Developing a Natural Language Interface to Complex Data

GARY G. HENDRIX, EARL D. SACERDOTI, DANIEL SAGALOWICZ, AND JONATHAN SLOCUM
SRI International

Aspects of an intelligent interface that provides natural language access to a large body of data distributed over a computer network are described. The overall system architecture is presented, showing how a user is buffered from the actual database management systems (DBMSs) by three layers of insulating components. These layers operate in series to convert natural language queries into calls to DBMSs at remote sites. Attention is then focused on the first of the insulating components, the natural language system. A pragmatic approach to language access that has proved useful for building interfaces to databases is described and illustrated by examples. Special language features that increase system usability, such as spelling correction, processing of incomplete inputs, and run-time system personalization, are also discussed. The language system is contrasted with other work in applied natural language processing, and the system's limitations are analyzed.

Key Words and Phrases: natural language, intelligent interface, database access, semantic grammar, human engineering, run-time personalization
CR Categories: 3.42, 3.62, 3.79, 4.22, 4.33

1. INTRODUCTION

In dealing with a very large database (VLDB), which is perhaps distributed among multiple computers with different database management systems (DBMSs) on remote sites, a central problem faced by would-be users is that of formulating queries in terms communicable to the system.

It is usually the case that business executives, government officials, and other decision makers have a good idea of the kind of information residing in their databases. Yet to obtain the answer to a particular question, they generally need to employ the services of a technician who works with the database on a regular basis

The work reported herein, other than the development of the LIFER system, was supported by the Advanced Research Projects Agency of the Department of Defense under Contract DAAG29-76-C-0012 with the U.S. Army Research Office. Development of LIFER was conducted under SRI's Internal Research and Development Program.
A version of this paper was presented at the Third International Conference on Very Large Data Bases, Tokyo, Japan, October 1977.
Authors' address: Artificial Intelligence Center, SRI International, Menlo Park, CA 94025

and who is thoroughly familiar with its file structure, the DBMSs on which it resides, how it is distributed among various computer systems, the coded field names for the data items, the kinds of values that different fields are expected to contain, and other idiosyncrasies.

The technician must understand the decision maker's question, reformulate it in terms of the data that is actually stored, plan a sequence of requests for particular items from particular files on particular computers, open connections with remote sites, build programs to query the remote systems using the primitives of the DBMSs of the remote systems, monitor the execution of those programs, recover from errors, and correlate the results. This is a demanding, time-consuming, and exacting task requiring much attention to detail. Escalated levels of sophistication are needed as the VLDB increases in size and complexity and as it is distributed over a wider range of host computers.

With the goal of making large, distributed databases directly available to decision makers (while freeing technicians from increasingly tedious details), a group of researchers at SRI International has developed a prototype system that, for many classes of questions, automates the procedures usually performed by technicians. This paper presents an overview of this system, called LADDER (for language access to distributed data with error recovery) [16],[1] and then concentrates on the particular problem of translating user queries from English into the terms of the database. The other aspects of the LADDER system are presented in greater detail elsewhere in the literature [13, 16, 17]. The system was developed as a management aid to Navy decision makers, so examples throughout the paper will be drawn from the domain of Navy command and control.

2. SYSTEM ARCHITECTURE

The running demonstration system consists of three major components that provide levels of buffering of the user from the underlying DBMSs. The LADDER user can think he is retrieving information from a "general information base" rather than retrieving specific items of data from a set of highly formatted, traditional databases that are scattered across a computer network. The user provides a question about the information base in English; LADDER applies all the necessary information concerning the vocabulary and syntax of the question, the names of specific fields, how they are formatted, how they are structured into files, and even where the files are physically located, to provide an answer.

LADDER's first component, called INLAND (for informal natural language access to Navy data), accepts questions in a restricted subset of natural language and produces a query or sequence of queries to the VLDB as a whole. The queries to the VLDB, as produced by INLAND, refer to specific fields, but make no commitment about how the information in the database is broken down into files.

For example, INLAND translates the question "What is the length of the Kennedy?" into the query

((NAM EQ JOHN#F.KENNEDY) (? LENGTH)),

where LENGTH is the name of the length field, NAM the name of the ship name

[1] A glossary of system names precedes the Appendix.

field, and JOHN#F.KENNEDY the value of the NAM field for the record concerned with the Kennedy.

Queries from INLAND are directed to the second component of LADDER, called IDA (for intelligent data access) [17]. In general, a query to IDA is a command list of constraints (such as (NAM EQ JOHN#F.KENNEDY) or (* MAX LENGTH)) and requests for values of fields (such as (? LENGTH)). INLAND operates by building a (possibly null) fragment of a query to IDA for each lower-level syntactic unit in the English input. These fragments are combined as higher-level syntactic units are recognized. At the sentence level, the combined fragments are sent as a command string to IDA.

Employing a model of the structure of the VLDB, IDA breaks down a query against the entire VLDB into a sequence of queries against individual files. Linkages among the records retrieved are preserved so that appropriate answers to the overall query may be composed and returned.

For example, suppose that the database consists of a single file whose records contain the fields

(NAM CLASS LENGTH).

Then, to answer the database query issued above, IDA can simply create one file retrieval program that says, in essence, "For the ship record with NAM equal JOHN#F.KENNEDY, return the value of the LENGTH field." Suppose, however, that the database is structured in two files, as follows:

SHIP: (NAM CLASS . . .)
CLASS: (CLASSNAME LENGTH . . .).

In this case the single query about the Kennedy's length must be broken into two file queries. These would say, first, "Obtain the value of the CLASS field for the SHIP record with NAM equal JOHN#F.KENNEDY." Then, "Find the corresponding CLASS record and return the value of the LENGTH field from that record."[2] Finally, IDA would compose an answer that is relevant to the user's original query (i.e. it will return NAM and LENGTH data, suppressing the CLASS-to-CLASSNAME link).

In addition to planning the correct sequence of file queries, IDA must actually compose those queries in the language of the remote DBMSs. Currently the system accesses, on a number of different machines, a DBMS called the Datacomputer [4, 6], whose input language is called DATALANGUAGE. IDA creates the relevant DATALANGUAGE query by inserting field and file names into prestored templates. However, since the database is distributed over several machines, the DATALANGUAGE that IDA produces does not refer to specific files in specific directories on specific machines. It refers instead to *generic files*, files containing a specific kind of record. For example, the queries discussed above might refer to the SHIP file rather than file SHIP.ACTIVE in directory NAVY on machine CCA-2.

It is the function of the third major component of LADDER to find the location of the generic files and manage the access to them. To carry out this function, the third component, called FAM (for file access manager) [13], relies on a locally

[2] If it is possible to perform multiple file accesses with a single multifile query, IDA will do so.

stored model showing where files are located throughout the distributed database. When it receives a query expressed in generic DATALANGUAGE, it searches its model for the primary location of the file (or files) to which it refers. It then establishes connections over the ARPANET to the appropriate computers, logs in, opens the files, and transmits the DATALANGUAGE query, as amended to refer to the specific files that are being accessed. If at any time the remote computer crashes, the file becomes inaccessible, or the network connection fails, FAM can recover, and if a backup file is mentioned in FAM's model of file locations, it can establish a connection to a backup site and retransmit the query.

The existing system, written in INTERLISP [19], can process a fairly wide range of questions against a database consisting of some 14 files containing about 100 fields. Processing a typical question takes less than a second of CPU time on a DEC KL-10 computer. An annotated transcript of a sample session with the system is provided in the Appendix.

We emphasize that the three major components of LADDER each address separate portions of the data access problem. Although they have been designed to work in combination, each component is a separate, self-contained module that independently addresses one aspect of data access. For example, the virtual view of the data that IDA supports for its caller would be of value even without a natural language front end. Likewise, the general technology developed for natural language translation may be separated from the data access problem and applied in other domains.

3. THE NATURAL LANGUAGE COMPONENT

With the goal of supplying natural language interfaces to a variety of computer software, we have developed a language processing package, called LIFER (for language interface facility with ellipsis and recursion) [11], that facilitates the construction and run-time operation of special purpose, applications-oriented, natural language interfaces. INLAND, the linguistic component of our intelligent interface to distributed data, has been constructed within the LIFER framework. Figure 1 gives some indication of the diversity of language accepted by this system. Below we describe the nature of INLAND and illustrate how it was created using LIFER's interactive language definition facilities. Of course, the examples can show only limited aspects of INLAND. We believe the existing INLAND system to be one of the most robust computerized natural language systems ever developed, accepting a wide range of questions about information in the database (as shown in Figure 1) as well as metaquestions about definitions of database fields and the grammar itself.

3.1 Overview of LIFER

Although work in artificial intelligence and computational linguistics has not yet developed a general approach to the problems of understanding English and other natural languages, mechanisms do exist for dealing with major fragments of language pertinent to particular application areas. The idea behind LIFER is to adapt existing computational linguistic technology to practical applications while investigating and extending the human engineering aspects of the technology. The LIFER system supplies basic parsing procedures and an interactive methodology needed by a system developer to create convenient interfaces (such as INLAND) in reasonable amounts of time. Certain user-oriented features, such as spelling correction, processing of incomplete inputs, and the ability of the run-time user to extend the language accepted by the system through the use of paraphrase, are also included in the LIFER package.

LIFER is composed of two basic parts: a set of interactive language specification functions, and a parser. The language specification functions are used to define an application language, a subset of a natural language (e.g. English) that is appropriate for interacting with existing software, such as a DBMS. Using this language specification, the LIFER parser interprets natural language inputs, translating them into appropriate interactions with the application software.

Figure 2 shows simplified example interactions with the LIFER parser using the INLAND language specification. A sequence of complete examples is presented in the Appendix. The user of the system types in a question or command in ordinary English, followed by a carriage return. The LIFER parser then begins processing the input. When analysis is complete, the system types "PARSED!" and invokes database functions (IDA) to respond.

An important feature of the parser is an ability to process elliptical (incomplete) inputs. Thus if the system is asked, as in question 1 of Figure 2,

WHAT IS THE LENGTH OF THE CONSTELLATION,

then the subsequent input

OF THE NAUTILUS

will be interpreted as WHAT IS THE LENGTH OF THE NAUTILUS.

If a user misspells a word, LIFER attempts to correct the error, using the INTERLISP spelling corrector [19]. If the parser cannot account for an input in terms of the application language definition, error messages, such as that produced after question 6, are printed that indicate how much of the input was understood and that suggest means of completing the input.

Provision is included in INLAND for interfacing with LIFER's own language specification functions, making it possible for users to give natural language commands for extending the language itself. In particular, computer-naive users may extend the language accepted by the system by employing easy-to-understand notions such as synonyms and paraphrases. This is illustrated by interactions 7 and 10 in Figure 2.

In using LIFER to define a language for INLAND, we have followed the approach taken by most real-time language processing systems in embedding considerable semantic information in the syntax of the language. Such a language specification is typically called a "semantic grammar." For example, words like NAUTILUS and DISPLACEMENT are not grouped together into a single ⟨NOUN⟩ category. Rather, NAUTILUS is treated as a ⟨SHIP-NAME⟩ and DISPLACEMENT as an ⟨ATTRIBUTE⟩. Similarly, very specific sentence pat-

What kind of information do you know about
Is there a doctor on board the Biddle
Display all the American cruisers in the North Atlantic
What is the name and location of the carrier nearest to New York
What is the commanding officer's name
Who commands the Kennedy
What is the Kennedy's beam
When will the Los Angeles reach Norfolk
Tell me when Taru is scheduled to leave port
Where is she scheduled to go
When will Los Angeles arrive in its home port
When will the Sturgeon arrive on station
What aircraft units are embarked on the Constellation
To which task organization is Knox assigned
Where is the Sellers
Where is Luanda
What is the next port of call of the Santa Inez
When will Tarifa get underway
Which convoy escorts have inoperative sonar systems
When will they be repaired
Which U.S. Navy DDGs have casreps involving radar systems
What Soviet ship has hull number 855
To what class does the Soviet ship Minsk belong
What class does the Whale belong to?
What is the normal steaming time for the Wainwright from Gibraltar to Norfolk
What American ships are carrying vanadium ore
How far is it to Norfolk
How far away is Norfolk
How many nautical miles is it to Norfolk
How many miles is it to Norfolk from here
How close is the Baton Rouge to Norfolk
How far is the Adams from the Aspro
What is the distance from Gibraltar to Norfolk
What is the nearest oiler
What is the nearest oiler to the Constellation
How far is it from Naples to 23-00N, 45-00W
What is the distance from the Kitty Hawk to Naples
How long would it take the Independence to reach 35-00N, 20-00W
How long is the Philadelphia
How long would it take the Aspro to join Kennedy
What is the nearest ship to Naples with a doctor on board
What is the nearest USN ship to the Enterprise with an operational air search radar
What is known about that ship
How many merchant ships are within 400 miles of the Hepburn
What are their identities and last reported locations
What cargo does the Pecos have
Who is CTG 67.3
What are the length, width, and draft of the Kitty Hawk
To whom is the Harry E. Yarnell attached
What type ships are in the Knox class
Where are the Charles F. Adams class ships
What are their current assignments

Fig. 1. Sample of acceptable inputs to LADDER
Figure 1 is continued on the next page

What subs in the South Atlantic are within 1000 miles of the Sunfish
What is the Kittyhawk doing
How many USN asw capable ships are in the Med
Where are they
What are their current assignments and fuel states
What ships are NOT at combat readiness rating C1
When will Reeves achieve readiness rating C1
Why is Hoel at readiness rating C2
When will the sonar be repaired on the Sterett
What ships are carrying cargo for the United States
Where are they going
What are they carrying
When will they arrive
Where is Gridley bound
Which cruisers have less than 50 per cent fuel on board
Where are all the merchant ships
When will the Kitty Hawk's radar be up?
What ships are in the Los Angeles class
What command does Adm. William have
Under whose opcon is the Dale
Show me where the Kennedy is!
What ship has hull number 148?
What is the next port of call for the South Carolina?
Are doctors embarked in the Kawishiwi
What kind of cargo does the Francis McGraw have?
What air group is embarked in the Constellation?
What do you know about the employment schedule of the Lang?
Which systems are down on the Kitty Hawk
What ships in the Med have doctors embarked?
How many ships carrying oil are within 340 miles of Mayport?
What sub contacts are within 300 miles of the Enterprise?
List the current position and heading of the US Navy ships in the Mediterranean every 4 hours
What is the status of the Enterprise's air search radar?
Where is convoy NL53 going
What convoy is the Transgermania in
How many embarked units are in Constellation
What ships are in British ports
What U.S. ships are within 500 miles of Wilmington?
What U.S. ships faster than the Gridley are in Norfolk
What is the fastest ship in the Mediterranean Sea
How close is that ship to Naples?
What is its home port
Print the American cruisers' current positions and states of readiness!
How is the Los Angeles powered
What ship having a normal cruising speed greater than 30 knots is the largest
Display the last reported position of all ships that are in the North Atlantic
When did the Endeavour depart the port of New York
What nationality is the ship with international radio call sign UA1D
What ports are in the database
What merchant ships are enroute to New York and within 500 miles of the Saratoga
To what country does the fastest sub belong?

