these can be included as a detachable module of world knowledge specific to the particular text, along with any other world knowledge that is applicable to the set of questions. It is still possible for the student to refer to knowledge not contained in the model, but it is unlikely. Though the students have vast amounts of knowledge not in the system model, they have insufficient vocabulary and syntactic forms to be able to express this knowledge initially.

Thus, in the environment of foreign language instruction, the system is in the unique position of having more vocabulary and syntactic forms than the user and, therefore, has more domain knowledge than the user can express. Obviously most systems do not have this property.

Presuppositions were very effective in the German tutor system, though they are a crucial semantic check for natural language systems in general. Checking presuppositions against the world model of the German tutor provides recognition for several types of errors. First, given information of questions presented by the system must be inferable from a student's answer; otherwise the answer is inappropriate for the question. Consequently, the tutor diagnoses misunderstanding of a question by checking that the given information of a question (which it knows independently) is among the inferences of a student's answer. Only a very simple inference mechanism is used.

Second, given information in the student's answer is checked against the world model. If the given information does not exist in the system's knowledge base, the tutor finds one of two errors. If the presupposition is from a definite noun phrase, the tutor prints the noun phrase and informs the student that it knows of nothing that it could refer to. If the presupposition is associated with the semantics of a particular word, it assumes that this student, who is just learning German, has used the word incorrectly. For instance, *essen* presupposes that the one eating is human; *fressen* presupposes that the one eating is an animal.

Given information is important for any question-answering system with natural language input. The system must check the presuppositions of the input in order to guarantee that the user's assumptions are true in its world. If any are not, the system can list precisely the assumptions which are not true.

These ideas were discussed first in Weischedel (1977) and in Weischedel, et.al. (1978). Kaplan (1977,1979) develops the ideas much further, specifically for data base systems. He postulates a hierarchy for the presuppositions of an English query and has implemented strategies for guiding the user to new queries when the data base would list the empty set in response to a query.

Presupposition has received much attention in linguistics and philosophy; see for example Oh and Dineen (1979), Karttunen (1973), and Karttunen and Peters (1975).

3. Two Mechanisms for Diagnosing Syntactic Failures

We assume that the purpose of a syntactic component is to translate from natural language input to an internal semantic representation of the input. This need not be a completely syntactic process, but may use semantic computations and contextual expectations to guide the parsing/translating process. Several sources could prevent this process from finding a translation of the input. (We will refer to the input component as a "parser", though we do *not* presume that a parse tree is ever explicitly computed.)

An important way that an input may fail to parse is when the user employs incorrect forms of the language. If particular forms are anticipated, they may be explicitly included in the syntactic model along with the appropriate translation mechanism. In the German tutor mentioned in the previous section, there are several examples of this. For instance, English-speaking students frequently forget to put past participles at the end of a clause; e.g. using "Ich habe gegessen das Fleisch" rather than the correct "Ich habe das Fleisch gegessen," (I have eaten the meat). The path in the augmented transition net (ATN) corresponding to the incorrect form computes a message to tell students of the mistake, as well as computing the semantic representation of the answer for semantic analysis. This is particularly effective in the tutor system to catch instances of a student using English syntax patterns rather than German ones.

In a similar way, any natural language processing system may include all anticipated forms and translation rules for them whether or not they are strictly proper for the language.

Another way for a system to accept incorrect forms of language is suggested by observing a common style of writing grammars. Syntactic input components are often designed using a context-free grammar where each grammar rule may be augmented by *predicates* operating on the semantic representations or on the constituents linked by the grammar rule. The predicates must be satisfied for the constituents to be grouped as a larger constituent. (Of course, the grammar is no longer context-free then.) Augmented phrase structure grammars (Heidorn, 1975) encode parsers and translators specifically in this way. The augmented transition network formalism also directly lends itself to writing parsers and translators in this way by the predicates on arcs. The version of systemic grammar implemented by Winograd (1972) has this flavor as well. Still another example of this style of writing parsers is the linguistic string parser of Sager (1973) and Grishman (1973).

A straightforward example of the use of such predicates is for subject-verb agreement. It is easy for a user to make mistakes in long English sentences, resulting in parser failure. A solution would be simply to remove the predicate from the rule. However, Grishman (1973) reports from their experience in processing scientific texts that the predicates effectively eliminate a large number of spurious parses.

We suggest that, instead of forcing all predicates to be satisfied or ignoring the information inherent in them, that the designer should designate that *certain predicates can be relaxed,* with a record being kept of each predicate not satisfied during parsing. Only parses yielding the fewest unsatisfied predicates are completed. Since the number of predicates that evaluate to false in a partial parse is a non-decreasing number, only those partial parses with the fewest unsatisfied predicates have to be continued. Thus, the number of spurious parses added should be small. (Instead of assuming that all failed predicates have equal weight, one could assign a partial order to them; but we have not yet investigated this.)

This strategy was very effective in the German tutor system. Not only were several predicates allowed to fail, but also a procedural specialist was attached to the appropriate arc of the ATN to compute a specific error message and probable cause for the student's error. Subject-verb agreement is one example. Another is noun phrase declension, which is crucial to distinguishing "Das Mädchen gab dem Mann einen Hut" (the girl gave the man a hat) from "Dem Mädchen gab der Mann einen Hut" (the man gave the girl a hat).

The notion of allowing certain predicates to go unsatisfied is much more general than the highly special environment of the German tutor. In the system described in the next section, several predicates were made optional or "failable". By "failable" we mean that the predicates ought to be true for the pattern to match, but could be false without preventing the pattern from matching if there would be no parse with all such predicates true. In addition to subject-verb agreement, pronominal case was also made failable. The two together allow a sentence such as "Me think him win often" to be parsed, even though the parser has a model of correct language.

4. Responses to Unparsable Sentences

Some sentences will not be parsable even using the mechanisms described in the previous section. If one uses an augmented transition network as a top-down, left-right parser, the arcs leaving a state where a parse is blocked offer a set of predictions or expectations regarding what should occur next in the input string. These predictions include more than just the symbols or constituents that were expected to follow; they include the partial interpretation that was being followed. In fact, this partial interpretation is potentially far more informative than the symbols or constituents that were expected next. (In the realm of programming languages, an Algol compiler that gives a syntax error message of "SEMI-COLON EXPECTED" can be quite frustrating since the cause of the problem can be quite difficult to find.) Thus, one of our major goals was to develop and test heuristics that would enable a natural language system to describe what interpretation it was following as an explanation of why it expected specific items which were not present.

