MENU-BASED NATURAL LANGUAGE UNDERSTANDING

Harry R. Tennant, Kenneth M. Ross,
Richard M. Saenz, Craig W. Thompson,
and James R. Miller
Computer Science Laboratory
Central Research Laboratories
Texas Instruments Incorporated
Dallas, Texas

ABSTRACT

This paper describes the NLMenu System, a menu-based natural language understanding system. Rather than requiring the user to type his input to the system, input to NLMenu is made by selecting items from a set of dynamically changing menus. Active menus and items are determined by a predictive left-corner parser that accesses a semantic grammar and lexicon. The advantage of this approach is that all inputs to the NLMenu System can be understood thus giving a 0% failure rate. A companion system that can automatically generate interfaces to relational databases is also discussed.

I INTRODUCTION

Much research into the building of natural language interfaces has been going on for the past 15 years. The primary direction that this research has taken is to improve and extend the capabilities and coverage of natural language interfaces. Thus, work has focused on constructing and using new formalisms (both syntactically and semantically based) and on improving the grammars and/or semantics necessary for characterizing the range of sentences to be handled by the system. The ultimate goal of this work is to give natural language interfaces the ability to understand larger and larger classes of input sentences.

Tennant (1980) is one of the few attempts to consider the problem of evaluating natural language interfaces. The results reported by Tennant concerning his evaluation of the PLANES System are discouraging. These results show that a major problem with PLANES was the negative expectations created by the system's inability to understand input sentences. The inability of PLANES to handle sentences that were input caused the users to infer that many other sentences would not be correctly handled. These inferences about PLANES' capabilities resulted in much user frustration because of their very limited assumptions about what PLANES could understand. It rendered them unable to successfully solve many of the problems they were assigned as part of the evaluation of PLANES, even though these problems had been specifically designed to correspond to some relatively straightforward queries that PLANES could understand. Additionally, users did not successfully adapt to the system's limitations after some amount of use.

One class of problem that caused negative and false user expectations was the user's ability to distinguish between the limitations in the system's conceptual coverage and the system's linguistic coverage. Often, users would attempt to paraphrase a sentence many times when the reason for the system's lack of understanding was due to the fact that the system did not have data about the query being asked (i.e. the question exceeded the conceptual coverage of the system). Conversely, users' queries would often fail because they were phrased in a way that the system could not handle (i.e. the question exceeded the linguistic coverage of the system).

The problem pointed out by Tennant seems to be a general problem that must be faced by any natural language interface. If the system is unable to understand user inputs, then the user will infer that many other sentences cannot be understood. Often, these expectations serve to severely limit the classes of sentences that users input, thus making the natural language interface virtually unusable for them. If natural language interfaces are to be made usable for novice users, with little or no knowledge of the domain of the system to which they are interfacing, then negative and false expectations about system capabilities and performance must be prevented.

The most obvious way to prevent users of a natural language interface from having negative expectations is expand the coverage of that interface to the point where practically all inputs are understood. By doing this, most sentences that are input will be understood and few negative expectations will be created for the user. Then users will have enough confidence in the natural language interface to attempt to input a wide range of sentences, most of which will be understood. However, natural language interfaces with the ability to understand virtually all input sentences are far beyond current technology. Thus, users will continue to have many negative expectations about system coverage.

A possible solution to this problem is the use of a set of training sessions to teach the user the syntax of the system. However, there are several problems with this. First, it does not allow

untrained novices to use such a system. Second, it assumes that infrequent users will take with them and remember what they learned about the coverage of the system. Both of these are unreasonable restrictions.

II A DESCRIPTION OF THE NLMENU SYSTEM

In this paper, we will employ a technique that applies current technology (current grammar formalisms, parsing techniques, etc.) to make natural language interface systems meet the criteria of usability by novice users. To do this, user expectations must closely match system performance. Thus, the interface system must somehow make it clear to the user what the coverage of the system is. Rather than requiring the user to type his input to the natural language understanding system, the user is presented with a set of menus on the upper half of a high resolution bit map display. He can choose the words and phrases that make up his query with a mouse. As the user chooses items, they are inserted into a window on the lower half of the screen so that he can see the sentence he is constructing. As a sentence is constructed, the active menus and items in them change to reflect only the legal choices, given the portion of the sentence that has already been input. At any point in the construction of a natural language sentence, only those words or phrases that could legally come next will be displayed for the user to select.