Fig. 1 (continued from previous page)

```
1- What is the length of the Constellation
PARSED!
(LENGTH 1072 feet)

2- of the Nautilus
TRYING ELLIPSIS: WHAT IS THE LENGTH OF THE NAUTILUS
(LENGTH 319 feet)

3- displacement
TRYING ELLIPSIS: WHAT IS THE DISPLACEMENT OF THE NAUTILUS
(STANDARD-DISPLACEMENT 4040 tons)

4- length of the fastest American Nuclear sub
TRYING ELLIPSIS: WHAT IS THE LENGTH OF THE FASTEST AMERICAN
                     NUCLEAR SUB
(LENGTH 360 feet NAM LOS ANGELES SPEED 30.0 knots)

5- Who commands the Constallation
     SPELLING--)CONSTELLATION
PARSED!
(COMMANDER CAPT J. ELLISON)

6- Who commands JFK
TRYING ELLIPSIS: ELLIPSIS HAS FAILED
THE PARSER DOES NOT EXPECT THE WORD "JFK" TO FOLLOW
"WHO COMMANDS"
OPTIONS FOR NEXT WORD OR META-SYMBOL ARE:
⟨SHIP-NAME⟩

7- Define JFK to be like Kennedy
PARSED!
  • (JFK is now a synonym for KENNEDY, which is a ship name)
  •
  •

8- Who commands JFK        (that is, retry interaction 6)
PARSED!
(COMMANDER CAPT P. MOFFETT)

9- info JFK country
TRYING ELLIPSIS: ELLIPSIS HAS FAILED
  • (error message omitted)
  •
  •

10- Define "Info JFK country" to be like "what is the country of JFK"
PARSED!
  •
  •
  •

11- Info JFK country
PARSED!
(NATION USA)

12- Info fastest American nuclear submarine speed
PARSED!
(SPEED 30.0 knots NAM LOS ANGELES)

13- Nautilus
TRYING ELLIPSIS: INFO NAUTILUS SPEED
(SPEED 22 knots)
```

Fig. 2. Simplified interactions with LADDER

terns such as

WHAT IS THE ⟨ATTRIBUTE⟩ OF ⟨SHIP⟩

are typically used instead of more general patterns such as

⟨NOUN-PHRASE⟩ ⟨VERB-PHRASE⟩.

For each syntactic pattern, the language definer supplies an expression for computing the interpretation of instances of the pattern. INLAND's expressions for sentence-level patterns usually invoke the IDA component to retrieve information from the distributed database.

This method of language specification is easy to understand, easy to use, and, when pursued systematically, allows languages of broad coverage to be defined, as indicated in Figure 1.

To provide a more detailed view of how LIFER has been employed to produce an efficient and effective language processing system, let us examine in detail a highly simplified fragment of the INLAND language specification.

3.2 INLAND's Function in Brief

The central notions of how INLAND is constructed may be seen by considering the problem of providing English access to two files of the form

SHIP: (NAM CLASS COMMANDER HOME-PORT HULL# LOC)
CLASS: (CLASSNAME TYPE NATION FUEL LENGTH BEAM DRAFT SPEED)

located on different computers. IDA and FAM together provide levels of insulation from the real situation, so that INLAND need consider only the problem of specifying what subset of the overall database should be queried and what field values within that subset should be returned. IDA will dynamically plan the appropriate joins on the files in the database, and FAM will carry them out. (In the actual LADDER system, the intertwining of multiple files is much more complex than in the current example.)

3.3 A Miniature Language Specification

(1) *Productions.* The grammar rules may be viewed as productions of the form

metasymbol ⇒ *pattern* | *expression*,

where *metasymbol* is a metasymbol of the application language, *pattern* is a list of symbols and metasymbols in the language, and *expression* is a LISP expression whose value, when computed, is assigned as the value of the metasymbol.[3] The symbol ⟨L.T.G⟩ (LIFER top grammar) is the highest-level metasymbol of the grammar. The system's answer to complete inputs that match a pattern instantiating ⟨L.T.G⟩ will be the result of evaluating the associated LISP expression.

For example, the input

PRINT THE LENGTH OF THE KENNEDY

[3] In addition to computing values for acceptable applications of the production, the expression may also be used to reject some applications on semantic grounds. Rejection is signaled if the expression returns *ERROR* as its value.

is an instantiation of the sentence-level production

```
⟨L.T.G⟩ ⇒ ⟨PRESENT⟩ THE ⟨ATTRIBUTE⟩ OF ⟨SHIP⟩ |
          (IDA (APPEND ⟨SHIP⟩ ⟨ATTRIBUTE⟩)).
```

The input matches the pattern

```
⟨PRESENT⟩ THE ⟨ATTRIBUTE⟩ OF ⟨SHIP⟩,
```

where ⟨PRESENT⟩ matches PRINT, ⟨ATTRIBUTE⟩ matches LENGTH, and ⟨SHIP⟩ matches the phrase THE KENNEDY. If the semantic values for ⟨SHIP⟩ and ⟨ATTRIBUTE⟩, computed by means described shortly, are ((NAM EQ JOHN#F.KENNEDY)) and ((? LENGTH)), respectively, then the answer to the question is computed from the expression portion of the production as follows:

```
   (IDA (APPEND ⟨SHIP⟩ ⟨ATTRIBUTE⟩))
⇒ (IDA (APPEND '((NAM EQ JOHN.#F.KENNEDY))
        '(? LENGTH)))
⇒ (IDA '((NAM EQ JOHN#F.KENNEDY)(? LENGTH))).
```

(APPEND is a Lisp function that appends any number of lists together to form a larger list.) At this point, the IDA component is called with the argument

```
((NAM EQ JOHN#F.KENNEDY) (? LENGTH))
```

and the length of the Kennedy is retrieved as

```
   (IDA '((NAM EQ JOHN#F.KENNEDY) (? LENGTH)))
⇒ (LENGTH 1072 feet).
```

In LIFER, productions like the one just shown are defined interactively by issuing commands such as

```
PD[⟨L.T.G⟩
   (⟨PRESENT⟩ THE ⟨ATTRIBUTE⟩ OF ⟨SHIP⟩)
   (IDA (APPEND ⟨SHIP⟩ ⟨ATTRIBUTE⟩))],
```

where PD is the production definition function.

(2) *Lexical Entries.* Metasymbols, such as ⟨PRESENT⟩ and ⟨ATTRIBUTE⟩, are often associated with individual words or fixed phrases, which are maintained in LIFER's lexicons. The LIFER function MS (make set) is used to define a set of words and phrases that may match a particular metasymbol. For example, the call

```
MS[⟨ATTRIB⟩
   (CLASS COMMANDER FUEL TYPE NATION LENGTH
   BEAM DRAFT (LOCATION . LOC) (POSITION . LOC)
   (NAME . NAM) (COUNTRY . NATION)
   (NATIONALITY . NATION)((HOME PORT) . HOME-PORT)
   ((POWER TYPE) . FUEL)((HULL NUMBER) . HULL#))]
```

is used to define 16 words and fixed phrases that may match the symbol ⟨ATTRIB⟩ (which is used subsequently in defining ⟨ATTRIBUTE⟩).

After this call to MS, ⟨ATTRIB⟩ will match the words CLASS, COMMANDER, FUEL, TYPE, NATION, LENGTH, BEAM, and DRAFT. For these words, ⟨ATTRIB⟩ will take as its semantic value the word itself. ⟨ATTRIB⟩ will also match the word LOCATION, but for this match the value of ⟨ATTRIB⟩ will be LOC. Similarly, ⟨ATTRIB⟩ matches POSITION, NAME, COUNTRY, and NATIONALITY, but takes the values LOC, NAM, NATION, and NATION, respectively. ⟨ATTRIB⟩ also matches the two-word phrase HOME PORT, taking HOME-PORT as its value. For the phrase POWER TYPE, the value is FUEL; for HULL NUMBER it is HULL#. (It is assumed that the codes HOME-PORT, HULL#, LOC, and NAM are peculiar to the database and will not occur in natural language inputs.)

(3) *Subgrammars*. Metasymbols may also be defined by production rules. For example, the call

```
PD[⟨ATTRIBUTE⟩
    (⟨ATTRIB⟩)
    (LIST (LIST '? ⟨ATTRIB⟩))]
```

indicates that an ⟨ATTRIBUTE⟩ may be matched by an ⟨ATTRIB⟩, viz.:

⟨ATTRIBUTE⟩ ⇒ ⟨ATTRIB⟩.

For this production, the associated expression is

(LIST (LIST '? ⟨ATTRIB⟩)).

Since the word LENGTH matches ⟨ATTRIB⟩ and causes ⟨ATTRIB⟩ to take LENGTH as its value, the rule above indicates that LENGTH is an instantiation of ⟨ATTRIBUTE⟩. That is,

⟨ATTRIBUTE⟩ ⇒ ⟨ATTRIB⟩ ⇒ LENGTH.

The value assigned to ⟨ATTRIBUTE⟩ when it matches LENGTH is computed by the production's expression as follows:

```
   (LIST (LIST '? ⟨ATTRIB⟩))
⇒ (LIST (LIST '? 'LENGTH))
⇒ (LIST '(? LENGTH))
⇒ '((? LENGTH)).
```

This fragment of an IDA command requests the value of the LENGTH field. It was used above in answering the question "What is the length of the Kennedy?"

To recognize inputs such as

PRINT THE LENGTH BEAM AND DRAFT OF THE KENNEDY,

the concept of an ⟨ATTRIBUTE⟩ may be generalized[4] by adding two new productions as follows:

```
PD[⟨ATTRIBUTE⟩
(⟨ATTRIB⟩ AND ⟨ATTRIBUTE⟩)
(CONS (LIST '? ⟨ATTRIB⟩) ⟨ATTRIBUTE⟩)]
```

[4] The use of two symbols ⟨ATTRIB⟩ and ⟨ATTRIBUTE⟩ could be avoided by letting ⟨ATTRIBUTE⟩ directly match lexical items and by introducing such productions as ⟨ATTRIBUTE⟩ ⇒ ⟨ATTRIBUTE⟩ AND ⟨ATTRIBUTE⟩. Unfortunately, the collapse of the two symbols into one results in both ambiguity and left recursion. LIFER recognizes only one of the ambiguous interpretations. Left recursion can be tolerated by special mechanisms in LIFER's top-down left-to-right parser, but only at a considerable increase in parsing time.

```
PD[⟨ATTRIBUTE⟩
    (⟨ATTRIB⟩ ⟨ATTRIBUTE⟩)
    (CONS (LIST '? ⟨ATTRIB⟩) ⟨ATTRIBUTE⟩)].
```

(CONS is a LISP function that adds an element (in this case the list whose first element is ? and whose second element is the value of ⟨ATTRIB⟩) to the front of a list (in this case the value of ⟨ATTRIBUTE⟩).) These productions allow the phrase LENGTH BEAM AND DRAFT to be accounted for in terms of the syntax tree of Figure 3.

(4) *Complete Analysis of a Simple Query.* The examples above have indicated how the pattern

```
⟨PRESENT⟩ THE ⟨ATTRIBUTE⟩ OF ⟨SHIP⟩
```

may be defined as a top-level input and how the metasymbol ⟨ATTRIBUTE⟩ may be defined. To complete the analysis of the top-level pattern, consider now the following definitions for ⟨PRESENT⟩ and ⟨SHIP⟩.

To define ⟨PRESENT⟩, the function MS may be used:

```
MS[⟨PRESENT⟩
    (PRINT LIST SHOW GIVE ((GIVE ME) . PRINT)
    ((WHAT IS) . PRINT) ((WHAT ARE) . PRINT))].
```

This call allows ⟨PRESENT⟩ to match the words PRINT, LIST, SHOW, and GIVE and the phrases GIVE ME, WHAT IS, and WHAT ARE. The values assigned to ⟨PRESENT⟩, which might be used, for example, to direct output to the terminal or to a graphics subsystem, are not of interest here.