Our approach is that the parser writer can assign meaning to the states of a parser as it is being written, somewhat analogous to assigning meaning to programs (Floyd, 1967). Floyd suggested postulating computational states between the actions of a program and associating predicates with these states to capture the intent of the computational state. States are explicitly given in an ATN. The designer's insight into the meaning of a particular state offers potentially a wealth of information that can be presented to the user about the interpretation being followed and why it failed. This may be of significant help in selecting an alternative way to express the input.

The meaning of an ATN state may be specified by an ordered list of condition-action pairs, encoded as arbitrary LISP functions. These conditions and actions may be functions of the current word in the input, the previous word in the input, any ATN register having a value as of that state, any ATN register from higher levels in the graph, or the sequence of states traversed.

These condition-action pairs furnish a natural way to distinguish among several interpretations or paths that are collapsed at a particular state. The conditions are used to unravel the collapsed paths by referring to the ATN registers and input string. The action for any given condition provides a flexible way of computing and printing messages the parser writer has chosen to describe the interpretation being followed.

In general, the effectiveness of this idea for generating responses to unparsable sentences will depend on heuristics for determining the state at which the problem in the input was encountered. These ideas should be very effective for natural language front ends to applications such as data base systems, expert consultant systems, and computer-assisted instruction systems.

The ideas do *not* presume that the parser operates sequentially or prior to semantic or pragmatic components. The ideas would fit in equally well in a multiprocessing environment where syntactic, semantic, and pragmatic components communicate asynchronously, such as GUS (Bobrow, et.al. 1977). In a multiprocessing system, one would have to write the condition-action pairs to use information and decisions from the other components. The only assumption we have made is that the parser is top-down, left-right, and is written in the formalism of the ATN.

4.1 Selecting a State from the Set of Possible States

In essence, when a parse is blocked, one wants the partial parse nearest to being complete. We have chosen to select partial parses that have moved furthest in the input string, or, in other words, those that match the longest initial string. However, there may be several paths and blocked states matching the longest initial string. Furthermore, the parse may have gone several words beyond the point where the real problem occurred.

As a heuristic, *states where a block has occurred are selected only if they are on a "longest path" matching the most input words.* The "length of a path" is defined to be the number of arcs traversed other than PUSH or JUMP arcs with the universally (vacuously) true test, since those arcs make no test on the input string nor consume any part of it. (The pseudo-arcs POP are counted.) If there are still several states, one can be selected nondeterministically.

Given the state S selected by the heuristic above, there is a tree of states which are reachable from S using only a string of PUSH or JUMP arcs with the universally true test. S is the root of such a tree. The meaning of each of the states of the tree may oftentimes be summarized by the parser designer into one brief description characterizing all of them as one conceptual unit (rather than a disjunction of descriptions of each). For states where this seems inappropriate, a special function (LOOKAHEAD) can be added as an action in the meaning of S to call the message generating routine recursively for each state at distance one from S in the tree described above. Using these two ideas we found that selecting the root S for its meaning, while ignoring its descendants, proved satisfactory in our tests.

The heuristic for selecting one partial parse and one state along a path for it was implemented in a particular parser, to be described in section 4.2. We tested these ideas by constructing for each state an unparsable input such that the heuristic would select that state. Some states either could not be a blocking point or could be one only by a non-English input, such as, "John forced Mary not." After eliminating such states, we tested the heuristic on one sentence for each remaining state.

For an input that does not parse, there is some maximal initial input string consumed by any of the partial parse paths. Consider the set of states on the partial parse paths such that at each such state the maximal input string has been parsed in reaching that state. (The set can be more than a singleton even if there is only one path, since PUSH, JUMP, and POP arcs do not consume input symbols.) For the 39 example sentences, the average number of states in that set was four.

To measure the effectiveness of employing the heuristic of using only states at the end of a "longest" path (where JUMP and PUSH arcs with a universally true test are not counted in the length of the path), we counted the number of "longest" paths for each example. In 34 of the 39 test cases, this heuristic yielded only one state. In each of the five remaining cases, two states were left as the last state of a longest path.

As mentioned earlier, when more than one state is left after selecting only states at the end of a longest path, one can be selected nondeterministically. In three of the five test cases where nondeterminism was used, the two states would have produced essentially the same message, and therefore using both states would have added no insight.

Thus, in our test the heuristic seemed very effective. Of course, the effectiveness of the heuristic depends in large part on the style in which the parser is written. We describe the parser next.

4.2 The Parser on which the Ideas were Tested

We tested these ideas on a parser written in 1975 as part of a Ph.D. thesis (Weischedel, 1975). The purpose of the parser was to demonstrate how to compute two special classes of inferences, presuppositions and entailments of English sentences, by using an ATN and a lexicon. Because of its special purpose, the system included many constructions and lexical items of interest for their presuppositional or entailment content. For the same reason, many frequently occurring constructions of English were not implemented, e.g. conjunction reduction, reduced relative clauses, several consecutive prepositional phrases, and adverbials.

A subset of constructions were selected from the linguistic analysis of Anderson (1970), which was a basis of defining the lexical classes of the Linguistic String Parser, described in Sager (1973) and Grishman (1973). Anderson's analysis defined lexical classes by the left contexts (subject) in which a predicate (verb or noun) could occur, the right contexts (object) in which a predicate could occur, and the transformations which could act upon these strings of subject, predicate and object. (Note, in this section the word "predicate" refers to part of an English clause, not a boolean test as in Section 3.) Such strings define the parsable sentential forms; the transformations acting upon the strings give further forms. Of course, our ATN parser encoded all surface forms in the graphs explicitly. The actions on the ATN arcs had the effect of inverting the transformations given by Anderson while moving along an ATN path matching the surface form.

Conditions on the arcs were very significant in the style of our parser. For instance, a condition before POPping from the sentential level checks whether the left and right contexts matched for the word or predicate X form a string in the linguistic model of Anderson (1970).

In the lexical entries each semantic representation for a word was associated with corresponding lexical classes. Finding the semantic representation of a sentence, therefore, required determining lexical classes for each of the words.

