

SRI International



HUMAN ENGINEERING FOR APPLIED NATURAL
LANGUAGE PROCESSING

Technical Note 139

February 1977

By: Gary G. Hendrix
Artificial Intelligence Center

SRI Project 740D32 CTC

The work reported herein was conducted under
SRI's Internal Research and Development Program.

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

ABSTRACT

Human engineering features for enhancing the usability of practical natural language systems are described. Such features include spelling correction, processing of incomplete (elliptical) inputs, interrogation of the underlying language definition through English queries, and an ability for casual users to extend the language accepted by the system through the use of synonyms and paraphrases. All of the features described are incorporated in LIFER, an applications-oriented system for creating natural language interfaces between computer programs and casual users. LIFER's methods for realizing the more complex human engineering features are presented.

I INTRODUCTION

This paper describes aspects of an applications-oriented system for creating natural language interfaces between computer software and casual users. Like the underlying research itself, the paper is focused on the human engineering involved in designing practical and comfortable interfaces. This focus has led to the investigation of some generally neglected facets of language processing, including the processing of incomplete inputs, the ability to resume parsing after recovering from spelling errors, and the ability for naive users to input English statements at run time that extend and personalize the language accepted by the system. The implementation of these features in a convenient package and their integration with other human engineering features are discussed.

A. HISTORICAL PERSPECTIVE

There has been mounting evidence that the current state of the art in natural language processing, although still relatively primitive, is sufficient for dealing with some very real problems. For example, Brown and Burton (1975) have developed a usable system for computer assisted instruction, and a number of language systems have been developed for interfacing to data bases, including the REL system developed by Thompson and Thompson (1975), the LUNAR system of Woods et al. (1972), and the PLANES system of Waltz (1975). The SIGART newsletter for February, 1977, contains a collection of 52 short overviews of research efforts in the general area of natural language interfaces.

There has also been a growing demand for application systems. At SRI's Artificial Intelligence Center alone, many programs are ripe for the addition of natural language capabilities, including systems for data base accessing, industrial automation, automatic programming, deduction, and judgmental reasoning. The appeal of these systems to builders and users alike is greatly enhanced when they are able to accept natural language inputs.

B. THE LIFER SYSTEM

To add natural language capabilities to a variety of existing software systems, SRI has developed a package of convenient tools, collectively called LIFER, which facilitate the rapid construction of natural language interfaces. The idea behind the LIFER system (Hendrix 1976, 1977) is to adapt existing computational linguistic technology to practical applications while extending the technology to meet human needs. These human needs are perhaps not central to the science of language but they are certainly central factors in its application. Subsequent sections of this paper present some of the human engineering features for interface users included in LIFER.* Several of the nonlinguistic features were inspired by or taken directly** from INTERLISP (Teitelman 1975), an interactive LISP programming system which is itself an excellent example of thoughtful human engineering in computer software.

II HUMAN ENGINEERING FOR INTERFACE USERS

Some of LIFER's human engineering features are exemplified in the interactions shown in Figure 1. These particular interactions involve a LIFER interface to a data base of information about employees of a university. Analogous LIFER interfaces to other types of software may also be constructed.

From a language processing view, LIFER's most interesting features are the elliptical processor, the spelling corrector, and the paraphraser. However, the usability of LIFER is influenced not so much

* The LIFER Manual (Hendrix 1977) describes how to define special-purpose languages for particular applications. It also includes information about LIFER's human engineering features for interface builders and about issues concerning applied computational linguistics.

** LIFER makes direct use of the INTERLISP spelling corrector and the USE and REDO features.

FIGURE 1: Example Interactions with LIFER

1-What is the salary of Eric Johnson?

PARSED!

(ID 327-36-8172 SALARY 19250)

2-Position and date hired?