Sentences which cannot be processed by the natural language system can never be input to the system, giving a 0% failure rate. In this way, the scope and limitations of the system are made immediately clear to the user and only understandable sentences can be input. Thus, all queries fall within the linguistic and conceptual coverage of the system.

A. The Grammar Formalism

The grammars used in the NLMenu System are context-free semantic grammars written with phrase structure rules. These rules may contain the standard abbreviatory conventions used by linguists for writing phrase structure rules. Curly brackets ({}, sometimes called braces) are used to indicate optional elements in a rule. Additionally, square brackets ([]) are used as well. They have two uses. First, in conjunction with curly brackets. Since it is difficult to allow rules to be written in two dimensions as linguists do, where alternatives in curly brackets are written one below the other, we require that each alternative be put in square brackets. Thus, the rule below in (1) would be written as shown in (2).

(1) A --> B $\left\{\begin{matrix} C\ X \\ E\ Y \end{matrix}\right\}$ D

(2) A --> B {[C X] [E Y]} D

Note that for single alternatives, the square brackets can be deleted without loss of information. We permit this and therefore {A B} is equivalent to {[A] [B]}. The second use of square brackets is inside of parentheses. An example of this appears in rule (3) below.

(3) Q --> R ([M N] V)

This rule is an abbreviation for two rules, Q --> R M N and Q --> R V.

Any arbitrary context-free grammar is permitted except for those grammars containing two classes of rules. These are rules of the form X --> null and rules that generate cycles, for example, A --> B, B --> C, C --> D and D --> A. The elimination of the second class of rules causes no difficulty and does not impair a grammar writer in any way. If the second class of rules were permitted, an infinite number of parses would result for sentences of grammars using them. The elimination of the first class of rules causes a small inconvenience in that it prevents grammar writers from using the existence of null nodes in parse trees to account for certain unbounded dependencies like those found in questions like "Who do you think I saw?" which are said in some linguistic theories to contain a null noun phrase after the word "saw". However, alternative grammatical treatments, not requiring a null noun phrase, are also commonly used. Thus, the prohibition of such rules requires that these alternative grammatical treatments be used.

In addition to synactic information indicating the allowable sentences, the grammar formalism also contains semantic information that determines what the meaning of each input sentence is. This is done by using lambda calculus. The mechanism is similar to the one used in Montague Grammar and the various theories that build on Montague's work. Associated with every word in the lexicon, there is a translation. This translation is a portion of the meaning of a sentence in which the word appears. In order to properly combine the translations of the words in a sentence together, there is a rule associated with each context-free rule indicating the order in which the translations of the symbols on the right side of the arrow of a context-free rule are to be combined. These rules are parenthesized lists of numbers where the number 1 refers to the first item after the arrow, the number 2 to the second, etc.

For example, for the rule X --> A B C D, a possible rule indicating how to combine translations might be (3 (1 2 4)). This rule means that the translation of A is taken as a function and applied to the translation of B as its argument. This resulting new translation is then taken as a function and applied to the translation of 4 as its argument. This resulting translation is then the argument to the translation of 3 which is the function. In general, the translation of leftmost number applies to the translation of the number to its right as the argument. The result of this then is a function which applies to the translation of the item to its right as the

argument. However, parentheses can override this as in the example above. For rules containing abbreviatory conventions, one translation rule must be written for every possible expansion of the rule.

Translations that are functions are of the form "(lambda x (... x ...)). When this is applied to an item like "c" as the argument, "c" is plugged in for every occurrence of x after the "lambda x" that is not within the scope of a more deeply embedded "lambda x". This is called lambda conversion and the result is just the expression with the "lambda x" stripped off of the front and the substitution made.