The arc condition which checks whether a predicate X occurred in appropriate right and left contexts was *not* one of the ones we declared to be "failable", because this condition was necessary and sufficient for determining associated semantic representations. When the condition was not satisfied, the parser did not have in its lexicon a semantic representation for the word X. Consequently, the condition offered a very tight constraint for ascertaining when the parser had no semantic representation corresponding to an interpretation. Maintaining such tight control over what the system could and could not translate is somewhat akin in philosophy to using strongly typed programming languages with much error checking. People do not seem to have such a style in using natural languages; however, it might be a useful engineering principle for natural language systems where accuracy of understanding is crucial.

Another consciously applied strategy in designing the parser was to separate into distinct paths many strings which conceivably could have been merged. The criterion for making a distinct path was whether a string which was syntactically differentiable also had a distinct semantic representation. For instance, cleft sentences, such as "It was Mary who won", could have been incorporated by simply listing left and right contexts for "be" in the lexicon. However, the syntactic form has distinct meaning, namely in the example, the presupposition that "Someone won." Therefore, the parser has a special path for cleft sentences.

This aspect of style yielded several relatively long paths with little branching. For such paths, the messages for a blocked parse can pinpoint the interpretation that was being followed and what was expected next. Examples of this are provided in section 4.3.

One of the major advantages of testing our ideas on this parser was the fact that there were many ways in which a sentence could fail to parse. The parser was already available, but more important for the assigning of meaning to its states, its designer was readily available. A further reason for selecting this parser was that it did cover a wide range of constructions and was not a toy grammar.

In general, we specifically avoided enhancing the grammar to remove limitations. We wanted a full range of tests and example problems to exercise our ideas as thoroughly as possible. However, simple bugs such as erroneous omissions in lexical entries or typographical errors were corrected as they were detected. Also, we did add one action to most arcs to save surface phrases as they were found, for more helpful responses to the user.

The major drawback in selecting this parser for experimentation is its original purpose. Although its purpose is very precise, it did not have a natural task domain. Without a task, it seems impossible to make some significant tests, such as giving naive users a specific goal in the domain, then measuring how many trials they require to achieve the goal in the restricted natural language.

4.3 Examples and Analyses

In this section, we have organized example states, messages, and analyses around several themes; each of the following subsections comprises one theme. All graphs here are much simplified recursive transition net approximations of the actual graphs in Weischedel (1979). A double circle indicates a pop arc. Lower case indicates terminals; upper case indicates nonterminals.

4.3.1 Appropriate Phrasing for Naive Users

Though the parser writer may know precisely what interpretation was being followed and what caused it to block at a given state, it is very challenging to phrase that knowledge in a way helpful to the user. This is a problem common to all natural language systems, but the degree of the problem varies with the application of the system and with the style of the grammar. For instance, in the environment of an intelligent tutor for computer-assisted language instruction, the user is learning or has learned many informal grammatical concepts of the language (though these may not directly correspond to the formal ones implemented in the parser). Consequently, the parser writer in creating response messages as part of the condition-action pairs can use these concepts to pinpoint for the user the reason the parser blocked. In other applications, the user might have few, if any, concepts of grammar.

Since our tests were conducted on the English parser for generating presuppositions and entailments, the response messages were aimed at general users having only a few informal concepts of language, such as sentence, subject, verb, and object. In addition, the responses often include examples with a similar structure, rather than using technical terms. For instance, suppose that the phrase "It was the class ..." was being interpreted as a cleft sentence when it was blocked. The system prints that the input was being interpreted in the same way as a sentence such as "It is John who left," rather than calling it a cleft sentence.

The style of the particular parser also has a significant effect on the ability to phrase the reason for a parsing failure. For instance, if one uses a "semantic grammar" (Burton, 1976 and Brown and Burton, 1975) the parser writer can use the concepts of the domain encoded in the grammar as a powerful description of the interpretation being followed and of the cause of a blocked parse. In INLAND (Hendrix, et.al., 1978), one can see how effective the domain concepts encoded in the semantic grammar can be in

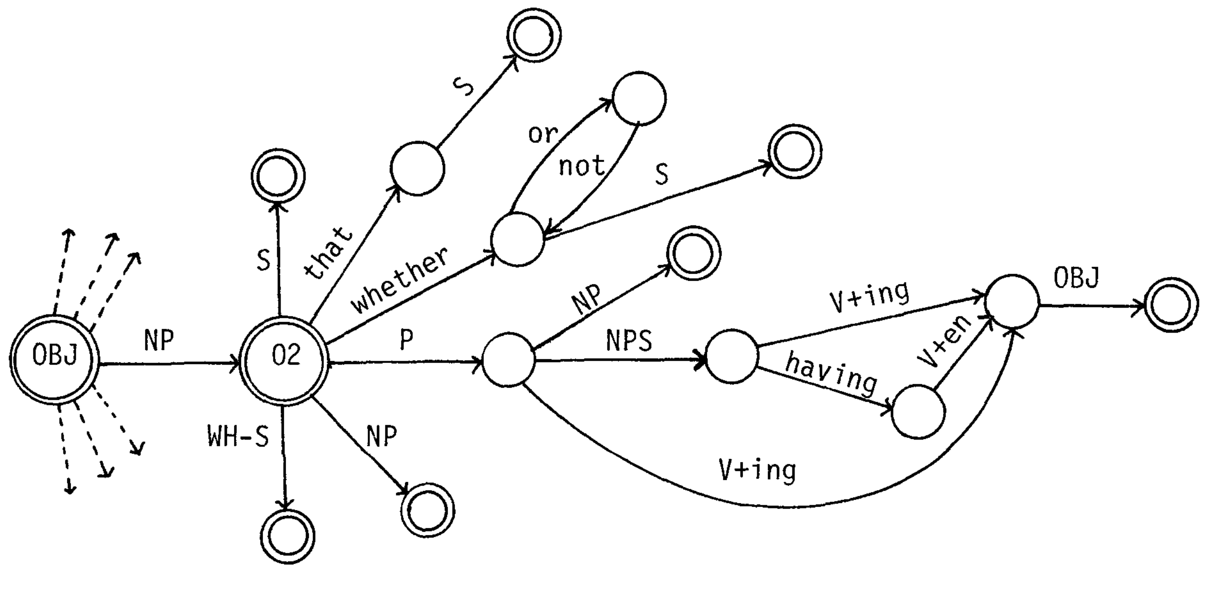

Figure 1. Paths involving state 02.

responding to the user. One class of response messages in INLAND is a list of the elements that could have occurred next when the parse blocked. Even though this list does not indicate the interpretation being followed, the semantic concepts of the domain which could occur next (e.g. ship) are more meaningful to the user than such a list for a general-purpose grammar would be (e.g. noun phrase).