A ⟨SHIP⟩ may be designated in any one of a number of ways, the simplest being by name. The call

```
PD[⟨SHIP⟩
    (⟨SHIP-NAME⟩)
    (LIST (LIST 'NAM 'EQ ⟨SHIP-NAME⟩))]
```

causes ⟨SHIP⟩ to match a ⟨SHIP-NAME⟩ and to take as its value an IDA command fragment restricting the value of the NAM field to be EQ (equal) to the particular name. ⟨SHIP-NAME⟩ may be defined by MS:

```
MS[⟨SHIP-NAME⟩
    (CONSTELLATION NAUTILUS
```

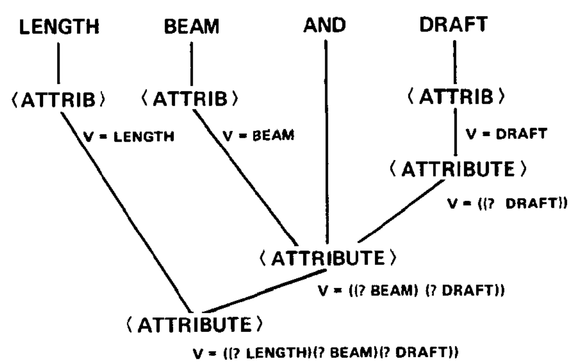

Fig. 3. ⟨ATTRIBUTE⟩ syntax tree

(KENNEDY . JOHN#F.KENNEDY)
((JOHN F . KENNEDY) . JOHN#F.KENNEDY) etc.)].

For an actual database, this list is, of course, much more extensive. To allow the optional use of "the" before the name of a ship, a supplementary production for ⟨SHIP⟩ may be defined:

PD[⟨SHIP⟩
(THE ⟨SHIP⟩)
⟨SHIP⟩].

With these definitions, LIFER has been given all the information needed to process a small class of sentence-level inputs. For example, the complete analysis of the input

WHAT IS THE LENGTH OF THE KENNEDY

is shown in the syntax tree of Figure 4. Note how the query given to IDA was generated by combining fragments from ⟨SHIP⟩ and ⟨ATTRIBUTE⟩.

From the definitions for complete inputs defined above, LIFER can infer how to process incomplete inputs in context. For example, having just parsed the input

WHAT IS THE LENGTH OF THE KENNEDY,

the system may, without additional knowledge, also handle the following sequence of incomplete inputs:

BEAM
(i.e. what is the beam of the Kennedy)
HOME PORT AND CLASS
(i.e. what is the home port and class of the Kennedy)
NAUTILUS
(i.e. what is the home port and class of the Nautilus).

The method by which these incomplete inputs are processed is discussed below.

Other inputs that the rules defined thus far will accept include:

GIVE ME THE POSITION OF THE NAUTILUS
PRINT THE HULL NUMBER AND POWER TYPE OF CONSTELLATION
SHOW THE COMMANDER COUNTRY AND TYPE OF THE JOHN F. KENNEDY.

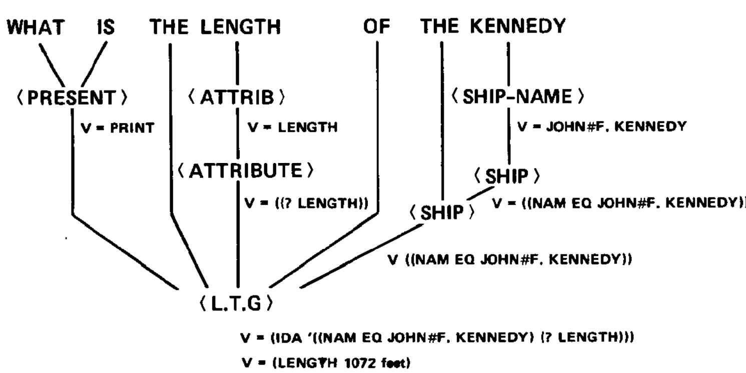

Fig. 4. Syntax tree for a complete question

3.4 Some Generalizations

The tiny fragment of language defined above already allows English access to most of the fields in the example database, given the name of a ship. This fragment may be expanded easily along many dimensions.

(1) *Generalizing* ⟨SHIP⟩. Generalizing ⟨SHIP⟩ provides one of the most fruitful expansions. Naval ships are divided into major sets called classes. For example, the Constellation is in the Kitty Hawk class. Sometimes users will wish to ask questions about all ships of a particular class; for example, HOW LONG ARE KITTY HAWK CLASS SHIPS. To do this, the language may be extended by the call

```
PD[⟨SHIP⟩
    (⟨CLASS⟩ CLASS SHIP)
    (LIST (LIST 'CLASS 'EQ ⟨CLASS⟩))],
```

where ⟨CLASS⟩ is defined to match class names and takes their database designations as values.[5] After this extension, the system will accept such inputs as

PRINT THE LENGTH OF KITTY HAWK CLASS SHIPS.

A ⟨SHIP⟩ might also match a general category such as CARRIERS, CRUISERS, or MERCHANT SHIPS. Such categories may usually be defined in terms of the TYPE field in the database. For example, CARRIERS are of type CVA, CVAN, or CVS. OILERS are AO or AOR. ⟨CATEGORY⟩ might be defined by

```
MS[⟨CATEGORY⟩
    ((CARRIER.((TYPE EQ CV)
               OR (TYPE EQ CVAN)
               OR (TYPE EQ CVS)))
     (OILER.((TYPE EQ AO)
             OR (TYPE EQ AOR)))
     etc.)].
```

A new production for ⟨SHIP⟩ may then be added such as

```
PD[⟨SHIP⟩
    (⟨CATEGORY⟩)
    (LIST ⟨CATEGORY⟩)].
```

With this production, the command

PRINT THE LOCATION OF CARRIERS

will be accepted.

Modifiers such as AMERICAN, NUCLEAR, and CONVENTIONAL are also very useful; for example,

```
MS[⟨MOD⟩
    ((AMERICAN . (NATION EQ US))
```

[5] To simplify the language definition, a language builder may supply LIFER with a preprocessor that does certain kinds of morphological transformations. For example, plural nouns such as SHIPS may be converted to the singular SHIP plus the pluralizing suffix -S. Or, as is assumed here, the suffix may simply be discarded.

```
      (NUCLEAR . (FUEL EQ NUCLEAR))
      (CONVENTIONAL . (FUEL EQ DIESEL))
      etc.)].
```

By adding

```
PD[⟨SHIP⟩
    (⟨MOD⟩ ⟨SHIP⟩)
    (CONS ⟨MOD⟩ ⟨SHIP⟩)],
```

the system will then process inputs such as

GIVE ME THE POSITION OF THE AMERICAN NUCLEAR CARRIERS.

Superlative modifiers, such as FASTEST and SHORTEST, may be defined:

```
MS[⟨MOD⟩
    ((FASTEST . (* MAX SPEED))
     (SLOWEST . (* MIN SPEED))
     (LONGEST . (* MAX LENGTH))
     etc.)]
```

Then the system will accept inputs such as

GIVE ME THE NAME AND LOCATION OF THE FASTEST AMERICAN OILERS.

This would translate into the IDA call

```
(IDA '((* MAX SPEED) (NATION EQ US)
       ((TYPE EQ AO) OR (TYPE EQ AOR))
       (? NAM) (? LOC))).
```

(2) *Generalizing* ⟨L.T.G⟩. New sentence-level productions, defined in terms of the more primitive metasymbols already described to LIFER, also greatly extend the range of language accepted. For example,

```
PD[⟨L.T.G⟩
    (⟨PRESENT⟩ ⟨SHIP⟩)
    (IDA (CONS '(? NAM) ⟨SHIP⟩))]
```

allows inputs such as

WHAT ARE THE FASTEST NUCLEAR SUBMARINES
PRINT THE CARRIERS

and

GIVE ME THE KITTY HAWK CLASS SHIPS.

As another example,

```
PD[⟨L.T.G⟩
    (WHO COMMANDS THE ⟨SHIP⟩)
    (IDA (CONS '(? COMMANDER) ⟨SHIP⟩))]
```

allows the input

WHO COMMANDS THE KENNEDY.

(3) *Calculated Answers.* Sometimes a database does not contain the information needed to answer a question directly, but nevertheless contains information that may

be used as input to a procedure that can compute the answer from more primitive data. For example, the distance between two ships is not directly available in the example database, although position data is. Suppose the function STEAMING-TIME can take speed and position information returned by IDA and calculate the time for the first ship to travel to the position of the second ship. Then, after defining ⟨SHIP2⟩ like ⟨SHIP⟩,

```
PD[⟨SHIP2⟩
    (⟨SHIP⟩)
    ⟨SHIP⟩],
```

a new top-level production may be defined as follows:

```
PD[⟨L.T.G⟩
    (HOW MANY HOURS IS ⟨SHIP⟩ FROM ⟨SHIP2⟩)
    (STEAMING-TIME
      (IDA (APPEND '((? SPEED)(? LOC)) ⟨SHIP⟩))
      (IDA (CONS '(? LOC) ⟨SHIP2⟩)))].
```

This production allows such queries as

HOW MANY HOURS IS KENNEDY FROM THE CONSTELLATION.

3.5 Extending the Lexicon with Predicates

In certain instances, it is impractical to use the MS function to explicitly list all of the symbols that might match some metasymbol. For example, if the metasymbol ⟨NUMBER⟩ is to match any number, then MS is of little value. For such cases, LIFER allows a metasymbol to be associated with a predicate function. The metasymbol will match any symbol for which the predicate returns a non-NIL value. When such a match occurs, the metasymbol will take as its semantic value the response returned by the application of the predicate.

To define a metasymbol in terms of a predicate, the function MP (make predicate) is used. For example,

```
MP[⟨NUMBER⟩ NUMBERP]
```

defines ⟨NUMBER⟩ to match any symbol for which LISP predicate function NUMBERP returns a non-NIL value. When applied to numbers, NUMBERP returns the number itself. When applied to anything else, it returns NIL.

As the following questions indicate, ⟨NUMBER⟩ has many uses in the example database:

WHAT CARRIERS HAVE LENGTHS GREATER THAN 1000 FEET
HOW FAR IS CONSTELLATION FROM 40 DEGREES NORTH 6 DEGREES EAST
WHAT SHIPS ARE WITHIN 100 MILES OF KENNEDY.

As the size of the lexicon becomes large, the predicate feature may be used to push certain large classes of words out of the natural language system and into the database itself. For example, ⟨SHIP-NAME⟩ could be defined in terms of a predicate that accesses the NAM field of the database. (This would slow the parsing operation, of course, and spelling correction could not be performed easily.)

3.6 Accepting Metalanguage Inputs

(1) *Interrogating the Language System.* It is possible to define input patterns that make reference to the LIFER package itself. For example, LIFER contains a function called SYMBOL.INFO which takes a metasymbol as its argument and prints lexical items, patterns, and predicates that may be used to match the symbol. The interface builder may incorporate this function in response expressions as in

```
PD[⟨L.T.G⟩
   (HOW IS ⟨SYMBOL⟩ USED)
   (SYMBOL.INFO ⟨SYMBOL⟩)]
```

After this call to PD,[6] a user might ask the metaquestion

```
HOW IS ⟨SHIP⟩ USED
```

and receive the reply

```
⟨SHIP⟩ MAY BE ANY SEQUENCE OF WORDS FOLLOWING
ONE OF THE PATTERNS:
  ⟨SHIP⟩ ⇒ ⟨SHIP-NAME⟩
            THE ⟨SHIP⟩
            ⟨CLASS⟩ CLASS SHIP
            ⟨MOD⟩ ⟨SHIP⟩.
```

Using other system interrogation functions, it is possible to provide in an application language for such inputs as

```
PRINT THE GRAMMAR ON FILE APP.GRAM
DISPLAY THE PRODUCTIONS EXPANDING ⟨SHIP⟩
SHOW LEXICAL ENTRIES FOR CATEGORY ⟨SHIP-NAME⟩
WHAT PREDICATE DEFINES ⟨NUMBER⟩
DRAW THE SYNTAX TREE FOR THE LAST INPUT
HOW WOULD YOU PARSE "HOW FAST IS KENNEDY"
IN WHAT PRODUCTIONS DOES ⟨SHIP⟩ APPEAR ON THE RIGHT.
```

Metaquestions requesting general information about the system, such as

```
WHAT KIND OF INFORMATION DO YOU KNOW ABOUT
WHAT'S IN THE DATABASE
HELP,
```

may easily be included in the application language. Top-level response expressions for such inputs may simply return canned explanation texts. With more sophistication, the expressions might access a semantic schema of the database to help formulate an up-to-date reply.

(2) *Personalizing the Application Language.* LIFER contains a function called SYNONYM that allows a new word to be defined as having the same meaning as a model word that is already known to LIFER. Using this function, an interface builder may introduce structures into the application language that allow users to

[6] As specified more fully in [10], the metasymbol ⟨SYMBOL⟩ may itself be defined by function MP, using a predicate that sees whether its argument is included in the list of defined metasymbols. Being so defined, ⟨SYMBOL⟩ can even match itself so that the input HOW IS ⟨SYMBOL⟩ USED may be parsed and answered.

define their own synonyms at run time. In particular,

```
PD[⟨L.T.G⟩
    (DEFINE ⟨NEW-WORD⟩ LIKE ⟨OLD-WORD⟩)
    (SYNONYM ⟨NEW-WORD⟩ ⟨OLD-WORD⟩)]
```

allows the parser to accept inputs such as

DEFINE JFK LIKE KENNEDY.

The symbols ⟨NEW-WORD⟩ and ⟨OLD-WORD⟩ are defined by predicates that will match any word. SYNONYM works by copying lexical information from ⟨OLD-WORD⟩ to ⟨NEW-WORD⟩.

LIFER also contains a function called PARAPHRASE that allows a new sequence of words to be defined as having the same meaning as a model sequence of words that the parser already accepts as a complete sentence. Using function PARAPHRASE in a response expression, the interface builder may extend the grammar by

```
PD[⟨L.T.G⟩
    (LET ⟨NEW-SEQUENCE⟩ BE A PARAPHRASE OF ⟨OLD-SENTENCE⟩)
    (PARAPHRASE ⟨NEW-SEQUENCE⟩ ⟨OLD-SENTENCE⟩)]
```

where ⟨NEW-SEQUENCE⟩ matches any sequence of words and returns a list of the matched words as its value, and ⟨OLD-SENTENCE⟩ matches any sequence of words currently accepted as a sentence in the application language.

This new rule allows computer-naive users to personalize the syntactic constructions understood by the system at run time. For example, the user might say

LET "REPORT ON KENNEDY" BE A PARAPHRASE OF
  "PRINT THE LOCATION AND COMMANDER OF KENNEDY".