B. The Parser

The parser used in the NLMenu system is an implementation of an enhanced version of the modified left-corner algorithm described in Ross (1982). Ross (1982) is a continuation of the work described in Ross (1981) and builds on that work and on the work of Griffiths and Petrick (1965). The enhancements enable the parser to parse a word at a time and to predict the set of next possible words in a sentence, given the input that has come before.

Griffiths and Petrick (1965) propose several algorithms for recognizing sentences of context-free grammars in the general case. One of these algorithms, the NBT (Non-selective Bottom to Top) Algorithm, has since been called the "left-corner" algorithm. Of late, interest has been rekindled in left-corner parsers. Slocum (1981) shows that a left-corner parser inspired by Griffiths and Petrick's algorithm performs quite well when compared with parsers based on a Cocke-Kasami-Younger algorithm (see Younger 1967).

Although algorithms to recognize or parse context-free grammars can be stated in terms of push-down store automata, G+P state their algorithm in terms of Turing machines to make its operation clearer. A somewhat modified version of their algorithm will be given in the next section. These modifications transform the recognition algorithm into a parsing algorithm.

The G+P algorithm employs two push down stacks. The modified algorithm to be given below will use three, called alpha, beta and gamma. Turing machine instructions are of the following form, where A, B, C, D, E and F can be arbitrary strings of symbols from the terminal and non-terminal alphabet.

[A,B,C] ---> [D,E,F] if "Conditions"

This is to be interpreted as follows-

If A is on top of stack alpha,
B is on top of stack beta,
C is on top of stack gamma,
and "Conditions" are satisfied
then replace A by D, B by E, and C by F.

The modified algorithm follows-

(1) [V1,X,Y] ---> [Ø,V2 ... Vn t X,A Y]
if A --- V1 V2 ... Vn is a rule of the phrase structure grammar X is in the set of nonterminals and Y is anything

(2) [X,t,A] ---> [A X,Ø,Ø]
if A is in the set of nonterminals

(3) [B,B,Y] ---> [Ø,Ø,Y]
if B is in the set of nonterminals or terminals

To begin, put the terminal string to be parsed followed by END on stack alpha. Put the nonterminal which is to be the root node of the tree to be constructed followed by END on stack beta. Put END on stack gamma. The symbol t is neither a terminal nor a nonterminal. When END is on top of each stack, the string has been recognized. If none of the turing machine instructions apply and END is not on the top of each stack, the path which led to this situation was a bad path and does not yield a valid parse.

The rules necessary to give a parse tree can be stated informally (i.e. not in terms of turing machine instructions) as follows:

When (1) is applied, attach V1 beneath A.

When (3) is applied, attach the B on alpha B as the right daughter of the top symbol on gamma.

Note that there is a formal statement of the parsing version of NBT in Griffiths (1965). However, it is somewhat more complicated and obscures what is going on during the parse. Therefore, the informal procedure given above will be used instead.

The SBT (Selective Bottom to Top) algorithm is a selective version of the NBT algorithm and is also given in G+P. The only difference between the two is that the SBT algorithm employs a selective technique for increasing the efficiency of the algorithm. In the terminology of G+P, a selective technique is one that eliminates bad parse paths before trying them. The selective technique employed is the use of a reachability matrix. A reachability matrix indicates whether each non-terminal node in the grammar can dominate each terminal or non-terminal in the grammar in a tree where that terminal or non-terminal is on the left-most branch. To use it, an additional condition is put on rule (1) requiring that X can reach down to A.

Ross (1981) modifies the SBT Algorithm to directly handle grammar rules utilizing several abbreviatory conventions that are often used when writing grammars. Thus, parentheses (indicating optional nodes) and curly brackets (indicating that the items within are alternatives) can appear

in rules that the parser accesses when parsing a string. These modifications will not be discussed in this paper but the parser employed in the NLMenu System incorporates them because efficiency is increased, as discussed in Ross (1981).

At this point, the statement of the algorithm is completely neutral with respect to control structure. At the beginning of a parse, there is only one 3-tuple. However, because the algorithm is non-deterministic, there are potentially points during a parse at which more than one turing machine instruction can apply. Each of the parse paths resulting from an application of a different turing machine instruction to the same parser state sends the parser off on a possible parse path. Each of these possible paths could result in a valid parse and all must be followed to completion. In order to assure this, it is necessary to proceed in some principled way.