In effect, then, we tested the idea of using the meanings of states to generate responses on the hardest case, where the parser is general and the users are completely naive about even informal grammatical concepts. The remainder of this subsection describes the problems we encountered, by examples.

State 02 of Figure 1 exemplifies a very frustrating aspect of devising appropriate descriptions. 02 is part of the subgraph that recognizes right contexts (objects) of predicates. The meaning of the nonterminals is as follows: NP, a noun phrase; NPS, a possessive form of a noun phrase; S, a declarative sentence; WH-S, a wh-question; P, a preposition; V+ing, a verb's present participle; and V+en, a verb's past participle. Though there are four different possible reasons for a parse to block at 02, each of which is rather reliably recognized by a simple condition, the messages describing the problem are not precise.

Four condition-action pairs represent the meaning assigned to 02. The first checks for the input string being empty. If it is, the lexical entry for the predicate does not include the appropriate right context being matched, and therefore has no translation for it. Though the problem is pinpointed, describing it to the user is not easy; examples of uses that the system can understand seem to be the most helpful for this. Example 1 demonstrates the message. Each lexical item corresponding to a predicate has a list of sentences, one for each implemented left-right context pair; the examples are stored on disk files and read only if requested.

Example 1:

```
THE PROFESSOR PREVENTED A DULL LECTURE /.
NO PARSES
THE PROFESSOR PREVENTED A DULL LECTURE
STUCK AT THE END OF THE SENTENCE
SUBJECT UNDERSTOOD TO BE:  'THE PROFESSOR'
VERB UNDERSTOOD TO BE:  'PREVENTED'
THE WORD 'PREVENTED' IS BEING USED IN A
  WAY UNKNOWN TO THIS SYSTEM.
WOULD YOU LIKE EXAMPLES ?  *YES
EXAMPLES FOR THE USAGE OF 'PREVENTED'
  'THAT THE STUDENTS DID NOT ATTEND THE
     LECTURE PREVENTED THE PROFESSOR
     FROM ASSIGNING THE TEXT /.'
  'JOHN PREVENTED MARY FROM ATTENDING THE
     LECTURE /.'
  'JOHN WAS PREVENTED FROM LEAVING BY
     MARY /.'
  'JOHN WAS PREVENTED FROM TRANSLATING THE
     ASSIGNMENT BY ME /.'
```

A second condition-action pair associated with 02 is apparently never used, because the parse can always continue beyond state 02. This pair checks two conditions: whether the current input symbol is "that" and whether two noun phrases form an appropriate right context for the predicate found. In this case the parser should have interpreted "that" as being used referentially. However, we had not included its referential sense in the lexicon. The message associated with the pair would pinpoint the problem were this the final state of a longest parse. In one instance, the parser can go one arc farther by pushing for a noun phrase and verifying that the predicate allows such a right context. Example 2 is such an instance.

Example 2:

```
I WILL GIVE DR SMITH THAT /.

NO PARSES

I WILL GIVE DR SMITH
STUCK AT THE WORD 'THAT'

LOOKING FOR A NOUN PHRASE, BUT THE WORD
   'THAT' CANNOT BE USED AS A PRONOUN
   IN THIS SYSTEM TO REFER TO SOMETHING
```

Far more likely circumstances are that the parser can continue by interpreting "that" as the beginning of a relative clause modifying the noun phrase matched in reaching state 02. The meaning of the last state in that case does not pinpoint the problem, but at least it does explain the interpretation being followed, as demonstrated in Example 3.

Example 3:

```
I ASSIGNED THE STUDENT THAT /.

NO PARSES

I ASSIGNED THE STUDENT THAT
STUCK AT THE END OF THE SENTENCE

INTERPRETING 'THAT' AS THE BEGINNING OF A
   RELATIVE CLAUSE, SUCH AS THE FOLLOWING
   RELATIVE CLAUSES MODIFYING 'THE
   STUDENT':  'THE STUDENT THAT WON,' 'THE
   STUDENT WHICH WON,' OR 'THE STUDENT WHO
   WON /.
AT PRESENT, THE SYSTEM DOES NOT UNDERSTAND
   'THAT' USED ALONE AS IN 'I KNOW THAT'
THIS ERROR OCCURRED WHILE THE SYSTEM WAS
   WORKING ON WHAT IT INTERPRETED TO BE A
   SENTENCE EMBEDDED WITHIN THE MAIN
   SENTENCE.  THE SYSTEM'S INTERPRETATION
   OF THE WAY IT EXPECTED THAT EMBEDDED
   SENTENCE TO FIT INTO THE COMPLETE
   SENTENCE WAS:
SUBJECT UNDERSTOOD TO BE:  'I'
VERB UNDERSTOOD TO BE:  'ASSIGNED'
LOOKING FOR AN APPROPRIATE OBJECT FOR
   'ASSIGNED'.
```

In Example 3, the parser has gone one word beyond the real difficulty in the input. The problem of going beyond where the real block occurred is more apparent than real for state 02, however. If we had not decided a priori that for the purposes of testing our ideas we would not add to the parser or lexicon, we would have simply added the referential sense of "that" to the lexicon.

A third condition-action pair associated with 02 deals with an error in a design decision made when first building the parser. In Anderson (1970), the lexical analysis cites many predicates whose right contexts include prepositions specific to a particular predicate. For instance, "tell" has right contexts specifically allowing "of" or "about". Paths leaving 02 upon finding a preposition require that it specifically be listed in the lexical entry of the predicate. However, in 1975 we made the erroneous assumption that only one preposition would be listed per predicate. The condition-action pair checks whether this could be the problem; unfortunately, describing the problem to a naive user is itself a problem. As Example 4 indicates, the best description we could think of is the same as for the first condition-action pair of 02 (Example 1).

