SRI International

PRACTICAL NATURAL-LANGUAGE PROCESSING BY COMPUTER

Technical Note 251

October 1981

By: Robert C. Moore
Artificial Intelligence Center
Computer Science and Technology Division

SRI Project 1605

This paper is based on a presentation made at the Pergamon Infotech conference "Programming: New Directions," London, England, June 15-17, 1981.

Preparation of this paper was supported by the Defense Advanced Research Projects Agency under Contract N00039-80-C-0645 with the Naval Electronic Systems Command.

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

ABSTRACT

This paper describes the state of the art in practical computer systems for natural-language processing. We first consider why one would want to use natural language to communicate with computers at all, looking at both general issues and specific applications. Next we examine what it really means for a system to have a natural-language capability. This is followed by a discussion of some major limitations of current technology. The bulk of the paper is devoted to looking in detail at a single application of natural-language processing: database retrieval by natural-language query. We lay out an overall system architecture, explaining what types of processing and information are required. Then we look at two general classes of systems, special-purpose and general-purpose, explaining how they differ and their relative advantages and disadvantages. Afterwards we point out some remaining problems that will require additional basic research. Finally we conclude by discussing when language-processing technology at various levels of capability is likely to be commercially practical, and what it may cost to develop and use applications of that technology.

I INTRODUCTION

Is communication with computers in natural languages, such as English or French, a near-term, practical possibility? Computer professionals not directly involved with computational linguistics may well react to such a suggestion with skepticism. Since the 1950s there has been no shortage of academic entrepreneurs willing to proclaim that "machine translation" or "story understanding" is just around the corner, but the reality has always fallen short of the promise. Nonetheless, it is my belief that natural-language processing is now crossing the threshold from being simply a laboratory curiosity to functioning as a practical tool for man-machine communication.

Before making a "guided tour" of the possibilities and problems of natural-language processing by computer, we should first ask whether the trip is worth taking. Is there really anything to be gained by using natural language to talk to computers? I believe there are many reasons the answer to this question is "yes," but I will focus on one that to me seems particularly important--the need overcome the "computer mystique" that makes many potential computer users reluctant to take advantage of the possibilities open to them.

It has almost become conventional wisdom that we are living in the midst of an information revolution whose impact on society will be as great as the agricultural and industrial revolutions that preceded it. Whether or not this overstates the case, it is clear that the communication and processing of information play a far greater role in society than ever before, and that the use of computers is central to this phenomenon. Still, countless opportunities for effective application of computer technology are lost because of the high personal cost of learning how to make the computer do what one wants, or worse yet, fear that one will be defeated by the complexity of the task.

I believe that making it possible to interact with computers in natural language would overcome much of the resistance to their use. A formal language, no matter how well designed, will seem intimidating to many novices; a system that accepts natural language is much more likely to be perceived as "user-friendly." Moreover, even sophisticated computer users often avoid using a new system because learning to use it requires more effort than performing the task at hand one more time without it. It will always be neccessary for users to make an investment in learning what a system is capable of doing, but the use of natural language could minimize the cost of learning how to make the system do it.

I foresee at least two roles natural language could play in making systems easier to learn. First, it could enable casual users to make more sophisticated use of complex systems. At present, being a casual user of a system often means knowing only a few of the commands the system can accept and, hence, being able to utilize only a few of the system's capabilities. Permitting natural-language input would allow such a user access to more advanced capabilities of a system without his having to know the exact formats of commands. A second role for natural-language interaction would be to ease the transition from novice to expert, by having the system show the user the abbreviated form of the natural-language expressions he inputs.

To illustrate these possibilites, consider the problem of learning to use a sophisticated text editor, which may have dozens of commands expressed by one or two (not always mnemonic) characters. This makes staggering demands on a new user's memory. Imagine, though, a text editor with a special "natural-language mode" that would allow the user to say "go back to the previous paragraph," or "replace every occurrence of 'thus' on this page with 'therefore.'" An expert user would never want so verbose a form, but it would enable a novice to meet his immediate need without pouring over a manual wondering whether to look under "replace," "substitute," or "change." Furthermore, if the system not only executed the command but also displayed the abbreviated form, the novice would soon become an expert.

This example can obviously be generalized to many other types of systems. Any system that, like a text editor, has a complex command set and is used interactively is a plausible candidate. Electronic mail systems, file handlers, display managers, interactive query systems, and even the command level of an interactive operating system (on a time-shared or personal machine) could benefit from a natural-language input mode as described above. Of these applications, only natural-language querying of formatted databases has been extensively explored, but the capabilities required in the other areas mentioned should not be essentially different. A step beyond this would be an "intelligent tutor" that responds to direct questions about the use of a system, perhaps giving alternative ways of performing some task and explaining the trade-offs involved. The language-processing problems would be much the same as in the other applications, there being little difference between the command "Copy this file to my directory," and the question "How do I copy this file to my directory?"

II WHEN IS LANGUAGE NATURAL?

How great a range of language would a system really have to accept to achieve these ends? We have been using the term "natural language" as if it were an all-or-nothing phenomenon, but there are many levels of performance that might be described as providing a natural-language capability. It is clearly not sufficient to define what is sometimes called an "English-like language." This term usually refers to what is in reality a purely formal language dressed up in a syntax that looks like English. The point is that it does little good to coat the formal command "Print salary Jones" with syntactic sugar like "the" and "of," if the system cannnot also understand "What is Jones's salary?" "How much does Jones earn?" and many other forms in which the question may be phrased. If only one form is acceptable, there is just as much to learn as with any other formal language. The similarity to natural language may be of some mnemonic value, but that is all. On the other hand, if we insist that a system accept anything a person can understand before considering it usable, we still have a long time to wait.