Trying Ellipsis: WHAT IS THE POSITION AND DATE HIRED OF ERIC JOHNSON

(ID 327-36-8172 POSITION Assoc-Prof DATE-HIRED 8/1/70)

3-Of Robert Morgan

Trying Ellipsis: WHAT IS THE POSITION AND DATE HIRED OF ROBERT MORGAN

(ID 437-26-1190 POSITION Assis-Prof DATE-HIRED 6/1/73)

4-Employee 282-93-5307

Trying Ellipsis: WHAT IS THE POSITION AND DATE HIRED OF EMPLOYEE

282-93-5307

(ID 282-93-5307 POSITION Secretary DATE-HIRED 3/13/69)

5-Of professors in the computer science department

Trying Ellipsis: WHAT IS THE POSITION AND DATE HIRED OF PROFESSORS

IN THE COMPUTER SCIENCE DEPT

(ID 131-20-8462 POSITION Prof DATE-HIRED 8/1/55)

(ID 416-16-2749 POSITION Prof DATE-HIRED 8/1/74)

(ID 179-80-1360 POSITION Prof DATE-HIRED 1/1/63)

(ID 257-58-3058 POSITION Prof DATE-HIRED 15/7/71)

(ID 207-49-9271 POSITION Prof DATE-HIRED 8/1/69)

6-What is the average salary and age for math department secretaries

AVERAGE <= spelling

PARSED!

SALARY

5 items accepted

Average is 7631.4999

AGE

4 items accepted

Average is 28.1471

[Note: the birth date of one secretary
was the value N/A for not available.]

7-Maximum

Trying Ellipsis: WHAT IS THE MAXIMUM SALARY AND AGE FOR MATH DEPARTMENT

SECRETARIES

SALARY 10730

AGE 32

FIGURE 1 (continued)

8-What assistant Professors in compsci were hired after 1975

Trying Ellipsis: ELLIPSIS HAS FAILED

THE PARSER DOES NOT EXPECT THE WORD "COMPSCI" TO FOLLOW

"WHAT ASSISTANT PROFESSOR IN".

Options for next word or meta symbol are:

<DEPARTMENT-NAME> <DEPARTMENT>

9-What is a <DEPARTMENT-NAME>

PARSED!

<DEPARTMENT-NAME> may be any member of the set

{ANTHRO ANTHROPOLOGY ART BS BUSINESS ... CS ... ZOO ZOOLOGY}

<DEPARTMENT-NAME> may be any sequence of words following a pattern:

<DEPARTMENT-NAME> => BUSINESS ADMINISTRATION

COMPUTER SCIENCE

HOME ECONOMICS

.

10-Use CS for Compsci in 8

PARSED!

(ID 263-42-6062 POSITION Assist-Prof DATE-HIRED 8/1/76)

(ID 501-13-1171 POSITION Assist-Prof DATE-HIRED 6/15/76)

11-What are their names and ages

PARSED!

(ID 263-41-6062 NAME (John W. Francis) AGE 36)

(ID 501-13-1171 NAME (Martha K. Scott) AGE 28)

12-How many associate professors are there in the compsci department

Trying Ellipsis: ELLIPSIS HAS FAILED

THE PARSER DOES NOT EXPECT THE WORD "COMPSCI" TO FOLLOW

"HOW MANY ASSOCIATE PROFESSORS ARE THERE IN THE".

Options for next word or meta symbol are:

<DEPARTMENT-NAME>

13-Define compsci like CS

PARSED!

COMPSCI

14-Redo -2

PARSED!

5

FIGURE 1 (concluded)

15-List the compsci secretaries
(ID 490-81-2131)
(ID 512-37-3594)
(ID 572-15-4276)

16-Print salary Martin Devine
Trying Ellipsis: ELLIPSIS HAS FAILED
THE PARSER DOES NOT EXPECT THE WORD "SALARY" TO FOLLOW
"PRINT".
Options for next word or meta symbol are:
THE

17-Let "print salary Martin Devine" be a paraphrase of "print the
salary of Martin Devine"
PARSED!
LIFER.TOP.GRAMMAR => <PRESENT> <ATTRIBUTES> <EMPLOYEE>
finished

18-Print salary Martin Devine
PARSED!
(ID 131-20-8462 SALARY 31500)

19-Age and position
Trying Ellipsis: PRINT AGE AND POSITION MARTIN DEVINE
(ID 131-20-8462 AGE 57 POSITION Prof)

20-Show date hired compsci professors making over 30000
PARSED!
(ID 131-20-8462 POSITION Prof DATE-HIRED 8/1/55)
(ID 207-49-9271 POSITION Prof DATE-HIRED 8/1/69)

21-Let "DUMPALL Martin Devine" be a paraphrase of
"What is the ID, position, department and
salary of Martin Devine"
PARSED!
LIFER.TOP.GRAMMAR => DUMPALL <EMPLOYEE>
finished

22-Dumpall employees earning over 35000
PARSED!
(ID 122-22-8769 POSITION Prof DEPT Math SALARY 35500)
(ID 178-31-1942 POSITION Prof DEPT Physics SALARY 36000)
(ID 206-56-1620 POSITION President DEPT N/A SALARY 37500)

23-! (CONS 'THIS '(INTERACTION USES INTERLISP DIRECTLY))
(THIS INTERACTION USES INTERLISP DIRECTLY)

by the power of individual features as by the aggregate effect of having a number of features working together to support the user. It is the mix of features at various levels of complexity that should be looked for in studying the interactions of the example.