Example 4:

```
A PROFESSOR PRESSURED THE STUDENT ABOUT
  LEAVING /.

NO PARSES

A PROFESSOR PRESSURED THE STUDENT
STUCK AT THE WORD 'ABOUT'

SUBJECT UNDERSTOOD TO BE:  'A PROFESSOR'
VERB UNDERSTOOD TO BE:  'PRESSURED'
THE WORD 'PRESSURED' IS BEING USED IN A
  WAY UNKNOWN TO THIS SYSTEM.

WOULD YOU LIKE EXAMPLES ? *YES

  'JOHN WAS PRESSURED INTO LEAVING /.'
  'THE PROFESSOR PRESSURED THE STUDENTS
     INTO STUDYING THE TEXT /.'
```

02 has one more condition-action pair which is used if no other pair applies. There are two possible causes in this case: the predicate's lexical entry might not include the right context present in the sentence or the NP that was just matched could have prepositional phrases modifying it. The message is essentially the same as that in Example 1. The cause, like the message, is not precise in this case.

State 02 illustrates that even though the designer may assign condition-action pairs that pinpoint the cause for a sentence not being parsed, descriptions of the cause may not be as precise or helpful to a naive user. Thus, the messages can be less helpful than one would have hoped.

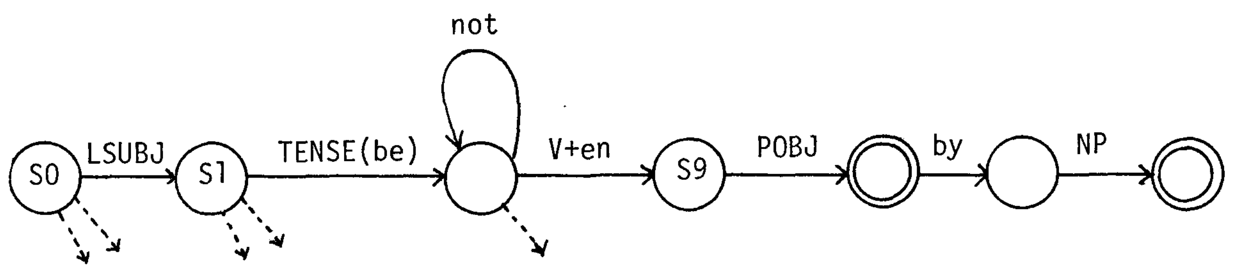

Figure 2. The path containing S9.

4.3.2 The Precision Possible

In spite of the problem illustrated in the last section, much precision is possible in messages to the user. For example, state S17, which appears in the full diagrams of Weischedel (1979), is on the path for recognizing a subset of the cleft sentences. The path that it is on is an example of many paths that are very long, with little branching, and that correspond to a particular interpretation. (This is a characteristic of the style of the parser.) At S17, the word "it", a string of tense and modal elements ending in a form of "be", and a noun phrase have been matched. The only arc leaving S17 matches a relative clause. If a block occurs here, either the input was not a complete cleft sentence, or the relative clause began in an unparsable way. The message printed appears as Example 5. The portion of the message describing relative clause restrictions was generated from the condition-action pairs of a different state; that state's pairs were involved because S17's pair explicitly called the LOOKAHEAD function after printing the first part of the message.

Example 5:

```
WAS IT JOHN ?

NO PARSES

WAS IT JOHN
STUCK AT THE END OF THE SENTENCE

INTERPRETING 'WAS IT JOHN' AS IN
     SENTENCES OF THE FORM:  'WAS IT
     JOHN THAT WAS DULL.'

EXPECTED A RELATIVE CLAUSE.  EXAMPLES
     OF RELATIVE CLAUSES ARE:  'WHICH
     THE STUDENT SELECTED' OR 'THAT THE
     PROFESSOR TOOK'.  THIS SYSTEM
     EXPECTS RELATIVE CLAUSES TO BEGIN
     WITH 'WHO', 'WHOM', 'WHICH', OR
     'THAT'.
```

Another example of the kind of precision possible comes from one of the messages of S9, shown in Figure 2. LSUBJ matches left contexts of a predicate; in this case the left context is the surface subject of the verb. TENSE(be) will match any tensed elements ending in a form of "be". V+en represents a past participle of a verb. POBJ looks for the right context of the verb, thus matching right contexts from which the surface subject was syntactically moved. By the time S9 has been reached, the system is interpreting the input as a passive sentence.

The first condition-action pair associated with S9 checks whether the past participle found is in a particular lexical subcategory, because passives of that subcategory are treated in a special manner. The arcs for the special case were not implemented. The printed message appears in Example 6 and corresponds exactly to the omission in the grammar.

Example 6:

```
I WAS DISAPPOINTED THAT THE LECTURE IS
     CROWDED /.

NO PARSES

I WAS DISAPPOINTED
STUCK AT THE WORD 'THAT'

CURRENT SYSTEM CANNOT HANDLE PASSIVE
     SENTENCES INVOLVING 'DISAPPOINTED'.
```

A second condition-action pair for S9 always prints a message if the first one did not apply. This clause corresponds to a general reason for blocking at S9: none of the expected right contexts for the verb could be found. This could arise if the lexical entry did not list the necessary right context and therefore had no translation for this case. It could also arise in a sentence such as "That I won was told immediately to Mary." (Recall that we simply did not include adverbial adjuncts in the parser.) Just as the cause is not very precise for this instance, the message given in Example 7 cannot be either. The example sentences given as output do parse. The input does not parse because the lexical entry simply did not include a noun phrase as one of its right contexts.

Example 7:

```
ABE WAS BELIEVED BY MARY /.

NO PARSES

ABE WAS BELIEVED
STUCK AT THE WORD 'BY'

SUBJECT UNDERSTOOD TO BE:  'ABE'
VERB UNDERSTOOD TO BE:  'WAS BELIEVED'
IN GENERAL, PHRASES INDICATING TIME,
     PLACE, OR MANNER ARE NOT ALLOWED.
     ALTERNATELY, YOU MAY HAVE USED THE
     VERB 'BELIEVED' IN AN UNKNOWN WAY.
WOULD YOU LIKE EXAMPLES ?  *YES

EXAMPLES FOR THE USAGE OF 'BELIEVED'

  'JOHN BELIEVED THAT I LEFT /.'
  'I BELIEVED JOHN ATTENDED THE
     LECTURE /.'
  'JOHN BELIEVED IN THE PROFESSOR'S
     TEACHING THE COURSE /.'
  'I BELIEVED IN JOHN'S HAVING TAKEN
     THE TEXT /.'
  'MARY BELIEVED IN JOHN'S TRANSLATING
     OF THE ASSIGNMENT /.'
  'THAT MARY LEFT WAS BELIEVED BY THE
     STUDENTS /.'
```

Using states S17 and S9 along with the corresponding Examples 5 and 6, we have demonstrated that the messages can sometimes pinpoint the cause of a parsing failure. There are many other states whose condition-action pairs yield a precise diagnosis for the cause of a parsing failure.