What is needed is for the system to accept sufficiently many straightforward expressions of the user's intentions to avoid frustration. In practice this means understanding most inputs on the first try and a majority of the rest with one or two rephrasings. Experience with the LADDER database retrieval system developed at SRI International suggests that this is now feasible. Outside users of the system have quickly attained acceptance rates of 80% or better, with most of the rejections being on queries that exceed the capabilities of the "back-end" database system and that, consequently, the natural-language processor was never intended to handle.

III LIMITATIONS

We will shortly be looking in detail at how natural-language processing systems actually work, but two broad constraints on the current technology need to be mentioned first. One of these is that the domain of discourse--what is being talked about--must be restricted. It turns out that one must know far more than just the rules of language to interpret natural-language expressions; one must know a lot about the world as well. Frequently the rules of language will permit many syntactic interpretations, or "parsings," of a single sentence. Knowing what makes sense in terms of the domain being discussed is necessary to determine which interpretation is correct. One famous example of this is the sentence, "Time flies like an arrow." Most English speakers would perceive as the only interpretation of this sentence a simile about the passage of time, but there are other possible interpretations parallel to the assertion "Fruit flies like bananas," and the command "Time race cars like a stopwatch." We know that there are no such things as "time flies" and, even if there were, they could not eat arrows, and that arrows cannot be used to time anything, including flies. It is apparently this knowledge that makes us blind to the alternative interpretations.

It is no accident, therefore, that all the applications of natural-language processing mentioned above involve attempts by a person to get a particular computer system to do something, as this is an excellent way to limit what is talked about to manageable proportions. Unrestricted dialogue remains as elusive as ever. Even if we knew how to provide a computer with all the knowledge it needed in a form it could use (which we do not), the task of putting it all in would be so great as to be totally impractical. This fact bodes ill for a number of applications of natural-language processing that would be highly

desirable, including most of those that require dealing with pre-existing texts.

The most obvious application that falls into this class is automated translation from one natural language to another. This was the dream that gave computational linguistics its start over twenty years ago, but the extensive knowledge of the world that a competent translator must have remains a fundamental obstacle. There is some hope for useful results if the subject is sufficiently restricted--particularly to a narrow technical field--but a system that can translate wire-service news dispatches is as far away as ever.

The same problem plagues applications in the information sciences: information retrieval from text databases, document retrieval by content, and automatic abstracting. The sort of thorough syntactic and semantic analysis necessary for anything that could be called "text understanding" is simply impractical. More limited techniques such as keyword matching have some utility, but they are inevitably subject to gross errors. For example, in one case a system designed to scan the wire services for stories about natural disasters retrieved an item about Congressman Daniel Flood of Pennsylvania.*

The other major limitation of current technology is that natural-language input must be presented in computer-readable form; e.g., typed at a terminal. Speech input is not yet practical. It is possible to recognize isolated words or short phrases, chosen from a restricted vocabulary, but transcribing continuous speech--without distinct pauses between words--is much more difficult. There is at least one experimental system [1] that recognizes continuous speech in close to real time, but the speaker must confine himself to such a highly restricted subset of English that it can be considered, at best, an "English-like" language in the sense discussed in the preceding section. At the current state of the art, therefore, speech input in relatively unconstrained natural language is possible only if the speaker is willing to talk like

---

* Whether Congressman Flood objected to being classified as a natural disaster is unrecorded.

"What | is | the | salary | of | Jones?"

Computer recognition of fluent, continuous speech will require a much deeper understanding of the acoustics of speech than we currently possess.

IV DATABASE RETRIEVAL BY NATURAL-LANGUAGE QUERY

A. Simplifying Database Retrieval

The application of natural-language processing most extensively pursued to date has been interactive retrieval of information from formatted databases. A number of systems have been built for this purpose, but the most comprehensive is probably the LADDER system developed at the Artificial Intelligence Center of SRI International [2] [3]. This system and its descendants are the models on which this paper is based.

Database retrieval is well-suited to natural language. Restricting discourse to the information in a database creates the kind of closed domain that is required, and natural language is a much more convenient medium than a formal query language for expressing many types of requests. Suppose we have a corporate database containing a DEPARTMENT file, among whose fields are DEPT_NAME and MANAGER. If we want to know who manages the sales department, a typical database management system would require us to write a program:*

```
   OPEN FILE DEPARTMENT;
   FIND FIRST DEPARTMENT RECORD;
10 IF NO-RECORD GO TO 30;
   IF DEPARTMENT-DEPT_NAME NOT-EQUAL 'SALES' GO TO 20;
   PRINT DEPARTMENT-MANAGER;
20 FIND NEXT DEPARTMENT RECORD;
   GO TO 10;
30 CLOSE FILE DEPARTMENT;
   RETURN;
   END;
```

A nonprocedural, high-level query language would let us express this much more simply:

---

* This program is a simplified version of what would be required by the DBMS-20 database management system that runs on DECSYSTEM-20 computers.

```
FOR EACH X IN DEPARTMENT FILE
        WITH DEPT_NAME.X = 'SALES'
        RETURN MANAGER.X
```

but this is still a formal language whose conventions must be followed explicitly. It is far simpler to just ask: "Who manages the sales department?"

B. System Architecture

Figure 1 shows the overall structure of a hypothetical natural-language database query system. The major decomposition is between the linguistic system, whose task is to determine what information is requested, and the data access system, whose task is to determine how to obtain that information from the database (via a conventional database management system).