One strategy is to push one state as far as it will go. That is, apply one of the rules that are applicable, get a new state, and then apply one of the applicable rules to that new state. This can continue until either no rules apply or a parse is found. If no rules apply, it was a bad parse path. If a parse is found, it is one of possibly many parses for the sentence. In either case, the algorithm must continue on and pursue all other alternative paths. One way to do this and assure that all alternatives are pursued is to backtrack to the last choice point, pick another applicable rule, and continue in the manner described earlier. By doing this until the parser has backed up through all possible choice points, all parses of the sentence will be found. A parser that works in this manner is a depth-first backtracking parser. This is probably the most straightforward control structure for a left-corner parser.

Alternative control structures are possible. Rather than pursuing one path as far as possible, one could go down one parse path, leave that path before it is finished and then start another. The first parse path could then be pursued later from the point at which it was stopped. It is necessary to use an alternative control structure to enable parsing to begin before the entire input string is available.

To enable the parser to function as described above, the control structure for a depth-first parser described earlier is used. To introduce the ability to begin parsing given only a subset of the input string, the item MORE is inserted after the last input item that is given to the parser. If no other instructions apply and MORE is on top of stack alpha, the parser must begin to backtrack as described earlier. Additionally, the contents of stack beta and gamma must be saved. Once all backtracking is completed, additional input is put on alpha and parsing begins again with a set of states, each containing the new input string on alpha and one of the saved tuples containing beta and gamma. Each of these states is a distinct parse path.

To parse a word at a time, the first word of the sentence followed by MORE is put on alpha. The parser will then go as far as it can, given this word, and a set of tuples containing beta and gamma will result. Then, each of these tuples along with the next word is passed to the parser. The ability to parse a word at a time is essential for the NLMenu System. However, it is also beneficial for more traditional natural language interfaces. It can increase the perceived speed of any parser since work can proceed as the user is typing and composing his input. Note that a rubout facility can be added by saving the beta-gamma tuples that result after parsing for each of the words. Such a facility is used by the NLMenu System.

The ability to predict the set of possible nth words of a sentence, given the first n-1 words of the sentence is the final modification necessary to enable this parser to be used for menu-based natural language understanding. This feature can be added in a straightforward way. Given any beta-gamma pair representing one of the parse paths active after n-1 words of the sentence have been input, it is possible to determine the set of words that will allow that state to continue. This is by examing the top-most symbol on stack beta of the tuple. It represents the most immediate goal of that parse state. To determine all the words that can come next, given that goal, the set of all nodes that are reachable from that node as a left daughter must be determined. This information is easily obtainable from the reachability matrix discussed earlier. Once the set of reachable nodes is determined, all that need be done is find the subset of these that can dominate lexical material. If this is done for all of the beta-gamma pairs that resulted after parsing the first n-1 words and the union of the sets that result is taken, the resulting set is a list of all of the lexical categories that could come next. The list of next words is easily determined from this.

III APPLICATIONS OF THE NLMENU SYSTEMS

Although a wide class of applications are appropriate for menu-based natural language interfaces, our effort thus far has concentrated on building interfaces to relational databases. This has had several important consequences. First, it has made it easy to compare our interfaces to those that have been built by others because a prime application area for natural language interfaces has been to databases. Second, the process of producing an interface to any arbitrary set of relations has been automated.

A. Comparison to Existing Systems

We have run a series of pilot studies to evaluate the performance of an NLMenu interface to

the parts-suppliers database described in Data (1977). These studies were similar to the ones described in Tennant (1980) that evaluated the PLANES system. Our results were more encouraging than Tennant's. They indicated that both experienced computer users and naive subjects can successfully use a menu-based natural language interface to a database to solve problems. All subjects were successfully able to solve all of their problems.