A. ENTERING AN INPUT

After INTERLISP (the language in which LIFER is currently implemented) outputs its prompt characters, the user may type in queries, commands, or assertions to the system in ordinary English.* There is no need to call the parser explicitly. Both upper and lower case are allowed, and punctuation is optional. For example, in the first line of Figure 1, the user asks the question "what is the salary of Eric Johnson?" after INTERLISP types the prompt "1-".

B. FEEDBACK

LIFER parses typical inputs, such as interaction 1, in well under a second of CPU time on the DEC PDP KL-10.** However, when the CPU is heavily loaded, users may become concerned about their inputs after even a brief delay. LIFER seeks to relieve this anxiety by providing a constant stream of feedback. For example, the CRT cursor or teletype print head follows the parsing operation as it works through an input from left to right. This feedback is an important humanizing feature, analogous to eye contact, head nodding, and beard stroking. Another feedback is that the system types the message

PARSED!

when LIFER has finished analyzing an input and is ready to call application software (i.e., the system to which LIFER is providing an interface) to answer the question, carry out the command, or assimilate the assertion communicated by the input.

* Of course, only a subset of English is actually accepted by any particular interface, but experience has shown that this subset can be designed to have wide coverage in a particular application area.

** Timings are based on a vocabulary of 1000 words and a grammar containing over 600 production rules.

C. INCOMPLETE INPUTS

If the user has just asked

WHAT IS THE SALARY OF ERIC JOHNSON

and now wishes to know Johnson's position and date hired, it is far more convenient and natural to simply ask

POSITION AND DATE HIRED

than to laboriously type out

WHAT IS THE POSITION AND DATE HIRED OF ERIC JOHNSON

Accommodating the human tendency to abbreviate inputs is an important consideration for applications systems. Although some other systems make it possible to define grammars that accept incomplete sentences as "complete" inputs,* LIFER makes this unnecessary by automatically deducing possible elliptical (i.e., incomplete) structures from the grammars supplied for complete constructions. (See interaction 2 of Figure 1.)

LIFER first attempts to parse an input as a complete sentence.** Only when this fails is elliptical analysis attempted. To give the user feedback concerning this shift in operations, LIFER types the message

TRYING ELLIPSIS:

when the elliptical analysis routine is invoked. If elliptical analysis is successful, then, as an additional feedback to the user, the system's expansion of the elliptical input is printed after the "TRYING ELLIPSIS:" message, replacing the "PARSED!" message printed for complete inputs.

Inputs 2 through 5 of Figure 1 are different elliptical variations on the same basic sentence pattern, the pattern of input 1. Input 2 causes a substitution for the attributes sought. Inputs 3 through 5 substitute for the individuals whose attributes are sought. Note that

* This was done, for example, in the SRI Speech Understanding System. See Walker (1976).

** But this operation may be skipped by typing a comma as the first character in an input that is only to be processed elliptically.

input 5 seeks the position and date hired for a whole class of individuals.

D. SPELLING CORRECTION

A significant consideration when dealing with human-generated inputs is that they often contain spelling errors. Whether the user actually misspells a word or simply mistypes it, the effect is the same: garbled input. In constructing a language system for the sake of studying language understanding, there is no real need for a spelling correction capability. But users of application systems are justly irritated when spelling errors cause abortion of processing and result in delays and tedious retyping.

LIFER's spelling correction ability, which makes use of INTERLISP's spelling corrector, is illustrated by interaction 6. A message is printed indicating that a spelling correction has been made, and the respelling is printed directly below the originally misspelled word.