The expression associated with the top-level production that matches this input sentence calls upon the paraphraser. Given the language definition defined above, LIFER then automatically adds the new production

⟨L.T.G⟩ ⇒ REPORT ON ⟨SHIP⟩

to the system, with an appropriate response expression. This new, user-defined production will allow the system to accept such new inputs as

REPORT ON THE KENNEDY
REPORT ON OILERS
REPORT ON THE FASTEST AMERICAN SUBMARINES.

LIFER's methods for learning paraphrases are discussed below.

3.7 Extendibility

The preceding subsections have indicated how a few simple notions may be drawn together to create a small interface. But can the same notions be used to create much more sophisticated systems? Until our recent experience, we would have joined others in answering, "Not likely." Long before reaching an acceptable level of performance, previous language systems, including our own, have generally grown so complex and unwieldy that further extension has been stifled.

In designing LIFER, much attention has been given to the problem of supplying interface builders with an environment supporting the incremental development of relatively broad interfaces. All LIFER functions are interactive. Parsing and language specification tasks may be intermixed, allowing interface builders to operate in a rapid, extend-and-test mode. Transition trees, which are an efficient representation for the parser to work with, are automatically produced from productions, which we have found to be an efficient representation for interface builders to work with. The system contains a grammar editor and numerous special functions for answering questions about the structure of the language definition and for tracing and debugging a grammar. Details concerning these and other features of LIFER are specified more fully in the LIFER Manual [10].

We believe that the support features of LIFER have enabled us to give the INLAND language broader coverage than previous systems. Unfortunately, we know of no adequate measure of "breadth of coverage." However, some feeling for the types of inputs accepted by LADDER may be gained by considering a sample of acceptable inputs, such as that shown in Figure 1.

4. THE TRANSITION TREE PARSER

The LIFER parser is a top-down, left-to-right parser based on a simplification of the augmented transition network (ATN) system developed by Woods [23]. Rather than use true ATNs, LIFER works with transition trees. If ⟨L.T.G⟩ is defined by the productions

⟨L.T.G⟩ ⇒ ⟨PRESENT⟩ THE ⟨ATTRIBUTE⟩ OF ⟨SHIP⟩ | e1
⇒ ⟨PRESENT⟩ ⟨SHIP'S⟩ ⟨ATTRIBUTE⟩ | e2
⇒ HOW MANY ⟨SHIP⟩ ARE THERE | e3
⇒ HOW MANY ⟨SHIP⟩ ARE THERE WITH ⟨PROPERTY⟩ | e4,

then the transition (not syntax) tree of Figure 5 would be constructed for use by the parser.

Starting at the box labeled ⟨L.T.G.⟩, the parser attempts (nondeterministically) to move toward the response expressions on the right. At each step, the parser may move to the right on a branch if the left part of the remaining portion of the input can be matched by the symbol on the branch. Literal words on a branch can be matched only by themselves. A metasymbol, such as ⟨PRESENT⟩, may be matched by a lexical item in the associated set created by MS. Or it may be matched by the predicate, if any, that has been defined for the metasymbol. Or it may be matched

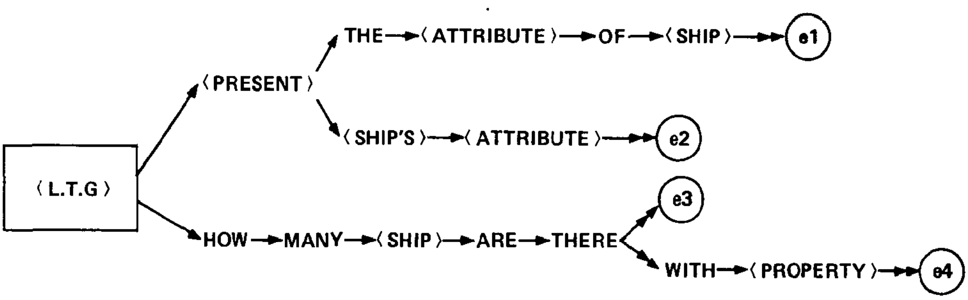

Fig. 5. Transition tree

by successfully transversing some branch of the transition tree that encodes the productions expanding the metasymbol.

At the top level, if the parser reaches a response expression as a result of accounting for the last word of an input, then a top-level match for the input has been found and the response expression is evaluated to compute a response.

5. IMPLEMENTATION OF SPECIAL LIFER FEATURES

This section presents an overview of LIFER's implementation of the spelling corrector, elliptical processor, and paraphraser.

5.1 Implementation of Spelling Correction

Each time LIFER's left-to-right, ATN parser discovers that it can no longer follow transitions along the current path, it records the failure on a failpoint list. Each entry on this list indicates the state of the system when the failure occurred (i.e. the position in the transition net and the values of various stacks and registers) and the current position in the input string. Local ambiguities and false paths make it quite normal for many failpoints to be noted even when a perfectly acceptable input is processed.

If a complete parse is found for an input, the failpoints are ignored. But if an input cannot be parsed, the list of failpoints is used by the spelling corrector, which selects those failpoints associated with the rightmost position in the input at which failpoints were recorded. It is assumed that failpoints occurring to the left were not caused by spelling errors, since some transitions using the words at those positions must have been successful for there to be failpoints to their right.[7]

The spelling corrector further restricts the rightmost failpoints by looking for cases in which a rightmost failpoint G is dominated by another rightmost failpoint F. G is dominated by F if G is a failpoint at the beginning of a subordinate transition tree that was reached in an attempt to expand F.

Working with the rightmost dominating failpoints, the spelling corrector finds all categories of words that would be valid at the point where the suspected misspelling occurred. This typically requires an exploration of subgrammars. Using the INTERLISP spelling corrector, the word of the input string associated with the rightmost failpoints is compared with the words of the categories just found. If the misspelled word is sufficiently similar to any of these lexical items, the closest match is substituted. Failpoints associated with lexical categories that include the new word are then sequentially restarted until one leads to a successful parse. (This may produce more spelling corrections further to the right.) If all restarts with the new word fail, other close lexical items are substituted for the misspelled word. If these also fail, LIFER prints an error message.

[7] This heuristic can cause LIFER to fail to find and correct certain errors. For example, if the user types CRAFT for DRAFT in WHAT DRAFT DOES THE ROARK HAVE, the spelling error will not be caught since a sentence such as WHAT CRAFT ARE NEAR ROARK would account for the initial sequence WHAT CRAFT. This is traded off against faster processing for the majority of spelling errors.

5.2 Implementation of Ellipsis

LIFER's mechanism for treating elliptical inputs presumes that the application language is defined by a semantic grammar so that a considerable amount of semantic information is encoded in the syntactic categories. Thus similar syntactic constructions are expected to be similar semantically. LIFER's treatment of ellipsis is based on this notion of similarity. During elliptical processing, LIFER is prepared to accept any string of words that is syntactically analogous to any contiguous substring of words in the last input. (If the last input was elliptical, its expansion into a complete sentence is used.)

LIFER's concept of analogy appeals to the syntax tree of the last input that was successfully analyzed by the system. For any contiguous substring of words in the last input, an "analogy pattern" may be defined by an abstraction process that works backward through the old syntax tree from the words of the substring toward the root. Whenever the syntax tree shows a portion of the substring to be a complete expansion of a syntactic category, the category name is substituted for that portion. The analogy pattern is the final result after all such substitutions.

For example, consider how an analogy pattern may be found for the substring

OF SANTA INEZ,

using the syntax tree shown in Figure 6 for a previous input, WHAT IS THE LENGTH OF SANTA INEZ.

Note that the syntax tree used in Figure 6 reflects production rules similar to those defined previously, but introduces a new metasymbol, ⟨ITEM⟩, to add more substance to the discussion. Since the SANTA INEZ portion of the substring is a complete expansion of ⟨SHIP-NAME⟩, the substring is rewritten as OF ⟨SHIP-NAME⟩. Similarly, since ⟨SHIP⟩ expands to ⟨SHIP-NAME⟩, the substring is rewritten as OF ⟨SHIP⟩. Since no other portions of the substring are complete expansions of other syntactic categories in the tree, the process stops and OF ⟨SHIP⟩ is accepted as the most general analogy pattern. If the current input matches this analogy pattern, LIFER will accept it as a legitimate elliptical input. For example, the analogy pattern OF ⟨SHIP⟩, extracted from the last input, may be used to match such current elliptical inputs as

OF THE KENNEDY
OF THE FASTEST NUCLEAR CARRIER

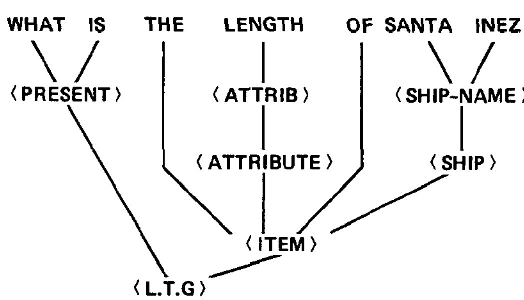

Fig. 6. Syntax tree

and

OF KITTY HAWK CLASS SHIPS.

Note that the expansion of ⟨SHIP⟩ need not parallel its expansion in the old input that originated the analogy pattern. For example, OF KITTY HAWK CLASS SHIPS is not matched by expanding ⟨SHIP⟩ to ⟨SHIP-NAME⟩ but by expanding ⟨SHIP⟩ to ⟨CLASS⟩ CLASS SHIP.

To compute responses for elliptical inputs matching OF ⟨SHIP⟩, LIFER works its way back through the old syntax tree from the common parent of OF ⟨SHIP⟩ toward the root. First, the routine for computing the value of an ⟨ITEM⟩ from constituents of the production

⟨ITEM⟩ ⇒ THE ⟨ATTRIBUTE⟩ OF ⟨SHIPS⟩

is invoked, using the new value of ⟨SHIP⟩ (which appeared in the current elliptical input) and the old value of ⟨ATTRIBUTE⟩ from the last sentence. Then, using the newly computed value for ⟨ITEM⟩ and the old value for ⟨PRESENT⟩, a new value is similarly computed for ⟨L.T.G⟩, the root of the syntax tree.

Some other substrings with their associated analogy patterns are shown below, along with possible new elliptical inputs matching the patterns:

substring:	THE LENGTH
pattern:	THE ⟨ATTRIBUTE⟩
a match:	THE BEAM AND DRAFT
substring:	LENGTH OF SANTA INEZ
pattern:	⟨ATTRIBUTE⟩ OF ⟨SHIP⟩
a match:	HOME PORTS OF AMERICAN CARRIERS
substring:	WHAT IS THE LENGTH
pattern:	⟨PRESENT⟩ THE ⟨ATTRIBUTE⟩
a match:	PRINT THE NATIONALITY
substring:	WHAT IS THE LENGTH OF SANTA INEZ
pattern:	⟨L.T.G⟩
a match:	[any complete sentence]

For purposes of efficiency, LIFER's elliptical routines have been coded in such a way that the actual generation of analogy patterns is avoided.[8] Nevertheless, the effect is conceptually equivalent to attempting parses based on the analogy patterns of each of the contiguous substrings of the last input.

5.3 Implementation of Paraphrase

LIFER's paraphrase mechanism also takes advantage of semantically-oriented syntactic categories and makes use of syntax trees. In the typical case, the paraphraser is given a model sentence, which the system can already understand, and a paraphrase. The paraphraser's general strategy is to analyze the model sentence and then look for similar structures in the paraphrase string.

In particular, the paraphraser invokes the parser to produce a syntax tree of the

[8] See Hendrix [11] for details of the algorithm.

model. Using this tree, the paraphraser determines all proper subphrases of the model, i.e. all substrings that are complete expansions of one of the syntactic categories listed in the tree. Any of these model subphrases that also appear in the paraphrase string are assumed to play the same role in the paraphrase as in the model itself. Thus the semantically-oriented syntactic categories that account for these subphrases in the model are reused to account for the corresponding subphrases of the paraphrase. Moreover, the relationship between the syntactic categories that is expressed in the syntax tree of the model forms a basis for establishing the relationship between the corresponding syntactic units inferred for the paraphrase.

(1) *Defining a Paraphrase Production.* To find correspondences between the model and the paraphrase, the subphrases of the model are first sorted. Longer phrases have preference over shorter phrases, and for two phrases of the same length, the leftmost is taken first. For example, the sorted phrases for the tree of Figure 6 are

1. ⟨ITEM⟩	THE LENGTH OF SANTA INEZ	
2. ⟨PRESENT⟩	WHAT IS	
3. ⟨SHIP-NAME⟩	SANTA INEZ	—not used
4. ⟨SHIP⟩	SANTA INEZ	
5. ⟨ATTRIB⟩	LENGTH	—not used
6. ⟨ATTRIBUTE⟩	LENGTH.	

Because the syntax tree indicates ⟨SHIP⟩ ⇒ ⟨SHIP-NAME⟩ ⇒ SANTA INEZ, both ⟨SHIP-NAME⟩ and ⟨SHIP⟩ account for the same subphrase. For such cases, only the most general syntactic category (⟨SHIP⟩) is considered. The category ⟨ATTRIB⟩ is similarly dropped.

Beginning with the first (longest) subphrase, the subphrases are matched against sequences of words in the paraphrase string. (If a subphrase matches two sequences of words, only the leftmost match is used.) The longer subphrases are given preference since matches for them will lead to generalizations incorporating matches for the shorter phrases contained within them. Whenever a match is found, the syntactic category associated with the subphrase is substituted for the matching word sequence in the paraphrase. This process continues until matches have been attempted for all subphrases.

For example, suppose the paraphrase proposed for the question of Figure 6 is

FOR SANTA INEZ GIVE ME THE LENGTH.