Comments from subjects indicated that although the phrasing of a query might not have been exactly how the subject would have chosen to ask the question in an unconstrained, traditional system, the subjects were not bothered by this and could find the alternative phrasing without any difficulty. One factor that appeared to be important in this was the displaying of the entire set of menus at all times. In cases where it was not clear which item on an active menu would lead to the users desired query, users looked at the inactive menus for hints on how to proceed. Additionally, the existence of a rubout facility that enabled users to rubout phrases they had input as far back as desired encouraged them to explore the system to determine how a sentence might be phrased. There was no penalty for choosing an item which did not allow a user to continue his question in the way he desired. All that the user had to do was rub it out and pick again.

B. Automatically Building NLMenu Interfaces To Relational Databases

The system outlined in this section is a companion system to NLMenu. It allows NLMenu interfaces to an arbitrary set of relations to be constructed in a quick and concise way. Other researchers have examined the problem of constructing portable natural language interfaces. These include Kaplan (1979), Harris (1979), Hendrix and Lewis (1981), and Grosz et. al. (1982). While the work described here shares similarities, it differs in several ways. Our interface specification dialogue is simple, short, and is supported by the database data dictionary. It is intended for the informed user, not necessarily a database designer and certainly not a grammar expert. Information is obtained from this informed user through a menu-based natural language dialogue. Thus, the interface that builds interfaces is extremely easy to use.

1. Implementation

The system for automatically generating NLMenu interfaces to relational databases is divided into two basic components. One component, BUILD-INTERFACE, produces a domain specific data structure called a "portable spec" by engaging the user in an NLMenu dialog. The other component, MAKE-PORTABLE-INTERFACE, generates a semantic grammar and lexicon from the "portable spec".

The MAKE-PORTABLE-INTERFACE component takes as input a "portable spec", uses it to instantiate a domain independent core grammar and lexicon, and returns a semantic grammar and a semantic lexicon pair, which defines an NLMENU interface. The core grammar and lexicon can be small (21 grammar rules and 40 lexical entries at present), but the size of the resulting semantic grammars and lexicons will depend on the portable spec.

A portable-spec consists of a list of categories. The categories are as follows. The COVERED TABLES list specifies all relations or views that the interface will cover. The retrieval, insertion, deletion and modification relations specify ACCESS RIGHTS for the covered tables. Non-numeric attributes, CLASSIFY ATTRIBUTES according to type. Computable attributes are numeric attributes that are averageable, summable, etc. A user may choose not to cover some attributes in interface. IDENTIFYING ATTRIBUTES are attributes that can be used to identify the rows. Typically, identifying-attributes will include the key attributes, but may include other attributes if they better identify tuples (rows) or may even not include a full key if one seeks to identify sets of rows together. TWO TABLE JOINS specify supported join paths between tables. THREE TABLE JOINS specify supported "relationships" (in the entity-relationship data model sense) where one relation relates 2 others. The EDITED ITEMS specification records old and new values for menu phrases and the window they appear in. The EDITED HELP provides a way for users to add to, modify or replace automatically generated help messages associated with a menu item. Values to these last categories record changes that a user makes to his default menu screen to customize phrasings or help messages for an application.

The BUILD-INTERFACES component is a menu-based natural language interface and thus is really another application of the NLMenu system to an interface problem. It elicits the information required to build up a "portable spec" from the user. In addition to allowing the user to create an interface, it also allows the user to modify or combine existing interfaces. The user may also grant interfaces to other users, revoke them, or drop them. The database management system controls which users have access to which interfaces.

2. Advantages

The system for automatically constructing NLMenu interfaces enjoys several practical and theoretical advantages. These advantages are outlined below.

End-users can construct natural language interfaces to their own data in minutes, not weeks or years, and without the aid of a grammar specialist. There is heavy dependence on a data dictionary but not on linguistic information.

The interface builder can control coverage. He can decide to make an interface that covers only a semantically related subset of his

tables. He can choose to include some attributes and hide other attributes so that they cannot be mentioned. He can choose to support various kinds of joins with natural language phrases. He can mirror the access rights of a user in his interface, so that the interface will allow him to insert, delete, and modify as well as just retrieve and only from those tables that he has the specified privileges on. Thus, interfaces are highly tunable and the term "coverage" can be given precise definition. Patchy coverage is avoided because of the uniform way in which the interface is constructed.