4.3.3 Embedded Sentences

For sentences with embeddings, merely to give information based on the last state of the longest path seems intuitively insufficient, for explanation of the higher levels of the sentence may be ignored if the message is based solely on the last state at an embedded level. Consequently, the system prints messages for each incomplete sentential level represented in the partial parse. First, the message from the last state is printed. Then, starting at the highest level, an explanatory message is printed for each incomplete sentential level.

These messages are printed using the same ideas as described for the last state on the longest path. The criterion for selecting states is simple. The parser's stack contains all the states with an exiting PUSH arc that has been started but remains unfinished. Of the states in that stack, only the ones corresponding to a sentential level are relevant; these begin with an "S" or an "I" in our graph. The set of condition-action pairs for these states was written assuming this was the last state on the longest path. Consequently, we wrote a second, smaller set of condition-action pairs especially assuming that partially parsed embedded sentences follow this state.

Example 8 illustrates messages for embedded sentences. The output beginning with "This error occurred while ..." is the start of messages from higher level, partially parsed sentences. The useful hint at the true problem in parsing Example 8 comes from one of the states in the system's stack; the right context necessary to parse Example 8 has not been defined.

Example 8:

```
DID MARY ASK DR SMITH IF I ATTENDED
      THE LECTURE ?

NO PARSES

DID MARY ASK DR SMITH IF I ATTENDED
      THE LECTURE
STUCK AT THE END OF THE SENTENCE

EXPECTED '/,' TO SEPARATE 'IF I
      ATTENDED THE LECTURE' FROM
      A QUESTION WHICH IS EXPECTED
      TO FOLLOW THE '/,'.  YOUR
      INPUT BEGAN WITH AN 'IF' CLAUSE.
      IF THAT CLAUSE WAS NOT FULLY
      PROCESSED, THERE ARE SEVERAL
      POSSIBLE REASONS:
    1) ADVERBIAL MATERIAL TELLING HOW,
       WHEN, OR WHERE CANNOT BE
       PROCESSED
    2) NO PREPOSITIONAL PHRASES CAN
       MODIFY A NOUN (IN THIS SYSTEM).
THIS ERROR OCCURRED WHILE THE SYSTEM WAS
      WORKING ON WHAT IT INTERPRETED TO
      BE A SENTENCE EMBEDDED WITHIN THE
      MAIN SENTENCE.  THE SYSTEM'S
      INTERPRETATION OF THE WAY IT
      EXPECTED THAT EMBEDDED SENTENCE TO
      FIT INTO THE COMPLETE SENTENCE WAS:
SUBJECT UNDERSTOOD TO BE:  'MARY'
VERB UNDERSTOOD TO BE:  'DID ASK'
LOOKING FOR AN APPROPRIATE OBJECT FOR
      'ASK'.
```

4.3.4 Testing the Longest Path Heuristic

A serious difficulty in using the longest path as a heuristic for generating responses is that the parser may be able to continue further in the input than where the real parsing problem occurred. To examine how well the longest path heuristic performs in locating the true cause of the problem, we analyzed the 39 sentences described in section 4.1. In only three of the 39 cases did the parser continue beyond the point where the true problem occurred. Contrasted with this success rate, Woods (personal communication, 1977) reported that in LUNAR, the parser very often was able to continue beyond the point of the problem in the input before becoming blocked.

There are several factors that affect the success of the longest path heuristic. One is the extent of the grammar; the fact that adverbial adjuncts, reduced relative clauses, and multiple, consecutive prepositional phrases are not present in the grammar we tested undoubtedly contributed to the high success rate. Therefore, the heuristic should be very effective in applied natural language interfaces that are constrained.

Second, the style of grammar can affect the success of the heuristic. For instance, our grammar immediately upon finding the main predicate (e.g. verb) of a clause requires that its syntactic expectations for right contexts of that particular main predicate be satisfied at each step through the remainder of the string containing a right context. Also, as near as possible, semantically different senses were usually separated into distinct paths, even though they might have been collapsed into one.

Third, applying semantic constraints and expectations while parsing should also contribute to the effectiveness of the longest path heuristics, just as the syntactic constraints and expectations do. The additional constraints will inhibit the parser from continuing beyond a problem in the input by preventing it from processing a phrase with the expected syntactic form but which is unacceptable semantically. For instance, suppose the actual right context of a predicate (e.g. verb) starts with a noun phrase, but the lexicon lists no right contexts for the predicate that begin with a noun phrase. A parser might be able to continue by interpreting the noun phrase as an adverbial adjunct specifying a time, such as "last night." If the parser interacts with a semantic component requiring that the noun phrase be interpretable as a time specification, the parser could not go on by interpreting the noun phrase erroneously. Since our grammar does not interact with a semantic component, we are interested in testing the longest path heuristic in RUS (Bobrow, 1978), a grammar which does interact closely with semantics.

4.3.5 Further Observations

A natural criterion for evaluating this strategy for unparsable sentences is the cost, both in processing and programming development. In processing, very little is added. Clearly, a small fraction of the parsing time and memory usage is added to record the longest path and to generate messages for the last state on it (and possibly one state per incomplete sentential level). However, it is easy to see that this is a minute fraction compared to the time and memory in searching for a parse.

On the other hand, significant additional effort is required of the programmer to devise condition-action pairs for each state. However, spending that time has benefits in addition to the response ability added to the system. Analyzing the parser to develop the meaning of each state clarifies the programmer's understanding of the system. Furthermore, it serves as significant documentation, since it describes the intent of the programmer at each point.

For our graph having approximately 110 states, the average number of condition-action pairs per state was 1.4. The code for these pairs amounted to approximately one page of a listing for the conditions and approximately nine pages for the constant character strings used in generating the (rather long) printed messages. Therefore, it is clear that the condition-action pairs do not require a lot of programming, but do require a better understanding and description of the parser.