Subphrases 1 and 2, listed above, do not match substrings in this paraphrase. Subphrase 3 is not considered, since it is dominated by subphrase 4. Subphrase 4 does match a sequence of words in the paraphrase string. Substituting the associated category name for the word sequence yields a new paraphrase string:

FOR ⟨SHIP⟩ GIVE ME THE LENGTH.

Subphrase 5 is not considered, but subphrase 6 matches a sequence of words in the updated paraphrase string. The associated substitution yields

FOR ⟨SHIP⟩ GIVE ME THE ⟨ATTRIBUTE⟩.

Since there are no more subphrases to try, the structure

⟨L.T.G.⟩ ⇒ FOR ⟨SHIP⟩ GIVE ME THE ⟨ATTRIBUTE⟩

is created as a new production to account for the paraphrase and for similar inputs such as

FOR THE FASTEST AMERICAN SUB GIVE ME THE POSITION AND HOME PORT.

(2) *Defining a Response Expression for the Paraphrase Production.* A new semantic response expression indicating how to respond to inputs matching this paraphrase production is programmed automatically from information in the syntax tree of the model. In particular, the syntax tree indicates which productions were used in the model to expand various syntactic categories. Associated with each of these productions is the corresponding response expression for computing the interpretation of the subphrase from subphrase constituents. The paraphraser reuses selected response expressions of the model to create a new expression for the paraphrase production. The evaluation of this new expression produces the same effect that would be produced if the expressions of the model were reevaluated. Metasymbols that appear in both the paraphrase production and the model remain as variables in the new response expression. Those symbols of the model that do not appear in the paraphrase production are replaced in the expression by the constant values to which they were assigned in the model.

6. DISCUSSION

As implied by Figure 1 and the examples given in the Appendix, the INLAND system is a habitable, rather robust, real-time interface to a large database and is fully capable of successfully accepting natural language inputs from inexperienced users. In the preceding sections, we have indicated some of the key techniques used in creating this system. We now seek to place our previous remarks in perspective by considering some of the limitations of the system, the roles played by the nature of our task and the tools we built in developing the system, and some of the similarities and differences between other systems and our own.

6.1 Limitations

In considering the limitations of our work, the reader should distinguish between limitations in the current INLAND grammar and limitations in the underlying LIFER system.

(1) *Syntactic Limitations*

(a) *The Class of Languages Covered by LIFER.* Consider the set of sentences that LIFER can accept. Because in the worst case a special top-level production may be defined in LIFER to cover any (finite-length) sentence that an interface builder may wish to include in the application language, it is impossible to exhibit a single sentence that the LIFER parser cannot be made to accept. Therefore, the only meaningful questions concerning syntactic limitations of LIFER must relate to LIFER's ability to use limited memory in covering infinite or large finite sets of sentences.

LIFER application languages are specified by augmented context-free[9] grammars. Each rule in the grammar, as discussed previously, includes a context-free production, plus an arbitrarily complex response expression, which is the "augmentation." Although a purely context-free system would severely restrict the set of (nonfinite) languages that LIFER could accept, the use of augmentation gives the LIFER parser the power of a Turing machine. The critical question is whether or not the context-free productions and their more powerful augmentations can be made to support one another in meaningful ways.

To see the interplay between augmentation and context-free rules in the recognition of a classic example of non-context-free languages, consider the language composed of one or more X's followed by an equal number of Y's followed by an equal number of Z's. Let ⟨x⟩ be defined as

```
⟨x⟩ ⇒ X     | 1
⟨x⟩ ⇒ X ⟨x⟩ | (PLUS 1 ⟨x⟩).
```

Thus ⟨x⟩ matches an arbitrary sequence of X's and takes as its value the number of X's in the string. Similar definitions may be made for ⟨y⟩ and ⟨z⟩. A top-level sentence may be defined by the pattern ⟨x⟩⟨y⟩⟨z⟩, but the augmentation must check to see that the numeric values assigned to the metasymbols are all equal. If they are equal, the augmentation expression returns some appropriate response. But if they are unequal, the expression returns the special symbol *ERROR*, which the LIFER parser traps as a "semantic" (as opposed to syntactic) rejection.

The Turing machine power of LIFER is illustrated by the following trivial grammar:

```
⟨PRE-SENTENCE⟩ ⇒ ⟨WORD⟩ | (LIST ⟨WORD⟩)
               ⇒ ⟨WORD⟩ ⟨PRE-SENTENCE⟩
                        | (CONS ⟨WORD⟩
                                ⟨PRE-SENTENCE⟩)

⟨SENTENCE⟩     ⇒ ⟨PRE-SENTENCE⟩
                        | (TMPARSE ⟨PRE-SENTENCE⟩)
```

This grammar simply collects all of the words of the input into a list which is then passed to function TMPARSE, a parser of Turing machine power. In this extreme case, the LIFER parser makes virtually no use of the context-free productions, but relies exclusively on the augmentation. LIFER is best used in the middle ground between this extreme and a purely context-free system.

In other words, the class of languages for which LIFER was designed may be characterized as those allowing much of their structure to be defined by context-free rules but requiring occasional augmentation. It has been our experience that much of the subset of English used for asking questions about a command and control database falls in this class. However, we have not considered certain complex types of transformations which will be discussed in the next subsection.

(b) *Troublesome Syntactic Phenomena.* English speakers and writers often

[9] See Hopcroft and Ullman [12] for definitions of terms such as "context-free" and "context-sensitive."

omit from a sentence a series of words that do not form a complete syntactic unit. For example, consider the following family of conjunctive sentences:

(1) WHAT LAFAYETTE AND WASHINGTON CLASS SUBS ARE WITHIN 500 MILES OF GIBRALTAR

(2) WHAT LAFAYETTE CLASS AND WASHINGTON CLASS SUBS ARE WITHIN 500 MILES OF GIBRALTAR

(3) WHAT LAFAYETTE CLASS SUBS AND KITTY HAWK CLASS CARRIERS IN THE ATLANTIC ARE WITHIN 500 MILES OF GIBRALTAR

(4) WHAT LAFAYETTE CLASS SUBS IN AND PORTS ON THE ATLANTIC ARE WITHIN 500 MILES OF GIBRALTAR

(5) WHAT LAFAYETTE CLASS SUBS IN THE ATLANTIC AND KITTY HAWK CLASS CARRIERS IN THE MEDITERRANEAN SOON WILL BE WITHIN 500 MILES OF GIBRALTAR.

Sentence (1) omits the fragment CLASS SUBS ARE WITHIN 500 MILES OF GIBRALTAR from the "complete" question WHAT LAFAYETTE CLASS SUBS ARE WITHIN 500 MILES OF GIBRALTAR AND WHAT WASHINGTON CLASS SUBS ARE WITHIN 500 MILES OF GIBRALTAR. Note that the omitted fragment does not correspond to any well-formed syntactic unit, but begins in the middle of the noun phrase WHAT LAFAYETTE CLASS SUBS and continues to its right. Moreover, the fragment of the noun phrase that is left behind, namely, WHAT LAFAYETTE, is not likely to be a well-formed syntactic unit, because one would expect to have WHAT combine with LAFAYETTE-CLASS-SUBS rather than have WHAT-LAFAYETTE combine with CLASS-SUBS. As the family of sentences above illustrates, the omission of words, signaled by the conjunction AND, may be moved to the right through the sentence one word at a time, slicing up the well-formed syntactic units at arbitrary positions. INLAND has no difficulty in accepting either conjunctions or disjunctions of well-formed syntactic categories, but LIFER provides no general mechanism for dealing with omissions that slice through categories at arbitrary points.

In the SYSCONJ facility of Woods [24], special mechanisms for handling a large (but not exhaustive) class of conjunction constructions were built into the parser. Roughly, when SYSCONJ encounters the conjunction "AND" in an input X AND Y, it nondeterministically attempts to break both X and Y into three (possibly empty) parts X1-X2-X3 and Y1-Y2-Y3, such that X1-X2-X3-Y2-Y3 and X1-X2-Y1-Y2-Y3 parse as sentences with the same basic syntactic structure. In particular, X2-X3-Y2 and X2-Y1-Y2 must be expansions of the same metasymbol. The effect of SYSCONJ is to transform X1-X2-X3-AND-Y1-Y2-Y3 into X1-X2-X3-Y2-Y3 and X1-X2-Y1-Y2-Y3.

For example,

WHAT LAFAYETTE AND WASHINGTON CLASS SUBS ARE THERE

may be analyzed as

WHAT	empty	LAFAYETTE	AND	WASHINGTON	CLASS SUBS	ARE THERE
X1	X2	X3	and	Y1	Y2	Y3

Both X1-X2-X3-Y2-Y3 (WHAT LAFAYETTE CLASS SUBS ARE THERE) and X1-X2-Y1-Y2-Y3 (WHAT WASHINGTON CLASS SUBS ARE THERE) are parsed by what would correspond in INLAND to the sentence-level production

⟨L.T.G⟩ ⇒ WHAT ⟨SHIP⟩ ARE THERE

and both X2-X3-Y2 (LAFAYETTE CLASS SUBS) and X2-Y1-Y2 (WASHINGTON CLASS SUBS) are expansions of the same metasymbol, ⟨SHIP⟩. In effect, the original input is transformed into WHAT LAFAYETTE CLASS SUBS ARE THERE and WHAT WASHINGTON CLASS SUBS ARE THERE.

Handling conjunctions is just one example of the general need to perform transformations at parse time. A similar phenomenon occurs with comparative clauses, but much more is omitted and transformed. For example,

THE KITTY HAWK CARRIES MORE MEN THAN THE WASHINGTON

may be viewed as a transformed and condensed form of

THE KITTY HAWK CARRIES X-MANY MEN AND
THE WASHINGTON CARRIES Y-MANY MEN AND
X IS MORE THAN Y.

For further discussion of this subject, see Paxton [15].

(c) YES/NO *Questions*. A limitation of INLAND, although not of LIFER, is that few YES/NO questions are covered. The reason for this is pragmatic—INLAND users do not ask them. Upon reflection, the motivation for this is clear—WH questions (i.e. questions asking who, what, where, when, or how) produce more information for the questioner at a lower cost. A user might ask

IS THE KENNEDY 1000 FEET LONG,

but it is shorter to ask

HOW LONG IS THE KENNEDY,

and if the answer to the first question is NO (and if the system is so inconsiderate as to not indicate the correct length), then the user may still have to ask for the length.

Creating a grammar for YES/NO questions is easy enough. For example,

```
PD[(L.T.G)
    (IS ⟨NUMBER⟩ ⟨UNIT⟩ THE ⟨ATTRIB⟩ OF ⟨SHIP⟩)
    (YESNO.NUM.ATT ⟨NUMBER⟩ ⟨UNIT⟩ ⟨ATTRIB⟩ ⟨SHIP⟩)]
```

might be used to allow the input

IS 1000 FEET THE LENGTH OF THE KENNEDY.

Function YESNO.NUM.ATT finds the ⟨ATTRIB⟩ of ⟨SHIP⟩ using IDA. Knowing the units in which the database stores values of ⟨ATTRIB⟩, YESNO.NUM.ATT converts the returned answer into the units specified by ⟨UNIT⟩ and compares the converted value to ⟨NUMBER⟩. If the units are valid and the numbers match, YES is returned; otherwise NO is returned, and the correct answer, as computed by IDA, is printed.

(d) *Assertions.* INLAND was designed for retrieval and therefore does not handle such inputs as

THE LENGTH OF THE KENNEDY IS 1072 FEET
LET THE LENGTH OF THE KENNEDY BE 1072 FEET
SET THE LENGTH OF THE KENNEDY TO 1072 FEET.

Moreover, IDA itself does not provide for updating the database. Extending the language with new productions such as

⟨L.T.G⟩ $\Rightarrow$ SET THE ⟨ATTRIBUTE⟩ OF ⟨SHIP⟩ TO ⟨VALUE⟩

would be easy, but there are serious database issues involved regarding consistency, security, and priority. Such database problems are beyond the scope of our research.

(e) *Irregular Coverage.* One of the consequences of the ease with which interface builders can add new patterns to a LIFER grammar is that gaps may appear in coverage. For example, suppose a given language definition contains no passive constructions. Through the use of paraphrase or by direct action on the part of the interface builder, the language may be extended to cover some, but perhaps not all, passive constructions. That is, the system might be made to accept

(1) THE KENNEDY IS OWNED BY WHOM,

but not

(2) THE KENNEDY IS COMMANDED BY WHOM.

(The semantically-oriented syntactic categories for OWNED and COMMANDED may differ.) If a user knows that the system accepts (1) and that the system accepts the active

(3) WHO COMMANDS THE KENNEDY,

then he is likely to be upset when input (2) is not accepted.

In creating the language specification for INLAND, we have tried to minimize such irregularities in coverage by applying standard techniques of modular programming to the grammar specification. We feel this has been reasonably successful. Because LIFER gives the inference builder the freedom to add particular instances and subclasses of linguistic phenomena, it is his responsibility to avoid the gaps in coverage that may result.

(2) *Limitations Regarding Ambiguity.* The LIFER parser does not deal with syntactic ambiguity directly, but accepts its first successful analysis as being the sole interpretation of an input.[10] Because English contains truly ambiguous con-

[10] On October 31, 1977, LIFER was modified to allow optional production of all syntactically correct readings of an input. However, INLAND has not yet been revised to take advantage of this new option. When ambiguity is discovered, LIFER calls a user-defined subroutine with the list of parse trees (including response expressions and variable bindings at each nonterminal node) for all readings. One of the trees is to be returned for execution of the root-level response expression. A default "user" subroutine is supplied with LIFER that prints the various parse trees and asks the user to select one by number. More sophisticated subroutines are expected to be written that will enter into more natural clarification dialogues.

structions, even when semantic considerations (the "augmentations") are taken into account, this limitation can be serious. For example, in the request

(1) NAME THE SHIPS FROM AMERICAN HOME PORTS
THAT ARE WITHIN 500 MILES OF NORFOLK

the phrase THAT ARE WITHIN 500 MILES OF NORFOLK might modify either the SHIPS or the PORTS. The choice will, of course, influence the response made to the user. The current LIFER parser is biased against deep parses and will only consider the interpretation in which the clause modifies SHIPS. Even a single word can produce difficulties. For example, the word NORFOLK in request (1) could refer to a port in Virginia, a port in Great Britain, an American frigate, or a British destroyer. Thus the request is at least eight ways ambiguous.