The linguistic system contains a language executive that builds a formal representation of the meaning of a natural-language query (a "semantic representation"). To do this, it draws upon three sources of linguistic knowledge--syntactic rules, semantic rules, and a lexicon--plus a model of the domain of discourse.

Syntactic (or grammatical) rules govern the way words combine into phrases and phrases into sentences. It is necessary to understand how a sentence is syntactically organized to interpret its meaning correctly. Consider the pair of sentences:

John is managing director.
John is managing directors.

In the first sentence, "managing director" is a single unit, which presumably refers to John's job title. In the second sentence, however, the rules of the language force us to group managing with "is" rather than "directors," so we get the interpretation that directors are what John is managing.

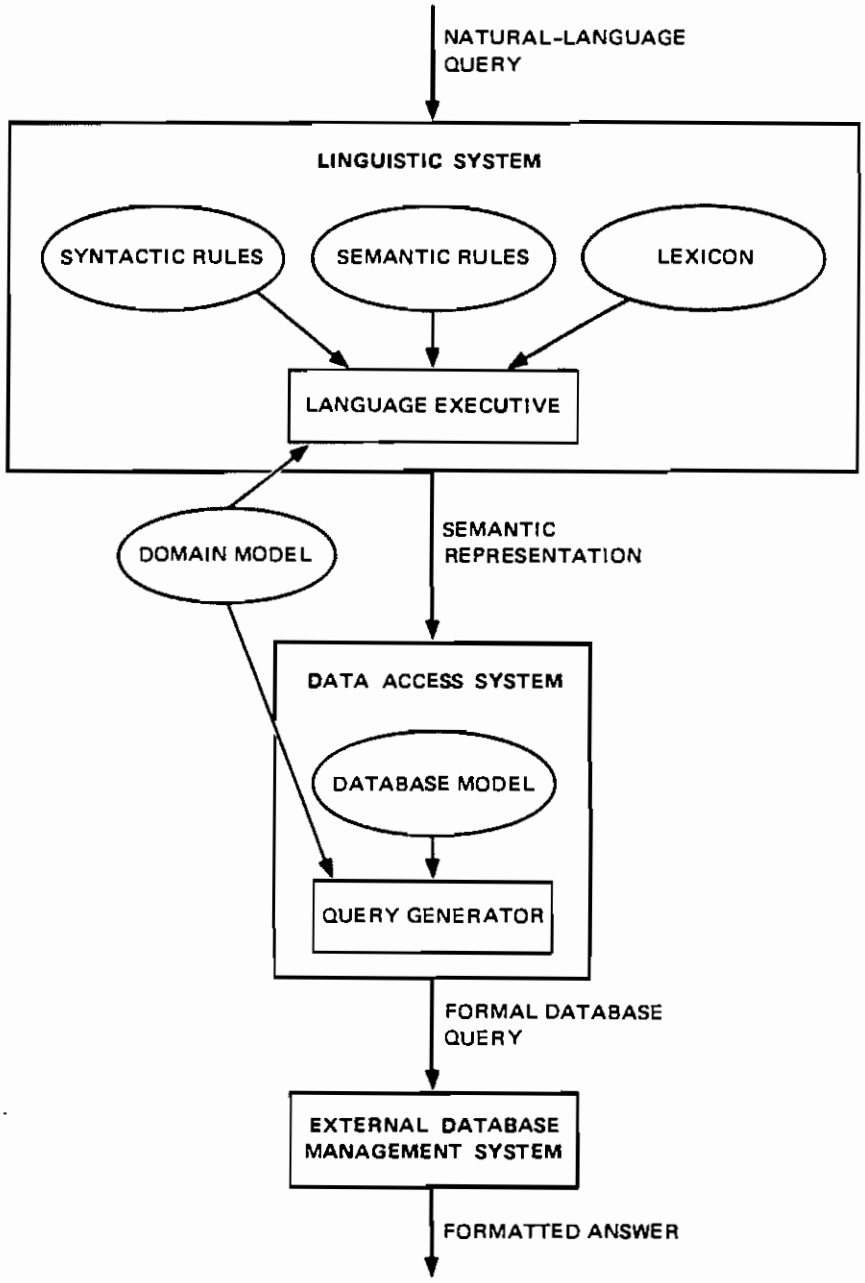

FIGURE 1 SYSTEM ARCHITECTURE

Semantic rules are used to map syntactic structures into formal representations of the semantic relationships they express. A representation of the semantic structure of a sentence is needed, in addition to a representation of the syntactic structure, because identical syntactic structures can encode different semantic relationships. Thus, the two sentences,

> John paid Mary for the book.
> John paid five dollars for the book.

have the same structure syntactically, but the first sentence says something very different about Mary than the second sentence says about five dollars.

The lexicon contains information about the syntactic and semantic roles played by particular words. For the words in the current example, the syntactic information would include the facts that "John" and "Mary" are proper nouns, that "book" is a common noun, and that "pay" is a verb that can take as many as two objects, plus a "for" prepositional phrase. The semantic information would specify that "pay" refers to a relation among a purchaser, a seller, a thing purchased, and an amount paid; and that if either the seller or the amount of money is omitted, the other can be the referent of the noun phrase immediately following the verb.