5. Related Work

Several other projects have concentrated on giving meaningful responses to partially understood input and of correcting erroneous assumptions.

Kaplan (1977,1978,1979) reports on research which extends the notion of presupposition. Furthermore, he has developed algorithms for computing the extended notion called presumption, particularly taking advantage of the simplifying aspects of natural language queries of a data base. The algorithms give helpful responses to data base users when the query as stated would have the empty set as a response. Mays (1980) deals with presumptions related to users' perceptions of the logical structure of a data base.

Codd, et.al. (1978) describes the first version of a system called RENDEZVOUS, specifically addressing the same problems as our paper, but proposing very different approaches. Unlike the ideas presented here, RENDEZVOUS is aimed only at interfaces to relational data bases. It provides many interesting human engineering features for clarification dialogue, even to a menu-driven specification of a formal query when natural language queries prove unsatisfactory.

Some very promising work which is complementary to ours is reported in Hendrix, et.al. (1978) and Hendrix (1977). They report on a new software tool LIFER, which enables rapid development of semantic grammars (Burton, 1976 and Brown and Burton, 1975). LIFER provides some error messages for unparsable forms by printing the possible items that could appear at the point where the parser could not proceed. Their heuristic for selecting one place where the block occurred is similar to ours. Combining the following additional features of LIFER with our work could offer a powerful natural language interface. LIFER allows naive users the ability to add synonyms for previously known words and to define new syntactic forms for sentences by the user presenting a sentence in the new form and an equivalent sentence which is already parsable. It also provides an automatic facility for handling ellipsis.

Kwasny and Sondheimer (1979) have extended our notion of selectively relaxing predicates to deal with co-occurrence violations and relaxation of expected word categories. Their paper also reports a uniform way of treating ellipsis and conjunction, including gapping.

Allen (1979) argues that good clarification dialogue requires that the system have a model of the plan the user is following and of how the sequence of speech acts by the user fits into that plan. We agree, and one of our long-term goals is use of a model of user goals, plans, and speech acts for this purpose. Other computational models of speech acts appear in Cohen and Perrault (1979), Levin and Moore (1978), and Mann (1979).

Pattern-matching as an alternative to a top-down, left-to-right parser, has often been suggested as a means of processing ill-formed input, as discussed in Kwasny and Sondheimer (1979), for example. Hayes and Reddy (1979) also advocate pattern-matching as a part of an approach that they are implementing to cover the broad spectrum of problems in graceful interaction, including anaphora resolution, explanation facilities, flexible parsing, generating descriptions of entities in context, monitoring focus, and robust communication.

6. Conclusions

We have drawn eight conclusions from our experience with the two systems on which our heuristics were tested. *First*, computing the presuppositions, or given information, of user input provides a means for detecting some of the user's assumptions inherent in the input. These may be checked against world knowledge in the system to recognize discrepancies between the user's model and the system's world model and to point out an incorrect assumption to the user.

Second, an effective strategy for increasing the robustness of a parser is to allow relaxation of predicates (on ATN arcs) that the parser designer designates as relaxable, or "failable." The system will prefer parses where no such predicates are false. If no parse can be found with all predicates true, the system will relax the predicates designated as failable, and will search for a parse with the fewest failable predicates false.

The remaining conclusions regard our technique of assigning meanings to states as a means of generating responses when no parse can be found. The *third* conclusion is that the meanings of states, used with the longest path heuristic, can often pinpoint the cause of an input not parsing.

Fourth, though the cause of the input not parsing can often be pinpointed with the technique, describing the cause to the user may be quite difficult because of the technical nature of the problem in the input.

Fifth, the effectiveness of the longest path heuristic in correctly selecting the state corresponding to the actual problem in processing the input depends on the style of the grammar and the extent of the subset of language covered. The more constrained the language used in the application domain, the less possibility for the parser continuing beyond the point of the problem. Alternatively, the more syntactic and semantic constraints used as expectations by the parser, the greater the likelihood that the problem in the input will correctly correspond to a violated expectation, since violated expectations will help prevent the parser from going beyond the point of the problem. This does not conflict with the notion of relaxing predicates, since the longest path heuristic is used only after no parse can be found even after relaxing predicates. In our grammar, the longest path heuristic selected the correct state in over 90% of the test cases.

Sixth, based on the two previous conclusions, the heuristic of responding using the meaning of states will be most effective in semantic grammars or in parsers that interact closely with semantic processes.

Seventh, the longest path heuristic adds only a small fraction to the computing time and memory usage during parsing. Furthermore, adding the condition-action pairs to represent the meaning of states does not require a lot of programming, but does require a better understanding of the parser.

Eighth and last, the technique of assigning meaning to states is applicable to explaining compile-time errors in programming languages as well.

We also suggest four areas for further work. First, the heuristics should be tested in a parser that interacts closely with semantics while parsing. The purpose for that is twofold: (1) to more effectively respond to the user by paraphrasing the partial interpretation and semantic expectations when the input is unparsable and (2) to test further the effectiveness of the longest path heuristic. Second, the user's goals and intent are critical constraints which we have not incorporated in any of our heuristics. The aforementioned work on computational models of speech acts and dialogue games provide a starting point for this. A third area is to combine the ideas presented here with the heuristics in LIFER (Hendrix, et.al., 1978); the combination could provide a very user-oriented, flexible interface. Fourth, the effectiveness of our technique for responding to unparsable sentences should be examined in the domain of programming language compilers, because the user of a compiler knows many technical terms which the parser writer can employ in messages to convey effectively the cause of a blocked parse.

Acknowledgements

The authors gratefully acknowledge the many valuable contributions of the referees and George Heidorn to improving the exposition. Norm Sondheimer also contributed much in many discussions of our ideas.

References

Allen, James F., "A Plan-Based Approach to Speech Act Recognition," Ph.D. Thesis, Dept. of Computer Science, University of Toronto, Toronto, Canada, 1979.

Anderson, Barbara B., "Transformationally Based English Strings and their Word Subclasses," String Program Reports No. 7, Linguistic String Program, New York University, New York, NY 1970.

Bobrow, D. G., R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd, "GUS, A Frame Driven Dialog System," *Artificial Intelligence 8,* 2, 1977.

Bobrow, Robert J., "The RUS System," in B. L. Webber and R. Bobrow, Research in Natural Language Understanding, BBN Report 3878, Bolt Beranek and Newman Inc., Cambridge, MA, 1978.