Codd [3] has studied at some length the problem of ambiguity in the context of practical database systems and has developed the strategy of engaging in a dialogue in which the system articulates ambiguities (and other problems) and asks questions of the user to clarify the intent of his requests.

In addition to the simple syntactic form of ambiguity, exemplified by request (1), other forms of ambiguity may arise. For example, the question

(2) IS KENNEDY IN RADAR RANGE OF THE KNOX

is syntactically unambiguous, but the meaning might be either

IS KENNEDY IN KNOX'S RADAR RANGE

or

IS KNOX IN KENNEDY'S RADAR RANGE.

This example represents a purely semantic ambiguity.

Similarly,

(3) IS KENNEDY NEARER TO GIBRALTAR THAN KITTY HAWK

might be considered syntactically unambiguous in its present form, yet it has two possible meanings. By adding "missing words" at two different points in the input it is possible to produce the readings

IS THE KENNEDY NEARER TO GIBRALTAR THAN
the kennedy is near to THE KITTY HAWK

and

IS THE KENNEDY NEARER TO GIBRALTAR THAN
THE KITTY HAWK **is near to Gibraltar.**

Examples of ambiguity such as queries (1), (2), and (3) just given begin to show the difficulty of dealing with the problem in any general way. However, in the domain of INLAND, ambiguities have tended to arise only infrequently and have presented only minor problems for our particular application. Fortunately, our users have been very helpful by tending to avoid the use of the long and complex constructions that are most likely to lead to ambiguities. Perhaps this is because the teletype medium inclines users to prefer short, simple constructions.

Even though LIFER does not deal with ambiguity directly, certain types of ambiguities may be trapped and treated by using the response expressions of LIFER production rules. For example,

PD[⟨L.T.G⟩
(IS ⟨SHIP1⟩ IN ⟨RANGE-TYPE⟩ RANGE OF ⟨SHIP2⟩)
(COMPUTE.RANGE ⟨SHIP1⟩ ⟨SHIP2⟩ ⟨RANGE-TYPE⟩)]

will accept such inputs as

IS KENNEDY WITHIN RADAR RANGE OF KNOX

and call the function COMPUTE.RANGE to respond. COMPUTE.RANGE is given the two ships and the range type as an input. Knowing the pattern to be inherently ambiguous, COMPUTE.RANGE may enter into a (formal) conversation with the user to resolve the ambiguity.

The INLAND grammar also tries to avoid ambiguity whenever possible. For example, the phrase AMERICAN ARMORED TROOP CARRIER might mean a ship (of any nationality) that carries armored troops from the U.S. military, or an American ship that carries armored troops (of any nationality), or a ship that carries troops that were armored by the U.S., or a ship that was armored by the U.S. and that carries troops, or any one of several other combinations. In INLAND, ARMORED-TROOP-CARRIER is recognized as a fixed phrase and all the problems with ambiguity vanish.

(3) *Limitations Regarding Definite Noun Phrases*

(a) *The Restricted Context Problem.* A phrase such as THE AMERICAN SUBS may be used to refer to different American submarines, depending upon the "context" in which it appears. For example, if the Washington, the Churchill, and the Lincoln are being discussed, then THE AMERICAN SUBS in

HOW OLD ARE THE AMERICAN SUBS

refers to the Washington and the Lincoln. Had the current context concerned the Roosevelt, Jefferson, and Leninsky Komsomol, then THE AMERICAN SUBS would have referred to the Roosevelt and the Jefferson. The point is that the meanings of certain noun phrases are dependent upon the contexts in which the phrases appear.

INLAND has only a limited ability to handle phrases such as THE SHIPS, THE AMERICAN SUB, and THOSE CRUISERS, which are said to be "definitely determined." As opposed to indefinitely determined noun phrases (such as A SHIP), which refer to the existence of objects not currently in context, definite noun phrases are often used to refer to a particular object or set of objects that is already in context. In dealing with a database, "in context" may usually be taken to mean "in the database." Thus the phrase THE AMERICAN SUBS generally means "the American subs in the database," and this is the interpretation that INLAND almost always places on this phrase. But suppose the user has just asked WHAT SUBS ARE IN THE MEDITERRANEAN and has been answered by a list of several subs, some of which are American and some of which belong to other countries. If the user now asks WHAT ARE THE POSITIONS OF THE AMERICAN SUBS he expects only the positions of American subs in the Mediterranean, but is given information about all American subs in the database. The problem is that the local context established by previous questions is more restricted than the total database and INLAND has not received enough lexical and syntactic clues to recognize this. (Had the input been WHAT ARE THE POSITIONS OF THE AMERICAN ONES, the use of the pronoun would have signaled the local context and INLAND would have replied properly.)

Where the context is very clear, INLAND can sometimes handle a restricted perspective on the database. For example, following

SELECT A MAP OF THE NORTH ATLANTIC

the query

DISPLAY THE AMERICAN SUBS

will cause the retrieval of only those subs in the North Atlantic, because others could not be displayed on the map in any case.

We know of no applied language system that deals adequately with this problem. However, significant experimental results are described in Grosz [7].

(b) *Some Methods for Treating Pronouns.* Even though the general problem of properly resolving pronouns is quite difficult, simple techniques can cover a large number of cases. For example, there are many trivial uses of pronouns in which no resolution is needed at all. Examples include

WHAT TIME IS IT
WAS IT 1968 WHEN THE KENNEDY WAS LAUNCHED

and instances in which the pronoun references an earlier phrase in a pattern as in

WHEN WILL ⟨SHIP⟩ ⟨HAVE⟩ ITS ⟨PART⟩ ⟨REPAIRED⟩
(e.g. WHEN WILL THE KENNEDY GET ITS RADAR FIXED).

Very often pronouns are used in natural language queries to refer to things mentioned in the previous question. Thus in the sequence

WHAT IS THE LENGTH OF THE KENNEDY
WHAT IS HER SPEED,

the pronoun HER refers to THE KENNEDY. Suppose the first sentence above is interpreted by means of the production

⟨L.T.G⟩ ⇒ ⟨PRESENT⟩ THE ⟨ATTRIBUTE⟩ OF ⟨SHIP⟩

and the second sentence by

⟨L.T.G⟩ ⇒ ⟨PRESENT⟩ ⟨SHIP'S⟩ ⟨ATTRIBUTE⟩.

The primary method for matching ⟨SHIP'S⟩ might be through a production such as

⟨SHIP'S⟩ ⇒ ⟨SHIP⟩ ⟨-'S⟩

where ⟨-'S⟩ is the possessive-forming suffix which is stripped off by a preprocessor.[11]

[11] Alternatively, a set of possessive nouns naming ships could be defined and the stripper not used, or, as is the case in INLAND, possessives could be thrown away altogether and ⟨SHIP'S⟩ could be made equivalent to ⟨SHIP⟩.

This primary method may be extended so that ⟨SHIP'S⟩ may also match HER or ITS or THEIR if a ⟨SHIP⟩ was used in the last input. This will allow WHAT IS HER SPEED to be interpreted as WHAT IS KENNEDY'S SPEED.

To extend the definition of ⟨SHIP'S⟩ to match pronouns, first define a predicate SHIP.PRONOUN that will return a non-NIL value if its argument is a possessive pronoun and the last input contained a ⟨SHIP⟩. The predicate may be defined as

```
(LAMBDA (WORD)
        (AND (MEMBER WORD '(HER ITS THEIR))
             (LIFER.BINDING '⟨SHIP⟩)))
```

where LIFER.BINDING is a LIFER function that determines whether the metasymbol given as its argument had a binding in the interpretation of the last input and, if so, returns the binding.[12] Using predicate SHIP.PRONOUN, the definition of metasymbol ⟨SHIP'S⟩ may be extended by the call

```
MP(⟨SHIP'S⟩ SHIP.PRONOUN).
```

Another technique, which works nicely for some classes of anaphoric references, involves the use of global variables (sometimes called "registers"). For example, suppose that each response expression associated with a pattern defining the metasymbol ⟨SHIP⟩ is so constructed that it will set the global variable LATEST-SHIP to the value it returns as the binding of ⟨SHIP⟩. To be concrete,

```
PD[⟨SHIP⟩
   (⟨SHIP-NAME⟩)
   (SETQ LATEST-SHIP
         (LIST (LIST 'NAM 'EQ ⟨SHIP-NAME⟩)))]
```

causes ⟨SHIP⟩ to match a ⟨SHIP-NAME⟩ as defined previously. The response expression that computes the value of ⟨SHIP⟩ will return the same value as defined above, but, as a side effect, it will now also set the global variable LATEST-SHIP to the same value. Later, when phrases such as THE SHIP or THAT SHIP are used to refer to the last ship mentioned, the global variable LATEST-SHIP may be used to recall that ship. For example, if ⟨DET-DEF⟩ is defined to match definite determiners (e.g. THAT, THE), then

```
PD[⟨SHIP⟩
   (⟨DET-DEF⟩ SHIP)
   LATEST-SHIP]
```

will define structures that allow ⟨SHIP⟩ to match THE SHIP and take as its value the value of the LATEST-SHIP. Note that LATEST-SHIP is always ready with the value of the latest ⟨SHIP⟩ mentioned, but (LIFER.BINDING '⟨SHIP⟩) is of help only if ⟨SHIP⟩ was used in the last input.

(4) *Limitations in Processing Elliptical Inputs.* After successfully processing the complete sentence

(1) HOW MANY CRUISERS ARE THERE

[12] If there were multiple occurrences of the symbol in the last input, the leftmost-topmost instance is returned.

LIFER will accept the elliptical input

(2) CRUISERS WITHIN 600 MILES OF THE KNOX

but not

(3) WITHIN 600 MILES OF THE KNOX.

The elliptical processor is based on syntactic analogies. Input (2) is a noun phrase which is analogous to the noun phrase CRUISERS of input (1). Input (3), on the other hand, is a modifier that is intended to modify the CRUISERS of input (1). Because input (1) has no modifiers, elliptical input (3) has no parallel in the original input and hence cannot be accepted.

(5) *Other Limitations.* A few other important limitations of INLAND and LIFER are worth mentioning briefly.

First, LIFER has no "core grammar" that is ready to be used on any arbitrary database. This is because LIFER was designed as a general purpose language processing system and makes no commitment whatever to the types of programs and data structures for which it is to provide a front end or even to which natural language is to be accepted. LIFER might, for example, be used to build a Japanese language interface to a program that controls a robot arm. This could not be done if assumptions had been made restricting LIFER to database applications and to the English language. Thus LIFER contrasts with systems such as Thompson and Thompson's REL (rapidly extendable language) [20], which provides a core grammar but which requires reformatting of data into the REL database.[13]

Some systems, such as ROBOT (Harris [8, 9]), use the information in the database itself as an extension of the language processor's lexicon. The LIFER interface may do this also but need not. If one elects not to use the database as lexicon, and this choice was made in INLAND, then the lexicon must be extended whenever new values are added to the database that a user may want to mention in his queries.[14] The price of using the database itself as an extended lexicon is that the database must be queried during the parsing process. For very large databases, this operation will probably be prohibitively expensive.

INLAND, of course, is basically a question answerer that relies on a database as its major source of domain information. In particular, INLAND cannot read newspaper articles or other extended texts and record their meaning for subsequent querying. Moreover, although it is perfectly reasonable that the LIFER parser might be used for a text reading system, LIFER itself contains no particular facilities other than calls to response expressions for recording or reasoning about complex bodies of knowledge.

6.2 The Role of the Task Domain

The limitations presented in the last subsection would cause major difficulties in dealing with many areas of natural language application. However, for our particu-

[13] Fragments of foreign-language versions of INLAND have been used to access the naval database in Swedish and Japanese.

[14] Because LADDER accesses data over the ARPANET, we felt the overall system would be intolerably slow if the actual database was used at parse time.

lar application, the limitations did not prevent the creation of a robust and useful system. In the next few paragraphs, we briefly outline some of the key features of the application that simplified our task.

The creation of INLAND was greatly facilitated by the nature of the particular interface problem that was addressed—providing a decision maker with access to information he knows is in a database. Because the user is expected to know what kinds of information are available and is expected to follow the technical terms and styles of writing that are typical in his domain of decision making, we can establish strong predictions about a user's linguistic behavior and hence INLAND needs to cover only a relatively narrow subset of language.

A second factor in facilitating the creation of the natural language interface was the interface provided by the IDA and FAM components of the LADDER system. By providing a simplistic view of what is in fact a complex and highly intertwined collection of distributed data, IDA and FAM helped greatly in simplifying the LISP response expressions associated with productions in the INLAND grammar.

In short, IDA allows the database to be queried by high-level information requests that take the form of an unordered list of two kinds of items: fields whose values are desired, and conditions on the values of associated fields. Using IDA, the INLAND grammar need never be concerned with any entities in the database other than fields and field values. Furthermore, because the input to IDA is unordered, the construction of segments of a call to IDA can be done while parsing lower-level metasymbols.

The performance of INLAND for a given user is also enhanced by the user's own, often subconscious, tendency to adapt to the system's limitations. Because INLAND can handle at least the most straightforward paraphrases of most requests for the values of any particular fields, even a new user has a good chance of having his questions successfully answered on the first or second attempt. It has been our experience that those who use the system with some regularity soon adapt the style of their questions to that accepted by the language specification. The performance of these users suggests that they train themselves to understand the grammar accepted by INLAND and to restrict their questions whenever possible to forms within the grammar. Formal investigation of this subjectively observed phenomenon might prove very interesting.