To construct the correct semantic representation, the linguistic system must also draw upon knowledge about objects, classes of objects, and relations in the domain of discourse.* This collection of information is often called "the domain model." In systems for database retrieval, the domain model is usually quite simple. It generally contains information about what classes specific objects belong to, what classes are subsets of a given class, and among what classes of objects a given relation may hold. For instance, for a system to interpret the examples involving "pay" properly, the domain model would have to include the information that the seller must be a person (actual or legal) and the amount paid must be an amount of money, plus the

---

* We saw this previously when we discussed the example "Time flies like an arrow."

information that Mary is a person and that five dollars is an amount of money. Without this information, it is impossible for the system to choose between the alternative interpretations the syntactic forms permit.

The data access system generates formal database queries from semantic representations of natural-language queries. To do this, it needs a model of the way in which information about the objects, classes, and relations included in the domain model is represented in the database. We will call this information the database model. In addition, the data access system needs information from the domain model itself, because the files and fields in the database may not correspond directly to the structure of objects, classes, and relations mentioned in the semantic representation of the query. For instance, if information about salaried and hourly employees is kept in different files, the system will be unable to answer a query about programmers or secretaries unless it knows whether they are salaried or hourly. This might not be explicit in the semantic representation of the query, but it should be stored in the domain model.

This last point turns out to be more important than it might first appear. A database can be thought of as embodying a particular view of part of the world. That view is implicit in the way the information in the database is organized into files, records, and fields. The user, on the other hand, may have a view of the world that diverges considerably from the view implicit in the database. For instance, if information about salaried and hourly employees is stored in separate files, the salaried/hourly distinction will seem to be a major division between parts of the world. In a sense, the database management system will be unable to "think about" employees without thinking of them as either salaried or hourly. To a user, however, this distinction may be of no consequence at all. The data access component must compensate for such differences in world view between the user and the system. In this case it is relatively simple to divide what the user views as a single class of entities into the two subclasses that the database recognizes; the

kind of information represented in the domain model is sufficient for that. We will see later, though, that the reconciling of differences in world view can be much more difficult.

C. Types of Systems: Special-Purpose and General-Purpose

The system architecture just presented provides an overall framework for database retrieval systems, but within that framework systems may differ in many ways. One of the most important differences among systems is whether they are designed for a specific domain and a specific database or for essentially general-purpose application. If a general-purpose system is desired, the linguistic component of the system should obviously be as domain-independent as possible. On the other hand, if the system is intended to accept queries about one domain only and retrieve information from a single database, many shortcuts can be taken to achieve higher performance at lower cost. In this section we will examine the differences between special-pupose and general-purpose systems, comparing their relative advantages and disadvantages.

1. Special-Purpose Systems

To date, almost all the successful demonstrations of natural-language access to databases have employed special-purpose systems. These systems achieve high performance by encoding directly in their syntactic and semantic rules much of the information that would be in the domain model and database model in a more general system. Let us see how such a system might answer the query, "Who manages the sales department?" given a database containing a DEPARTMENT file with DEPT_NAME and MANAGER fields.

The syntactic and semantic rules of the system will be combined into a single set of rules, which we will call simply "the grammar." The part of the grammar and lexicon needed to handle "Who manages the sales department?" might be as shown in Figure 2. The symbols in angle brackets represent syntactic categories. The domain-specific character of the system is evident from the fact that

```
<SENTENCE> => <PRESENT> <ATTRIBUTE> <DEPARTMENT>;
              (DB (SUBST GENVAR '*'
                         ('<DEPARTMENT> <ATTRIBUTE>')))

 <PRESENT> => who

<ATTRIBUTE> => <ATTRNAME>;
               'RETURN <ATTRNAME>.*'

<DEPARTMENT> => the <DEPNAME> department;
                'FOR EACH * IN DEPARTMENT FILE
                          WITH DEPT_NAME.* = '<DEPNAME>''

    manages -- <ATTRNAME>; 'MANAGER'

      sales -- <DEPNAME>; 'SALES'
```

Figure 2 Grammar and Lexicon for a Special-Purpose System

<DEPARTMENT> is a syntactic category, but "noun phrase" and "verb phrase" are not. Each grammar rule specifies for some syntactic category one of the linguistic patterns that belong to the category and the semantic interpretation of that pattern as a member of the category. Each lexical entry specifies for some content word the syntactic category it belongs to and its semantic interpretation as a member of that category. In each case, the two kinds of information-- syntactic and semantic--are separated by a semicolon.

The syntactic component of a grammar rule consists simply of a sequence of syntactic category symbols and words that the phrase to which the rule is applied must match. Figure 3 presents the syntactic analysis of "Who manages the sales department?" according to this grammar and lexicon. The sentence is decomposed into three phrases in the categories <PRESENT>, <ATTRIBUTE>, and <DEPARTMENT>. The <PRESENT> phrase consists of the word "who." The <ATTRIBUTE> phrase is further reduced to <ATTRNAME> which, according to the lexicon, is the category of the word "manages." The <DEPARTMENT> phrase is formed from the word "the," a <DEPNAME>, and the word "department." The <DEPNAME> in this case must be the word "sales," which is also consistent with the lexicon.