Automatically generated natural language interfaces are robust with respect to database changes; interfaces are easy to change if the user adds or deletes tables or changes table descriptions. One need only modify the portable spec to reflect the changes and regenerate the interface.

Automatically generated NLMenu interfaces are guaranteed to be correct (bug free). The interaction in which users specify the parameters defining an interface, ensures that parameters are valid, i.e. they correspond to real tables, attributes and domains. Instantiating a debugged core grammar with valid parameters yields a correct interface.

Natural language interfaces are constructed from semantically related tables that the user owns or has been granted and they reflect his access privileges (retrieval), insertion, etc). By extension, natural language interfaces become database objects in their own right. They are sharable (grantable and revokable) in a controlled way. A user can have several such NLMenu interfaces. Each gives him a user-view of a semantically related set of data. This notion of a view is like the notion of a database schema found in network and hierarchical but not relational systems. In relational systems, there is no convenient way for grouping tables together that are semantically related. Furthermore, an NLMenu interface can be treated as an object and can be granted to other users, so a user acting as a database administrator can make NLMenu interfaces for classes of users too naive to build them themselves (like executives). Furthermore, interfaces can be combined by merging portable specs and so user's can combine different, related user-views if they wish.

Since an interface covers exactly and only the data and operations that the user chooses, it can be considered to be a "model of the user" in that it provide a well-bounded language that reflects a semantically related view of the user's data and operations.

A final advantage is that even if an automatically generated interface is for some reason not quite what is needed for some application, it is much easier to first generate an interface this way and then modify it to suit specific needs than it is to build the entire interface by hand. This has been demonstrated already in the prototype where an automatically generated interface required for an appliction for another group at TI was manually altered to provide pictorial database capabilities.

Taken together, the advantages listed above pave the way for low cost, maintainable interfaces to relational database systems. Many of the advantages are novel when considered with respect to past work. This approach makes it possible for a much broader class of users and applications to use menu-based, natural language interfaces to databases.

3. Features of NLMenu Interfaces to Databases

The NLMenu system does not store the words that correspond to open class data base attributes in the lexicon as many other systems do. Instead, a meta category called an "expert" is stored in the lexicon. They may be user supplied or defaulted and they are arbitrary chunks of code. Possible implementations include directly doing a database lookup and presenting the user with a list of items to choose from or presenting the user with a type in window which is constrained to only allow input in the desired type or format (for example, for a date).

Many systems allow ellipsis to permit the user to, in effect, ask a parameterized query. We approach this problem by making all phrases that were generated by experts be "mouse sensitive" in the sentence. To change the value of a data item, all that needs to be done is to move the mouse over the sentence. When a data item is encountered, it is boxed by the mouse cursor. To change it, one merely clicks on the mouse. The expert which originally produced that data item is then called, allowing the user to change that item to something else.

The grammars produced by the automatic generation system permit ambiguity. However, the ambiguity occurs in a small set of well-defined situations involving relative clause attachment. Because of this, it has been possible to define a bracketed and indented format that clearly indicates the source of ambiguity to the user and allows him to choose between alternative readings. Additionally, by constraining the parser to obey several human parsing strategies, as described in Ross (1981), the user is displayed a set of possible readings in which the most likely candidate comes first. The user is told that the first bracketed structure is most probably the one he intended.

IV CONCLUSIONS

The menu approach to natural language input has many advantages over the traditional typing approach. Most importantly, every sentence that

is input is understood. Thus, a 100% success rate for queries input is achieved. Implementation time is greatly decreased because the grammars required can be much smaller. Generally, writing a thorough grammar for an application of a natural language understanding system consumes most of the development time. Note that the reason larger grammars are needed in traditional systems is that every possible paraphrase of a sentence must be understood. In a menu-based system, only one paraphrase is needed. The user will be guided to this paraphrase by the menus.

The fact that the menu-based natural language understanding systems guide the user to the input he desires is also beneficial for two other reasons. First, confused users who don't know how to formulate their input need not compose their input without help. They only need to recognize their input by looking at the menus. They need not formulate their input in a vacuum. Secondly, the extent of the system's conceptual coverage will be apparent. The user will immediately know what the system knows about and what it does not know about.