6.3 The Role of Human Engineering

Although the basic language processing abilities provided by LIFER are similar to those found in some other systems, LIFER embodies a number of human engineering features that greatly enhance its usability. These humanizing features include its ability to deal with incomplete inputs and to allow users to extend the linguistic coverage at run time. But, more importantly, LIFER provides easy-to-understand, highly interactive functions for specifying, extending, modifying, and debugging application languages. These features provide a highly supportive environment for the incremental development of sophisticated interfaces. Without these supporting features, a language definition rapidly becomes too complex to manage and is no longer extendable. With support, the relatively simple types of linguistic constructions accepted by LIFER may be used to produce far more sophisticated interfaces than was previously thought possible.

Creating a LIFER grammar that covers the language of a particular application may be thought of constructively as writing a program for a parser machine. All the precepts of good programming—top-down design, modular programming, and the like—are relevant to good design of a semantic grammar. A well programmed grammar is easy to augment, because new top-level patterns are likely to refer to lower-level metasymbols that have already been developed and shown to work reliably. Thus the task of adding new top-level productions to a grammar is analogous to the task of adding new capabilities to a more typical body of computer code (such as a statistics package) by defining new capabilities in terms of existing subroutines.

No matter how well programmed a grammar might be, as the complexity of the grammar increases, the interactions among components of the language specification will grow. This leads the language designer into the familiar programming cycle of program, test, and debug. With many systems for parsing and language definition, the cycle may take many minutes for each iteration. With LIFER, when a new production is interactively entered into the grammar, it is immediately usable for testing by parsing sample inputs. The time required for the cycle of program, test, and debug is thus dependent on the thinking time of the designer, not the processing time of the system. Because the designer can make very effective use of his time, he can support, maintain, and extend a language specification of far greater complexity than would otherwise be possible.

The basic parsing technology of LIFER is not really new. But the human engineering that LIFER provides for interface builders has allowed us to better manage the existing technology and to apply it on a relatively large scale.

6.4 Related Work

As indicated by the February 1977 issue of the SIGART Newsletter [5], which contains a collection of 52 short overviews of various research efforts in the general area, interest in the development of natural language interfaces is widespread. Our own work is similar to that of several others.

The LIFER parser is based on a simplification of the ideas developed in the LUNAR parser of Woods and others [23, 26]. In particular, LIFER manipulates internal structures that reflect Woods's ATN formalism. Woods's parser was used as a component of a system that accessed a database in answering questions about the chemical analysis of lunar rocks. The system did not use semantically-oriented syntactic categories and the database was smaller and less complex than that used by INLAND, although the database query language was more general than that accepted by IDA.

Woods's ATN formalism has been used in a variety of systems, including a speech understanding system [25], and the semantically-oriented systems of Waltz [22], Brown and Burton [1], and Burton [2]. These latter systems do not use the LUNAR parser, but rather compile the ATN formalism into procedures that in turn perform the parsing operation directly, without using a parser/interpreter to interpret a grammatical formalism. Compilation results in greater parsing speed, which is of importance for many applications. However, compilation also makes personalization features, such as PARAPHRASE, much more difficult to implement and increases the time of the program-test-debug cycle.

The first natural language systems to make extensive use of semantic grammars were those of Brown and Burton [1] and Burton [2]. These systems were designed for computer-assisted instruction rather than as interfaces to databases.

In work very similar to our own, Waltz [22] has devised a system called PLANES which answers questions about the maintenance and flight histories of airplanes. PLANES uses both an ATN and a semantic grammar. Apparently the system does not include a paraphrase facility similar to LIFER's. It does support the processing of elliptical inputs by a technique differing from our own and supports clarification dialogues with users.

The PLANES language definition makes less use of syntactic information than INLAND. In particular, PLANES looks through an input for constituent phrases matching certain semantically-oriented syntax categories. When one of these constituent phrases is found, its value is placed in a local register that is associated with the given category. Rather than attempt to combine these constituents into a complete sentence by syntactic means, "concept case frames" are used. Essentially, PLANES uses case frames to decide what type of question has been asked by looking at the types and values of local registers that were set by the input. For example, the three questions

WHO OWNS THE KENNEDY
BY WHOM IS KENNEDY OWNED
THE KENNEDY IS OWNED BY WHOM

would all set, say, an ⟨ACT⟩ register to OWN and a ⟨SHIP⟩ register to KENNEDY. The case frames can determine what question is asked simply by looking at these registers. Performing a complete syntactic analysis such as INLAND does require different constructions for each question pattern.[15]

If the input following one of the three questions asked in the preceding paragraph is the elliptical fragment "KNOX," the ⟨SHIP⟩ register is reset. Because no case frame is associated with ⟨SHIP⟩ alone and because ⟨SHIP⟩ was used in the last input, the ⟨ACT⟩ register is inherited in the new context and the elliptical input properly analyzed. When more than one case frame matches an input, PLANES enters into a clarification dialogue with the user to decide which was intended. (This conversation prints interpretations of inputs in a formal query language.)

The use of case frames is very attractive in that it allows many top-level syntactic patterns to be accounted for by a single rule. However, it is inadequate for complex

[15] LIFER may be used to support case frames, although this was not done in INLAND. In particular, ⟨L.T.G⟩ may be defined as an arbitrary sequence of ⟨CONSTITUENT⟩s, where ⟨CONSTITUENT⟩ may be expanded as any of the semantically-oriented syntax categories used by the case system. The response expression associated with the expansions of ⟨CONSTITUENT⟩ cause global registers to be set, and the response expression associated with ⟨L.T.G⟩ ⇒ ⟨CONSTITUENTS⟩ may make use of these registers and the case frames in computing a top-level response. A case frame system supported by LIFER would, of course, inherit LIFER's run-time personalization and introspection features.

inputs. The question IS KNOX FASTER THAN KENNEDY contains two ⟨SHIP⟩s. Only the syntax tells us which to test as the faster of the two. Compound-complex sentences would be extremely difficult to process without extensive use of syntactic data. Waltz is investigating ways of supplementing his case frames with nominal pieces of syntactic information.

Codd's concept of the RENDEZVOUS system [3] for interface to relational databases provides many ideas concerning clarification dialogues that might be included in LIFER at some later date. RENDEZVOUS is failsafe in that it can fall back on multiple choice selection if natural language processing fails completely.

Another applied natural language system whose underlying philosophy is akin to that of LIFER is the REL system developed by Thompson and Thompson [20]. REL is a data retrieval system like LADDER, though REL requires data to be stored in a special REL database. The grammar rules of REL contain a context-free part and an augmentation very much like those of LIFER. As its name implies, REL was intended to be easily extendable by interface builders. Much effort has gone into making REL run rapidly and it is almost certainly faster than LIFER. However, this speed was gained by a low-level language implementation with the unfortunate side effect that response expressions are not easily written.

Recently, the Artificial Intelligence Corporation introduced a commercial product called ROBOT for interfacing to databases. As described in Harris [8], ROBOT "calls for mapping English language questions into a language of database semantics that is independent of the contents of the database." The database itself is used as an extension of the dictionary, and the structure of files within the database helps in guiding the parser in the resolution of ambiguities. Our own research indicates that the types of linguistic construction employed by users are rather dependent on the content of the database. We also worry that extensive recourse to a database of substantial size may greatly slow the parsing process, unless the file is indexed on every field. Moreover, our database is coded largely in terms of abbreviations that are unsuitable as lexical entries. Nevertheless, the notion of using the data itself to extend the capabilities of the language system is very attractive.

In addition to the work on near-term application systems, a number of workers are currently addressing longer-range problems of accessing databases through natural language. See, for example, Mylopoulos et al. [14], Sowa [18], Walker et al. [21], and Sacerdoti [16]. There are, of course, many people engaged in research in the general area of natural language processing, but a survey of their work is beyond the scope of this paper.

7. CONCLUSION

We have described a system called LADDER that provides natural language access to a large, distributed database. We have shown that the language processing component of this system, although based on simple principles and subject to certain limitations, is sufficiently robust to be useful in practical applications. Moreover, we have indicated that LADDER is not an isolated system but that other applied language systems have achieved significant levels of performance as well, particularly in interfacing to databases. We believe that the evidence presented indicates clearly that, for certain restricted applications, natural language access to databases has become a practical and practicable reality.

8. GLOSSARY

DBMS	Database management system.
FAM	File access manager. Maps generic file names onto specific file names on specific computers at specific sites. Initiates network connections, opens files, and monitors for certain errors.
IDA	Intelligent data access. Presents a structure-free view of a distributed database.
INLAND	Informal natural language access to Navy data. The natural language interface to IDA, which incorporates a special-purpose LIFER grammar.
LADDER	Language access to distributed data with error recovery. Our total system composed of INLAND, IDA, and FAM.
LIFER	Language interface facility with ellipsis and recursion. The general facility for creating and maintaining linguistic interfaces.
MP	Make predicate. The LIFER function for defining a metasymbol as a predicate function.
MS	Make set. The LIFER function for defining a metasymbol as a set of lexical items.
PD	Pattern define. The LIFER function for defining a metasymbol as a pattern expansion.
VLDB	Very large database.