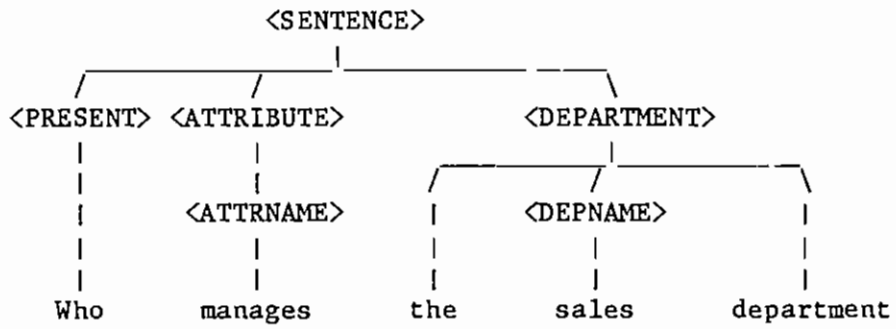

Figure 3 Syntactic Analysis in a Special-Purpose System

This analysis imposes relatively little syntactic structure on the sentence, and what structure it does impose has much more to do with the domain and the database than with standard linguistic concepts. The degree to which the structure of the database has shaped the grammar should be noted. If Smith manages the sales department, that seems to be just as much a fact about Smith as about the sales department. The grammar, however, treats a manager as an attribute of a department rather than vice versa, because that is how the database is organized. If, instead of a DEPARTMENT file, the database had a MANAGER file with a DEPARTMENT field, the syntactic analysis of the sentence might be quite different.

The semantic components of our grammar rules consist of (1) fragments of formal database query language (enclosed by single quotes) into which the semantic interpretations of subphrases (indicated by syntactic category symbols) are to be substituted, and (2) procedure calls to manipulate these fragments. Figure 4 shows the derivation of the semantic analysis for the current example. This is much more complex than the syntactic analysis and is really "where the action is" in this type of system. The semantic interpretation of "manages" as an <ATTRNAME> is the string 'MANAGER', which on reinterpretation as an <ATTRIBUTE> is incorporated in the string 'RETURN MANAGER.*'. As a <DEPNAME>, "sales" is interpreted as the string 'SALES', which, when analyzed as part of a <DEPARTMENT> phrase, is incorporated in the string

```
'FOR EACH * IN DEPARTMENT FILE
        WITH DEPT_NAME.* = 'SALES''.
```

The semantic rule for <SENTENCE> concatenates this with 'RETURN MANAGER.*', invokes the procedure GENVAR to produce a new variable name (say 'X'), and calls SUBST to substitute 'X' for the dummy variable '*'. This produces

```
'FOR EACH X IN DEPARTMENT FILE
        WITH DEPT_NAME.X = 'SALES'
        RETURN MANAGER.X'          ,
```

which is handed off to the database management system by the procedure DB.

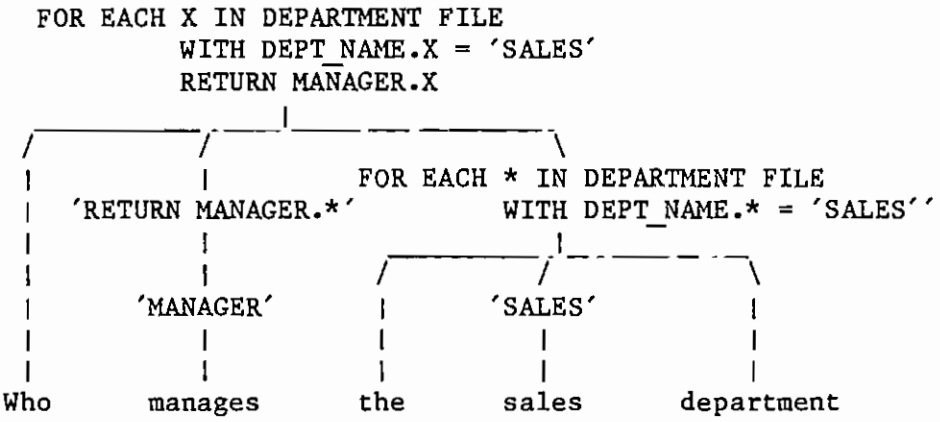

Figure 4 Semantic Analysis in a Special-Purpose System

Looking back at Figure 1, we can see that this kind of special-purpose system does not really exhibit the clear separation of functions and knowledge laid out in our system architecture. The domain model has been incorporated into the grammar and lexicon by having things like <DEPARTMENT> and <ATTRIBUTE> as syntactic categories; the database model has been incorporated into the semantic rules by making them build and execute formal database queries. This renders the data access system almost empty, except perhaps for a translator to compile

the nonprocedural database query produced by the linguistic system into a procedural query to be executed by the external database management system.

Special-purpose systems of this kind have both advantages and disadvantages. On the positive side, they can be finely tuned to a specific application and a specific database. The particular linguistic expressions the user wants can readily be incorporated without a comprehensive analysis of their syntactic and semantic properties. Besides, these systems usually require fewer computational resources than more general systems, as they do not "waste time" looking for interpretations of a query that cannot be answered because the database lacks the required information.

On the negative side, this type of system suffers from uneven syntactic coverage. Since standard linguistic categories are not represented in the grammar, the linguistic generalizations that depend on them are not captured. In grammars that use the concepts "verb phrase" and "noun phrase," the relationship between active and passive sentences ("Smith manages the sales department." / "The sales department is managed by Smith.") is usually encoded in just a few rules. In the special-purpose systems we have been discussing, however, there may be many rules in which the active/passive distinction shows its effects. Inevitably some will be missed.

A second problem with these systems is that, by consolidating what were originally different levels of representation and different knowledge sources in our system architecture, they run roughshod over the distinctions these were introduced to preserve. Two natural-language queries with the same semantic structure may correspond to different types of database queries. In these systems, the queries would be forced to have different semantic representations. Similarly, we noted that two sentences with the same syntactic structure can have different semantic structures--but since each syntactic rule has a single corresponding semantic rule, the sentences must be treated as if they were <u>different</u> syntactically in order to treat them as different

semantically. Because all lower level distinctions must be reflected at higher levels, there can be a veritable explosion in the number of grammar rules that have to be produced.