Only allowing for one paraphrase of each allowable query not only makes the grammar smaller. The lexicon is smaller as well. NLMenu lexicons must be smaller because if they were the size of a lexicon standardly used for a natural language interface, the menus would be much too large and would therefore be unmanageable. Thus, it is possible that limitations will be imposed on the system by the size of the menus. Menus can necessarily not be too big or the user will be swamped with choices and will be unable to find the right one. Several points must be made here. First, even though an inactive menu containing, say, a class of modifiers, might have one hundred modifiers, it is likely that all of these will never be active at the same time. Given a semantic grammar with five different classes of nouns, it will most likely be the case that only one fifth of the modifiers will make sense as a modifier for any of those nouns. Thus, an active modifier menu will have roughly twenty items in it. We have constructed NLMenu interfaces to about ten databases, some reasonably large, and we have had no problem with the size of the menus getting unmanageable.

The NLMenu System and the companion system to automatically build NLMenu interfaces that are described in this paper are both implemented in Lisp Machine Lisp on an LMI Lisp Machine. It has also proved to be feasible to put them on a microcomputer. Two factors were responsible for this: the word by word parse and the smaller grammars. Parsing a word at a time means that most of the work necessary to parse a sentence is done before the sentence has been completely input. Thus, the perceived parse time is much less than it otherwise would be. Parse time is also made faster by the smaller grammars because it is a function of grammar size so the smaller the grammar, the faster the parse will be performed. Smaller grammars can be dealt with much more easily on a microcomputer with limited memory available. Both systems have been implemented in C on the Texas Instruments Professional Computer. These implementation are based on the Lisp Machine implementations but were done by another division of TI. These second implementations will be available as a software package that will interface either locally to RSI's Oracle relational DBMS which uses SQL 3.0 as the query language or to various remote computers running DBMS's that use SQL 3.0 as their query language.

V REFERENCES

Data, C. J. An introduction to database systems. New York: Addison-Wesley, 1977.

Griffiths, T. On procedures for constructing structural descriptions for three parsing algorithms, Communications of the ACM, 1965, 8, 594.

Griffiths, T. and Petrick, S. R., On the relative efficiencies of context-free grammar recognizers, Communications of the ACM, 1965, 8, 289-300.

Grosz, B., Appelt, D., Archbold, A., Moore, R., Hendrix, G., Hobbs, J., Martin, P., Robinson, J., Sagalowicz, D., and Warren, P. TEAM: A transportable natural language system. Technical Note 263, SRI International, Menlo Park, California. April, 1982.

Harris, L. Experience with ROBOT in 12 commercial natural language database query applications. Proceedings of the sixth IJCAI. 1979.

Hendrix, G. and Lewis, W. Transportable natural language interfaces to databases. Proceedings of the 19th Annual Meeting of the ACL. 1981.

Kaplan, S. J. Cooperative responses from a portable natural language query system. Ph.D. Dissertation, University of Pennsylvania, Computer Science Department, 1979.

Konolige, K. A Framework for a portable NL interface to large databases. Technical Note 197, SRI International, Menlo Park, CA, October, 1979.

Ross, K. Parsing English phrase structure, Ph.D. Dissertation, Department of Linguistics, University of Massachusetts, 1981.

Ross, K. An improved left-corner parsing algorithm. Proceedings of COLING 82. 1982, 333-338.

Slocum, J. A practical comparison of parsing strategies, Proceedings of the 19th Annual Meeting of the ACL. 1981, 1-6.

Tennant, H. R. Evaluation of natural language processors. Ph.D. Dissertation Department of Computer Science, University of Illinois 1980.

Thompson, C. W. SURLY: A single user relational DBMS. Technical Report, Computer Science Department, University of Tennessee, Knoxville, 1979.

Ullman, J. Principles of Database Systems Computer Science Press, 1980.

Younger, D. Recognition and parsing of context-free language in time n3. Information and Control, 1967, 10, 189-208