APPENDIX. AN EXAMPLE SESSION WITH LADDER

```
@LADDER
Please type in your name: TOD S.

Do you want instructions? (type FIRST LETTER of response) No

Do you want to use 2 Data Computers? No

Do you want to specify a current location (default = Norfolk)? No

Do you wish distance/direction calculations to default to GREAT CIRCLE,
or RHUMB LINE? (you can override by specifying in the query) Great Circle

1_What is the current position of the Kennedy?
PARSED!
Parse time: .68 seconds
        * This counts cpu time used by INLAND.
IDA: ((? PTP) (? PTD) (NAM EQ 'KENNEDY% JF'))
        * This is the call to IDA.
Connecting to Datacomputer at CCA1:
        * FAM indicates which computer is being accessed.  The next
        * 13 lines are interactions between FAM and the Datacomputer.
>> ;0031 771108184236   IONETI: CONNECTED TO SRI-KL-22700010
>> ;J150 771108184238   FCRUN: V='DC-4/10.00.1' J=3 DT='TUESDAY, NOVEMBER
8, 1977 13:42:38-EST' S='CCA'
>> !0041 771108184239   DNCTNX: DATACOMPUTER GOING DOWN IN 905 MIN BECAUSE
SYSTEM IS GOING DOWN AT WED NOV 9 77 5:00:00AM-EST FOR 240 MIN DUE TO
```

```
SCHEDULED PM
>> ;J200 771108184239   RHRUN: READY FOR REQUEST
*> Set parameters
*<      Exit
CCA1:^Z
*> Set parameters
*< V    Verbosity (-1 to 5):   1
*< P    PROCEED with Datalanguage  [confirm with <CR>]
        * The connection has now been established.   FAM now logs in
        * and opens the necessary files.
CCA1:LOGIN $TOP.ACCAT.GUEST ;
CCA1:OPEN $TOP.ACCAT.SAGALOWICZ.NSTDPORT1 WRITE;
CCA1:OPEN $TOP.ACCAT.NTRACKHIST READ;
CCA1:OPEN $TOP.ACCAT.NNSHIP READ;
        * FAM now transmits the query.
CCA1:FOR R1 IN NNSHIP WITH (NAM EQ 'KENNEDY JF') FOR NSTDPORT1 , R2 IN
CCA1:NTRACKHIST WITH R2.UICVCN EQ R1.UICVCN BEGIN STRING1 = R2.PTP STRING2 =
CCA1:R2.PTD END;
*> Total bytes transferred: 27
IDA = ((PTP '6000N03000W' PTD 7601171200))
        * This is the value returned by IDA.
Computation time for query: 4.077 seconds
        * This counts cpu time used by IDA and FAM.  Extra time is
        * needed to establish the network connection, log in, and
        * open files.
Real time for query: 224.725 seconds
        * This measures real time from the time the request is made
        * to IDA until IDA returns the answer.
(POSITION 6000N03000W DATE 7601171200)
        * Kennedy was last reported to be at 60 degrees North,
        * 30 degrees West, at noon on January 17, 1976.

2_of kitty hawk
Trying Ellipsis:  WHAT IS THE CURRENT POSITION OF KITTY HAWK
Parse time: .97 seconds

IDA: ((? PTP) (? PTD) (NAM EQ 'KITTY$ HAWK'))
CCA1:FOR R1 IN NNSHIP WITH (NAM EQ 'KITTY HAWK') FOR NSTDPORT1 , R2 IN
CCA1:NTRACKHIST WITH R2.UICVCN EQ R1.UICVCN BEGIN STRING1 = R2.PTP STRING2 =
CCA1:R2.PTD END;

*> Total bytes transferred: 27
IDA = ((PTP '3700N01700E' PTD 7601171200))
Computation time for query: 1.077 seconds
Real time for query: 78.105 seconds
(POSITION 3700N01700E DATE 7601171200)

3_To what country does each merchant ship in the north atlantic belong
PARSED!
Parse time: .386 seconds

IDA: ((? NAT) (? NAM) ((TYPE EQ 'BULK') OR (TYPE EQ 'TNKR'))
(PTPNS EQ 'N') (PTPEW EQ 'W') ((PTPY GT 600) OR (PTPX LT 3600) OR
(PTPX GT 3900)) (? PTP) (? PTD))
CCA1:OPEN $TOP.ACCAT.SAGALOWICZ.NSTDPORT2 WRITE;
CCA1:OPEN $TOP.ACCAT.NNMOVES READ;
CCA1:FOR R1 IN NNMOVES WITH ((TYPE EQ 'BULK') OR (TYPE EQ 'TNKR')) FOR R2 IN
CCA1:NTRACKHIST WITH (PTPNS EQ 'N') AND (PTPEW EQ 'W') AND
CCA1:((PTPY GT 600) OR (PTPX LT 3600) OR (PTPX GT 3900)) AND R2.UICVCN EQ
CCA1:R1.UICVCN FOR NSTDPORT2 , R3 IN NNSHIP WITH R3.UICVCN EQ R2.UICVCN BEGIN
CCA1:STRING1 = R2.PTP STRING2 = R2.PTD STRING3 = R3.NAT STRING4 = R3.NAM END;

*> Total bytes transferred: 4582
IDA = ((NAT 'US' NAM 'GREENVILLE$ VICTORY' PTP '1533N03010W' PTD
7601171200) (NAT 'US' NAM 'TOULE$-J' PTP '1533N03010W' PTD 7601171200)
...)
```

```
Computation time for query: 5.702 seconds
Real time for query: 228.231 seconds
I have 79 answers; do you want to see them all? ...Yes

NAT SHIP               POSITION    DATE
US  GREENVILLE VICTORY 1533N03010W 7601171200
US  TOULE J            1533N03010W 7601171200
          .
          .       {other entries omitted here}
          .
WG  TRANSQUEBEC        4633N02129W 7601171200
NO  TALABOT            0600N01130W 7601171200

4_the fastest sub
Trying Ellipsis:  TO WHAT COUNTRY DOES THE FASTEST SUB BELONG
Parse time: 1.484 seconds

IDA: ((? NAT) (? NAM) (* MAX MCSF) (TYPE1 EQ 'S') (TYPE2 EQ 'S'))
        * In the interests of brevity and clarity, the remaining
        * interactions with the Datacomputer are omitted.
IDA = ((NAT 'US' NAM 'LOS% ANGELES' MCSF '30.0'))
Computation time for query: 2.317 seconds
Real time for query: 173.3 seconds
(NAT US SHIP LOS ANGELES MXSPD 30.0)

5_what ships are en route to america
Trying Ellipsis:  Ellipsis has failed.
The parser does not expect the word "AMERICA" to follow
"WHAT SHIP ARE EN ROUTE TO"
Options for next word or symbol are:
<COUNTRY>,  <PORT>
end-of-list
        * LIFER cannot parse the question; "America" is not in
        * the vocabulary.

6_what is a <country>
PARSED!
        * In addition to questions about the data base, INLAND can
        * answer questions about the grammar itself.
"<COUNTRY>" may be any sequence of words following one of the patterns:
<COUNTRY> =>  THE <COUNTRY>
<COUNTRY> =>  U S
<COUNTRY> =>  U S S R
<COUNTRY> =>  U S S
<COUNTRY> =>  U S A
<COUNTRY> =>  U K
<COUNTRY> =>  SOVIET UNION
<COUNTRY> =>  UNITED STATES
<COUNTRY> =>  UNITED KINGDOM
<COUNTRY> =>  SOUTH AFRICA
<COUNTRY> =>  WEST GERMANY
<COUNTRY> =>  SAUDI ARABIA
<COUNTRY> =>  GREAT BRITAIN
<COUNTRY> =>  H M S
"<COUNTRY>" may be any member of the set {ANGOLA ANGOLAN ARABIA ARABIAN
ARGENTINA ARGENTINAN BRITAIN BRITISH CANADA CANADIAN DUTCH EGYPT EGYPTIAN
ENGLAND ENGLISH FOREIGN FRANCE FRENCH GERMAN GERMANY H.M.S. HMS ITALIAN
ITALY LIBERIA LIBERIAN NETHERLANDS NORWAY NORWEGIAN PORTUGAL PORTUGUESE
RUSSIA RUSSIAN SOVIET SPAIN SPANISH U.K. U.S. U.S.A. U.S.S. U.S.S.R. UK
US USA USS USSR VENEZUELA VENEZUELAN}
Finished
7_define america like usa
PARSED!
        * The user may add new synonyms to the vocabulary.
FINISHED
8_redo 5
```

```
        * Here we are using the "redo" feature of INTERLISP.
PARSED!
Parse time: .356 seconds

IDA: ((? NAM) (DSC EQ 'US'))

IDA = ((NAM 'KENNEDY% JF') (NAM 'LOS% ANGELES') (NAM 'BATON% ROUGE')
(NAM 'PHILADELPHIA') (NAM 'POGY') (NAM 'ASPRO') (NAM 'SUNFISH') (NAM
'KAWISHIWI'))
Computation time for query: 1.098 seconds
Real time for query: 67.16 seconds
SHIP = KENNEDY JF, LOS ANGELES, BATON ROUGE, PHILADELPHIA, POGY, ASPRO,
SUNFISH, KAWISHIWI

9_how many of them are navy ships
              THEM => ((DSC EQ 'US'))
PARSED!
        * 'Them' or 'she' is currently always interpreted as a
        * reference to a set of ships in the previous query.
Parse time: .505 seconds

IDA: ((? NAM) (DSC EQ 'US') ((TYPE NE 'BULK') AND (TYPE NE 'TNKR')))

IDA = ((NAM 'KENNEDY% JF') (NAM 'LOS% ANGELES') (NAM 'BATON% ROUGE')
(NAM 'PHILADELPHIA') (NAM 'POGY') (NAM 'ASPRO') (NAM 'SUNFISH') (NAM
'KAWISHIWI'))
Computation time for query: 1.205 seconds
Real time for query: 89.417 seconds
8 of them:
SHIP = KENNEDY JF, LOS ANGELES, BATON ROUGE, PHILADELPHIA, POGY, ASPRO,
SUNFISH, KAWISHIWI

10_give status kitty hawk
Trying Ellipsis: Ellipsis has failed.
The parser does not expect the word "STATUS" to follow
"GIVE"
Options for next word or symbol are:
<RELATIVE.CLAUSE>, <SHIP>, <VALUE.SPEC>, THE
end-of-list

11_define (give status kitty hawk)
like (list the employment schedule, state of readiness, commanding
officer and position of kitty hawk)
PARSED!
        * This is an example of the paraphrase feature of LIFER.  A
        * new pattern is defined by example.
Parse time: .705 seconds
        * The system answers the query as a side-effect of parsing
        * the paraphrase.
IDA: ((? ETERM) (? EBEG) (? EEND) (? READY) (? RANK) (? CONAM)
 (? PTP) (? PTD) (NAM EQ 'KITTY% HAWK'))

IDA = ((ETERM 'SURVOPS' EBEG 760103 EEND 760205 READY 2 RANK 'CAPT'
CONAM 'SPRUANCE% R' PTP '3700N01700E' PTD 7601171200))
Computation time for query: 2.725 seconds
Real time for query: 173.404 seconds
(EMPLMNT SURVOPS EMPBEG 760103 EMPEND 760205 READY 2 RANK CAPT NAME
SPRUANCE R POSITION 3700N01700E DATE 7601171200)
LIFER.TOP.GRAMMAR  =>  GIVE STATUS <SHIP>
        * The generalized pattern for the paraphrase is added to
        * the grammar.
F0086 (GIVE STATUS <SHIP>)
        * F0086 is the new LISP function created to be the response
        * expression for this pattern.
```

```
12_give status us cruisers in the mediteranean
                      spelling-> MEDITERRANEAN
PARSED!
Parse time: 2.855 seconds
IDA: ((? ETERM) (? EBEG) (? EEND) (? READY) (? RANK) (? CONAM)
(? PTP) (? PTD) (? NAM) (NAT EQ 'US') (TYPE1 EQ 'C') (TYPE2 NE 'V')
(TYPE NE 'CGO'))

IDA = ((ETERM 'CARESC' EBEG 760101 EEND 760601 READY 1 RANK 'CAPT'
CONAM 'MORRIS% R' PTP '4000N00600E' PTD 7601171200 NAM 'CALIFORNIA')
(ETERM 'CARESC' EBEG 751231 EEND 760615 READY 1 RANK 'CAPT' CONAM
'HARMS% J' PTP '3700N01700E' PTD 7601171200 NAM 'DANIELS% J') ...)
Computation time for query: 3.738 seconds
Real time for query: 195.698 seconds

EMPLMNT:  CARESC       CARESC       CARESC       CARESC
EMPBEG:   760101       751231       751231       751231
EMPEND:   760601       760615       760615       760615
READY:    1            1            1            1
RANK:     CAPT         CAPT         CAPT         CAPT
NAME:     MORRIS R     HARMS J      EVANS O      FRENZINGER T
POSITION: 4000N00600E  3700N01700E  3700N01700E  3700N01700E
DATE:     7601171200   7601171200   7601171200   7601171200
SHIP:     CALIFORNIA   DANIELS J    WAINWRIGHT   JOUETT

        {information about 8 other ships omitted}

13_done
PARSED!
        * The user indicates that he is finished with the session.
File closed  8-Nov-77 11:11:17
Thank you
@
```

ACKNOWLEDGMENTS

Our debugging of LIFER and the continuing development of human engineering features have been strongly influenced by interactions with interface builders. In particular, we would like to thank the following people for using the LIFER system extensively and sharing their experiences with us: Staffan Lof (Swedish version of LADDER and extensions to INLAND), Martin Epstein (medical database system of melanoma cases), and Harry Barrow and Keith Lantz (interactive aid for cartography and photo interpretation). We would also like to thank Gordon Novak for recent revisions to the elliptical processor.

REFERENCES

1. Brown, J.S., and Burton, R.R. Multiple representations of knowledge for tutorial reasoning. In *Representation and Understanding*, D.G. Bobrow and A. Collins, Eds., Academic Press, New York, 1975, pp. 311–349.
2. Burton, R.R. "Semantic grammar: An engineering technique for constructing natural language understanding systems. BBN Rep. 3453, Bolt, Beranek, and Newman, Boston, Mass., Dec. 1976.
3. Codd, E.F. Seven steps to rendezvous with the casual user. In *Data Base Management*, J.W. Klimbie and K.I. Koffeman, Eds., North-Holland, Amsterdam, 1974, pp. 179–200.
4. Computer Corporation of America. Datacomputer version 1 user manual. CCA, Cambridge, Mass., Aug. 1975.
5. Erman, L.D., Ed. ACM SIGART Newsletter *61*, Feb. 1977.
6. Farrell, J. The Datacomputer—A network data utility. Proc. Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, Calif., May 1976 pp. 352–364.

7. GROSZ, B.J. The representation and use of focus in dialog understanding. Ph.D. dissertation, U. of California, Berkeley, May 1977.
8. HARRIS, L.R. ROBOT: A high performance natural language processor for data base query. ACM SIGART Newsletter *61*, Feb. 1977, pp. 39–40.
9. HARRIS, L.R. User oriented data base query with the ROBOT natural language query system. Proc. 3rd Int. Conf. on Very Large Data Bases, Tokyo, Japan, Oct. 6–8, 1977.
10. HENDRIX, G.G. The LIFER manual: A guide to building practical natural language interfaces. Tech. Note 138, SRI Artificial Intelligence Center, Menlo Park, Calif., Feb. 1977.
11. HENDRIX, G.G. Human engineering for applied natural language processing. Proc. 5th Int. Joint Conf. on Artificial Intelligence, Cambridge, Mass., Aug. 1977.
12. HOPCROFT, J.E., AND ULLMAN, J.D. *Formal Languages and Their Relation to Automata*. Addison-Wesley, Reading, Mass., 1969.
13. MORRIS, P., AND SAGALOWICZ, D. Managing network access to a distributed data base. Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, Calif., May 1977.
14. MYLOPOULOS, J., BORGIDA, A., COHEN, P., ROUSSOPOULOS, N., TSOTSOS, J., AND WONG, H. TORUS—A natural language understanding system for data management. Proc. 4th Int. Joint Conf. on Artificial Intelligence, Tbilisi, U.S.S.R., Aug. 1975.
15. PAXTON, W.H. A framework for speech understanding. Tech. Note 142, SRI Artificial Intelligence Center, Menlo Park, Calif., June 1977.
16. SACERDOTI, E.D. Language access to distributed data with error recovery. Proc. 5th Int. Joint Conf. on Artificial Intelligence, Cambridge, Mass, Aug. 1977.
17. SAGALOWICZ, D. IDA: An intelligent data access program. Proc. 3rd Int. Conf. on Very Large Data Bases, Tokyo, Japan, Oct. 1977.
18. SOWA, J.F. Conceptual graphs for a database interface. *IBM J. Res. Develop. 20*, 4 (July 1976), 336–357.
19. TEITELMAN, W. INTERLISP reference manual. Xerox PARC, Palo Alto, Calif., Dec. 1975.
20. THOMPSON, F.B., AND THOMPSON, B.H. Practical natural language processing: The REL system as prototype. In *Advances in Computers 13*, M. Rubinoff and M.C. Yovits, Eds., Academic Press, New York, 1975.
21. WALKER, D.E., GROSZ, B.J., HENDRIX, G.G., PAXTON, W.H., ROBINSON, A.E., AND SLOCUM, J. An overview of speech understanding research at SRI. Proc. 5th Int. Joint Conf. on Artificial Intelligence, Cambridge, Mass., Aug. 1977.
22. WALTZ, D. Natural language access to a large data base: An engineering approach. Proc. 4th Int. Joint Conf. on Artificial Intelligence, Tbilisi, U.S.S.R., Sept. 1975, pp. 868–872.
23. WOODS, W.A. Transition network grammars for natural language analysis. *Comm. ACM 13*, 10 (Oct. 1970), 591–606.
24. WOODS, W.A. An experimental parsing system for transition network grammars. In *Natural Language Processing*, R. Rustin, Ed., Algorithmics Press, New York, 1973.
25. WOODS, W.A., BATES, M., BROWN, G., BRUCE, B., COOK, C., KLOVSTAD, J., MAKHOUL, J., NASH-WEBBER, B., SCHWARTZ, R., WOLF, J., AND ZUE, V. Speech understanding systems, Final technical progress report. Tech. Rep. 3438, Bolt, Beranek, and Newman, Cambridge, Mass., Dec. 1976.
26. WOODS, W.A., KAPLAN, R.M., AND NASH-WEBBER, B. The lunar sciences natural language information system. BBN Rep. 2378, Bolt Beranek and Newman, Cambridge, Mass., 1972.

Received June 1977; revised November 1977