A related point is that, since the semantic rules manipulate only fragments of database queries, the system behaves as if the user is actually talking about the database--files, records, and fields--rather than its subject--departments and managers. Thus, the system makes no distinction between ascertaining what information the user wants and determining how to get it from the database. The system must interpret the user's queries as if there were no difference between his view of the world and that of the database.

Finally, the most serious disadvantage of special-purpose systems is they are not transportable to new applications. Extensive reprogramming may be required to change domains, or even to change databases within the same domain. Furthermore, this is programming that requires an unusual combination of skills. A person needs considerable knowledge of both natural-language syntax and semantics, in addition to programming, to produce a system that performs satisfactorily. Currently such people can be found only in a handful of research laboratories.

2. General-Purpose Systems

The most serious disadvantages of special-purpose systems can be overcome by adhering more closely to the system architecture described in Section IV-B, with its domain-independent syntactic and semantic rules. Using such a system, we could change domains by changing only the lexicon, domain model, and database model. Since this information is much simpler in form than the syntactic and semantic rules, it can be put into the system by means of "canned" acquisition procedures. This would virtually eliminate the need for reprogramming to change domains or databases. As yet, no system offering this degree of flexibility has been developed to the point of being a practical tool, but one is currently being implemented at the SRI Artificial

Intelligence Center. We expect that within two to four years this system (called TEAM) will reach the stage of development currently exhibited by special-purpose systems.

In a general-purpose system the question, "Who manages the sales department?" would receive a syntactic analysis similar to that shown in Figure 5. The details need not concern us, but it should be noted that the syntactic categories are abstract, domain-independent concepts like VP (verb phrase), NP (noun phrase), V (verb) and N (noun). Thus, the same syntactic rules used to analyze this sentence would be able to parse "Who commands the U.S. ship?" or "What determines the blood count?" A change of domains need not require any changes in the grammar.

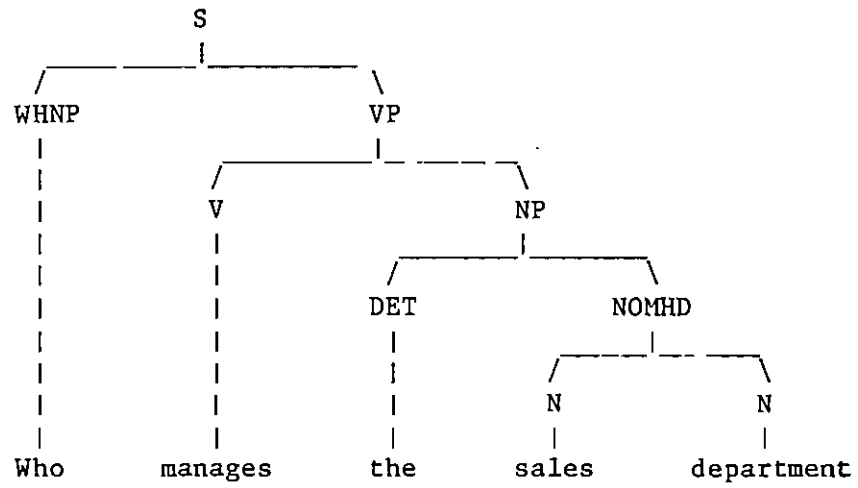

Figure 5 Syntactic Analysis in a General-Purpose System

The semantic representation for "Who manages the sales department?" might look like

```
(WHAT X (PERSON X)
        (MANAGE X (THE Y (DEPARTMENT Y)
                         (NAME-OF Y SALES))))
```

(This can be read as "What person X is such that X manages the department Y, such that the name of Y is sales?") The symbols PERSON,

MANAGE, DEPARTMENT, NAME-OF, and SALES would represent objects, classes, and relations in the domain model, not files or fields in the database. MANAGE is treated here simply as a relation between a PERSON and a DEPARTMENT. There is no presumption either that a manager is an attribute of a department or that a department is an attribute of a manager; whichever way the database is set up can be accommodated equally well.

All that remains at this point is to translate the semantic representation of the user's query into an equivalent formulation in terms of files and fields in the database. For the current example, this is quite straightforward. In the database model it would be recorded that X manages Y, just in case there is a record in the DEPARTMENT file, where the MANAGER field is the name of X and the DEPT_NAME field is the name of Y. This can be represented in a simple data structure such as

```
(MANAGE
   (FILE DEPARTMENT)
   (FIELDS
      (ARG1 MANAGER)
      (ARG2 DEPT_NAME)))
```

This would mean that information about the relation MANAGE is stored in the DEPARTMENT file, with the first argument of MANAGE stored in the MANAGER field and the second argument stored in the DEPT_NAME field.

A simple data structure like this works only as long as the relation the user talks about is represented directly by some subset of the fields of a file (a projection of the file, to use database jargon), but straightforward extensions will handle other cases as well. For example, if a programmer is an employee whose job code is 'PROG', this could be expressed by putting a restriction on the FILE attribute in the data structure:

```
(PROGRAMMER
   (FILE EMPLOYEE (JOB_CODE = 'PROG'))
   (FIELDS (ARG1 EMP_NAME)))
```

In this case, programmers are represented by records in the EMPLOYEE file, where the value of the JOB_CODE field is 'PROG', and the field that refers to the programmer himself is EMP_NAME.