Brown, J. S. and R. R. Burton, "Multiple Representations of Knowledge for Tutorial Reasoning." In D. G. Bobrow and A. Collins, Eds., *Representation and Understanding,* New York: Academic Press, 1975.

Burton, R. R., "Semantic Grammar: An Engineering Technique for Construction of Natural Language Understanding Systems." BBN Report 3453, Bolt, Beranek, and Newman, Cambridge, Mass. Also, Ph.D. Dissertation, University of California, Irvine, CA, 1976.

Clark, Herbert H. and Susan E. Haviland, "Comprehension and the Given-New Contract." In R. Freedle, Ed., *Discourse Processes: Advances in Research and Theory,* Vol. 1. *Discourse Production and Comprehension.* Norwood, NJ: Ablex Publishing Corporation, 1977.

Codd, E. F., R. S. Arnold, J. M. Cadiou, C. L. Chang, N. Roussopoulos, "RENDEZVOUS Version 1: An Experimental English-Language Query Formulation System for Casual Users of Relational Data Bases," Research Report RJ2144 (29407), IBM Research Laboratory, San Jose, CA, 1978.

Cohen, Philip R. and C. Raymond Perrault, "Elements of a Plan-Based Theory of Speech Acts," *Cognitive Science 3,* 3, 1979.

Floyd, R. W., "Assigning Meanings to Programs," *Proc. of a Symposium in Applied Mathematics,* Vol. 19. American Mathematical Society, 1967.

Grishman, Ralph, "Implementation of the String Parser of English." In R. Rustin, Ed., *Natural Language Processing.* New York: Algorithmics Press, 1973.

Haviland, Susan E. and Herbert H. Clark, "What's New? Acquiring new information as a process in comprehension." *Journal of Verbal Learning and Verbal Behavior, 13,* 1974.

Hayes, P. and R. Reddy, "An Anatomy of Graceful Interaction in Spoken and Written Man-Machine Communication," Technical Report, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1979.

Heidorn, George E., "Augmented Phrase Structure Grammars," *Theoretical Issues in Natural Language Processing,* 1975.

Hendrix, Gary G., Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum, "Developing a Natural Language Interface to Complex Data," *ACM Transactions on Data Base Systems 3,* 2, 1978.

Hendrix, G. G., "Human Engineering for Applied Natural Language Processing," *Proc. 5th International Joint Conference on Artificial Intelligence,* Cambridge, MA, August, 1977.

Joshi, Aravind K. and Ralph M. Weischedel, "Computation of a Subclass of Inferences: Presupposition and Entailment." *American Journal of Computational Linguistics, 1977,* 1, Microfiche 63, 1977.

Kaplan, S. Jerrold, "Cooperative Responses from a Natural Language Data Base Query System: Preliminary Report," Technical Report, Department of Computer and Information Science, Moore School, University of Pennsylvania, Philadelphia, PA, 1977.

Kaplan, S. Jerrold, "Indirect Responses to Loaded Questions," *Theoretical Issues in Natural Language Processing-2,* University of Illinois at Urbana-Champaign, July, 1978.

Kaplan, S. Jerrold, "Cooperative Responses from a Natural Language Data Base Query System," Ph.D. Dissertation, Dept. of Computer & Information Science, University of Pennsylvania, Philadelphia, PA, 1979.

Karttunen, L., "Presuppositions of Compound Sentences," *Linguistic Inquiry, 4,* 1973.

Karttunen, L. and S. Peters, "Conventional Implicature in Montague Grammar," *Proc. of the First Annual Meeting of the Berkeley Linguistics Society,* Berkeley, CA, 1975

Kwasny, Stan and Norman K. Sondheimer, "Ungrammaticality and Extra-Grammaticality in Natural Language Understanding Systems," *Proceedings of the 17th Annual Meeting of the Association for Computational Linguistics,* 1979.

Levin, J. A. and J. A. Moore, "Dialogue Games: Metacommunication Structures for Natural Language Interaction," *Cognitive Science 1,* 4, 1978.

Mann, W. C., "Dialogue Games," in K. Hintikka, et.al., Eds., *Models of Dialogue,* Amsterdam: North-Holland Publishing Company, 1979.

Mays, Eric, "Correcting Misconceptions About Data Base Structure," *Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence,* 1980.

Oh, Choon-Kyu and David A. Dineen, (Eds.), *Presupposition, Vol. 11, Syntax and Semantics,* New York: Academic Press, 1979.

Sager, Naomi, "The String Parser for Scientific Literature." In R. Rustin, Ed., *Natural Language Processing.* New York: Algorithmics Press, 1973.

Weischedel, Ralph M., "Computation of a Unique Subclass of Inferences: Presupposition and Entailment," Ph.D. Thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1975.

Weischedel, Ralph M., "A New Semantic Computation While Parsing: Presupposition and Entailment." In C. Oh and D. Dineen, Eds., *Presupposition, Vol. 11, Syntax and Semantics,* New York: Academic Press, 1979.

Weischedel, Ralph M., "Please Re-phrase," Technical Report #77/1, Department of Computer and Information Sciences, University of Delaware, Newark, DE 1977.

Weischedel, Ralph M., Wilfried Voge, and Mark James, "An Artificial Intelligence Approach to Language Instruction," *Artificial Intelligence 10,* 3, 1978.

Winograd, T., *Understanding Natural Language,* New York: Academic Press, Inc., 1972.

Woods, W. A., "Transition Network Grammars for Natural Language Analysis," *Comm. ACM, 13,* 10, 1970.

Woods, W. A., "An Experimental Parsing System for Transition Network Grammars." In R. Rustin, Ed., *Natural Language Processing.* New York: Algorithmics Press, 1973a.

Woods, W. A., "Progress in natural language understanding -- An application to lunar geology,," *AFIPS Conference Proceedings, NCC.* Montvale, NJ: AFIPS Press, 1973b.

Woods, W. A., Personal Communication, 1977.

Ralph M. Weischedel is an assistant professor in the Department of Computer and Information Sciences at the University of Delaware. He received the Ph.D. degree in computer and information science from the University of Pennsylvania in 1975.

John E. Black is Director of Computer Systems at W. L. Gore & Associates, Inc., in Newark, Delaware. He received the M.S. degree in computer and information sciences from the University of Delaware in 1979.