We could go on elaborating ways of handling various types of mappings between the domain model and the database, but it should suffice to say that any mapping can be handled that permits a domain model concept to be defined in terms of concepts explicitly represented in the database. For instance, if we have a database that includes data both on occupations and family relationships, we could define the concept of PROUD_PARENT to be the MOTHER or FATHER of at at least one PERSON whose OCCUPATION is either 'DOCTOR' or 'LAWYER'.

A general-purpose system of this type, therefore, addresses all the major deficiencies of the special-purpose systems. If it has a good, general grammar of the language, it will have much better coverage of syntactic patterns. By allowing syntactic structures, semantic structures, and formal database queries to be more independent, it avoids unecessary proliferation of syntactic and semantic rules, and it makes differences between the user's and the database's views of the world easier to deal with. Finally, and perhaps most importantly, it greatly facilitates transportability by isolating domain- and database-dependent information in the domain model, database model, and lexicon.

These benefits are not attained without cost, however. Because systems of this type will be more general, they will also be larger and slower than special-purpose systems. The grammar will have to allow for syntactic constructions that may not arise in talking about a particular domain. The extra grammar rules will take up memory space and consume execution time in looking for syntactic patterns that do not occur. Furthermore, in special-purpose systems a lot of "semantic filtering" is done by the syntactic rules, while in general-purpose systems this process is delayed--another cause of longer execution times. A second cost of generality is reduced flexibility. The grammar rules tend to be much less interdependent in special-purpose systems. If a new linguistic pattern is introduced, there will typically be fewer

interactions with other rules to take into account than in a tightly structured general-purpose grammar. Adding new linguistic forms to a general-purpose system requires much more careful syntactic and semantic analysis.

3. Long-Term Research

Although the kind of general-purpose system just described promises to be a significant improvement over existing special-purpose systems, there are important problems it leaves unresolved. Two of these deserve closer examination. The first is how to allow more complex relationships between the domain model and the database. As we pointed out, the simple sort of database model we described requires that all domain concepts be definable in terms of concepts explicitly represented in the database. From the user's point of view, however, it makes much more sense to define the meaning of files and fields in the database in terms of domain-model (i.e., his) concepts.

This is especially clear in the case of what we might call "feature fields." These are fields representing some complex property, with values that simply mean "true" or "false." For example, in one database we have had to deal with at SRI, there is a file SHIP with a field DOCTOR whose value is 'D' if the ship in question has a doctor on board. The problem is that, if the fundamental, or "atomic" concepts in the domain model correspond roughly to single natural-language words, then "having a doctor on board" will not be one of them. Rather, it will be a complex property involving the class DOCTOR and the relation ON-BOARD. The semantic representation of this property might be something like

```
X such that
   (SOME Y (DOCTOR Y)
           (ON-BOARD Y X))
```

(to be read as, "X such that there is some doctor Y such that Y is on board X.")

We have been assuming that information in the database model is attached to individual atomic concepts (recall MANAGE and PROGRAMMER). In this case, however, we would have to attach the information to a complex expression. Even this would not completely solve the problem. Suppose we also had a NURSE field which means "has a nurse on board." Whether or not a ship possesses the property "has a doctor or nurse on board" would then be derivable from the database. However, if information in the database model is attached only to "has a doctor on board" and "has a nurse on board," then to make the connection the system would have to recognize the logical equivalence of

```
X such that
   (SOME Y (OR (DOCTOR Y)
               (NURSE Y))
           (ON-BOARD Y X))
```

("X such that there is some doctor or nurse Y such that Y is on board X.") and

```
X such that
   (OR (SOME Y (DOCTOR Y)
               (ON-BOARD Y X))
       (SOME Z (NURSE Z)
               (ON-BOARD Z X)))
```

("X such that either there is some doctor Y such that Y is on board X, or there is some nurse Z such that Z is on board X.") The point is that the latter expression explicitly contains representations of "has a doctor on board" and "has a nurse on board," whereas the former does not. To be able to answer the question, the system must reformulate it in terms of concepts that it knows how to connect to the database.

It should be increasingly clear that this is an open-ended problem. We are asking the system not only to retrieve information that is explicitly stored in the database, but also to tell us whatever can be inferred from that information. On the basis of knowing about "doctors on board" and "nurses on board," we might like the system to give us at least partial answers to questions about "surgeons on board"

or "medical personnel on board," or even "doctors within 100 miles" (by inferring that, if a ship with a doctor on board is within 100 miles, so is the doctor himself).

For a system to answer questions such as these, it needs more than the simple sort of data access system we have been imagining; it needs a full-fledged deductive-reasoning system. Progress has been made on this problem in artificial-intelligence research laboratories [4], but practical systems of any substantial generality are still a long way off.

The second major problem left untouched by systems currently operational or under development is that, although they can engage in repeated exchanges of questions and answers, they cannot conduct real conversations. These systems treat each question as an isolated event; no attempt is made to build up a context of information as to what is being talked about, what the user knows, or what he is trying to do. At times this has almost comical results. Suppose that a company has many employees named Jones, but only one who is a programmer. If a user asks, "Who are the programmers?" and gets back the list "Jones, Smith, and Johnson," and then follows this exchange with "How much does Jones earn?" he is likely to get a response such as "There are 23 employees named Jones; which one do you mean?" In context, it is clear that the user is talking about Jones the programmer, but the system has no notion that the "conversation" is about programmers.

While the more obvious failures, as in this example, may be avoided by simple heuristics (such as checking the preceding answer), in general the effects of conversational context on interpretation can be much more subtle. The correct interpretation of what the user says often depends on information derivable from the preceding discourse as to how much he knows and what he is trying to do. For instance the question, "Is someone assigned to every task?" is ambiguous; it could be asking whether any single person is assigned to all tasks, or whether each task has at least one person assigned to it. If the question is asked immediately after "Is anyone assigned to more than one task?" has

been answered negatively, however, the first interpretation is extremely unlikely. We would like a system to be able to infer that the user knows that no person is assigned to more than one task; so no person is assigned to all tasks. If the system then assumes that the user is more likely to be asking a question that he does not know the answer to, it will arrive at the correct interpretation. Knowing a person's intentions can also aid in interpretation. If a user asks a travel information system "Is Rome closer to London than Madrid?", he could mean "Is Rome closer to London than to Madrid?" or "Is Rome closer to London than Madrid is?" If he could preface question with the explanation, "I want to vacation in either Italy or Spain after I finish my business in England," and the system understood what that implied, it would be clear that he meant the latter.

To handle problems like this, a system needs an additional model--a model of the conversation. The model would have to include what the speaker is talking about, what he knows, and what he wants. How to infer this information from the conversation, how to represent it in a model, and how to reason with it and use it to interpret discourse are all being actively studied in computational linguistics and artificial intelligence [5] [6] [7]. Possible solutions to these problems are beginning to emerge, but much work remains to be done.

V Prospects for Practical Systems

While the details of technical problems and their solutions are interesting in their own right, in a discussion of practical natural-language processing the final question has to be, when will this technology actually be available and what will it cost? For special-purpose systems, the answer to the availability question is essentially "now." The technology involved can be had "off the shelf" from several research laboratories; all that is really necessary is a serious engineering effort to transform it into a marketable package. At least one system, with somewhat limited capabilities, is already being offered as a commercial product (INTELLECT from Artificial Intelligence Corp.).

The costs of using such a system must include both the effort to develop a particular application and the investment in computational resources needed to support it. For special purpose-systems the development costs are rather high, because a new grammar has to be written for every application. Based on the experience with LADDER, six man-months to two man-years seems to be a reasonable estimate, depending on the experience of the implementers and the scope of the application. Interfacing to a database with ten files and one hundred fields would be considered a moderately large effort by these standards.

As regards computational costs, the LADDER system currently runs on a time-shared DECSYSTEM-2060 computer with six megabytes of physical memory, and one megabyte of virtual address space. On this system LADDER takes one to two seconds of machine time to produce a formal database query from an English input. (The database management system itself is accessed on another machine over a network.) The system takes up almost the entire address space of the machine, so a significantly larger system would have to be divided into several processes or implemented on a machine with a larger virtual memory. The DECSYSTEM-

2060 is an approximately $1-million facility, but it appears that smaller machines will soon be available that would support one to four users of a LADDER-like system for about $50,000. In the very near future, consequently, the cost of supplying one user with enough computational power to support this type of system should be about the same as the present cost of a word-processing workstation.

It will probably be two to four years before general-purpose systems of the sort that seem currently feasible are ready for commercial development. When they do become available, the effort required to create a new application system should be dramatically shortened--on the order of man-weeks to man-months, rather than man-months to man-years. The major uncertainty surrounding this estimate is whether the automatic acquisition routines will be sufficient to create a satisfactory system, or whether additional cutomizing will be necessary. The computational resources required will be significantly greater than with special-purpose systems. Current prototypes run three to five times slower than comparable special-purpose systems and use at least twice as much memory. It should be kept in mind, however, that as hardware costs continue to decline this factor will become less significant.

Finally, we can only speculate about the emergence of systems that have some of the capabilities described as long-term research problems. It seems safe to say it will be at least ten years before those problems are solved for practical applications. If such systems are eventually developed, they should provide the user with a much more accommodating environment, and their need for customizing by an expert programmer should asymptotically approach zero. The computational resources required, however, will probably be staggering by today's standards, so we must hope that the present trend of diminishing hardware costs continues. The computation needed to reason explicitly about how to access a database, as well as about what the user knows and wants to do, will probably far outstrip the computation needed for more narrowly defined linguistic processing. But this should not be surprising; people also find it much easier to talk than to think.

REFERENCES

1. B. T. Lowerre and D. R. Reddy, "The Harpy Speech Understanding System," in Trends in Speech Recognition, W. E. Lea, ed., Chapter 15 (Prentice-Hall, Englewood Cliffs, New Jersey, 1979).

2. G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a Natural Language Interface to Complex Data," ACM Transactions on Database Systems, Vol. 3, No. 2 (June 1978).

3. G. G. Hendrix, "The LIFER Manual: A Guide to Building Practical Natural Language Interfaces," SRI Artificial Intelligence Center Technical Note 138, Stanford Research Institute, Menlo Park, California (February 1977).

4. R. C. Moore, "Automatic Deduction for Commonsense Reasoning: An Overview," SRI Artificial Intelligence Center Technical Note 239, SRI International, Menlo Park, California (April 1981).

5. B. Grosz, "Focusing and Description in Natural Language Dialogues," in Elements of Discourse Understanding: Proceedings of a Workshop on Computational Aspects of Linguistic Structure and Discourse Setting, A. K. Joshi, I. Sag, and B. L. Webber, eds. (Cambridge University Press, Cambridge, England, 1981).

6. R. C. Moore, "Reasoning About Knowledge and Action," SRI Artificial Intelligence Center Technical Note 191, SRI International, Menlo Park, California (October 1980).

7. J. Allen and C. R. Perrault, "Participating in Dialogues: Understanding Via Plan Deduction," Proceedings, Second National Conference, Canadian Society for Computational Studies of Intelligence, Toronto, Canada, pp. 214-223 (19-21 July 1978).