E. ERROR MESSAGES

Interaction 8 illustrates how LIFER responds when it cannot successfully interpret an input. Having failed to parse at both the sentence level and the ellipsis level, and being unable to proceed through spelling correction, LIFER gives up and prints an error message. This error message is not such cryptic nonsense as

ERROR TRAP AT LOC 13730,

but is a piece of useful information that can help a naive user understand the problem plaguing his input and aid in a reformulation. (Interface builders may call special diagnostic routines for sophisticated error information, but that is another story.) The current error message (one of several) indicates that LIFER understood

WHAT ASSOCIATE PROFESSOR IN

but then had trouble with the word COMPSCI. It was expecting a <DEPARTMENT-NAME>.

At this point, the user may realize that COMPSCI might not be included in the system's lexicon. Another way of expressing the department name --such as COMPUTER SCIENCE-- could be tried. On the other hand, the user may be stumped, having no idea what <DEPARTMENT-NAME> is. This brings up the next topic, and interaction 9.

F. INSPECTION OF THE LANGUAGE DEFINITION

LIFER provides easy access to information about the underlying language definition through natural language. Sophisticated users and interface builders may use this mechanism to refresh their memories on the underlying structures and capabilities. Naive users, as illustrated in the last interaction, may need access to the language definition to aid in the understanding of error messages.

Interaction 9 shows one type of question that provides access to the underlying structures. The response to this input indicates both words and phrases that may be substituted for <DEPARTMENT-NAME>.

G. EXPLICIT SUBSTITUTIONS

When a user wishes to ask some simple variant of an earlier question but is not in the correct context for using ellipsis (e.g., there are intervening sentences), direct reference may be made to the earlier input, as is illustrated by interaction 10. Such references and substitutions may save typing and so reduce both the user's work and the likelihood of typing errors. This is a standard feature of INTERLISP and is not unique to LIFER.

H. PRONOMINAL REFERENCE

The resolution of anaphoric reference, especially pronouns, presents complex problems for language processing systems.* LIFER has no magic answers to these problems, but does provide facilities for handling some of the simpler cases. One such case is illustrated by interaction 11.

I. DEFINING SYNONYMS

In interaction 12, the user again attempts to use COMPSCI and again receives an error message. It may very well be that he is accustomed to using this abbreviation for computer science and does not want to adapt to any of the synonyms currently accepted by the system. Rather, he wants the system to adapt to HIS preferences. In interaction 13, the user tells the system to define COMPSCI like CS.** Henceforth, these words will be synonyms.

In interaction 14, interaction 12 is reinvoked through INTERLISP's REDO feature. This time, COMPSCI is understood. In interaction 15, COMPSCI is used in a new input.

* See Grosz (1977) for an interesting discussion of discourse problems and sophisticated mechanisms for dealing with them.

** Synonyms may also be defined through the more general concept of paraphrase. A paraphrase interaction equivalent to the use of synonyms in interaction 13 is the following:

13-Let "How many associate professors are there in the COMPSCI department" be a paraphrase of "How many associate professors are there in the CS department"

PARSED!

MAY LIFER ASSUME THAT "COMPSCI" MAY ALWAYS BE USED FOR "CS"

(TYPE YES OR NO)

YES

<DEPARTMENT-NAME> => CS

finished

14- ...

J. DEFINING PARAPHRASES

The synonym feature presented above allows LIFER to adapt to individual users by learning new words. The paraphrase feature allows LIFER to adapt to new grammatical constructions. For example, a user may grow tired of typing syntactically "correct" English queries and wish to use an abbreviated format. In interaction 16, the user attempts to use a condensed format and is confronted with an error message. In interaction 17, an ordinary English construction is employed to tell the system that the abbreviated form is henceforth to be accepted as legitimate. LIFER analyzes the specific paraphrase it has been given as an example, seeking to generalize the paraphrasing to other cases. (More will be said about this later.) Production rules showing the results of this generalization are printed for the benefit of the more sophisticated user.

In interaction 18, the new abbreviated format is tested. Interaction 19 illustrates an elliptical expansion based on the user-defined format. Interaction 20 illustrates the fact that LIFER has generalized the original paraphrase example to cover other abbreviated constructions that are similar.

Interactions 21 and 22 provide further illustrations of LIFER's paraphrase ability. Through interaction 21,

DUMPALL x

comes to have the meaning

INDICATE THE ID, POSITION, DEPARTMENT, AND SALARY OF x

K. ACCESSING THE HOST LANGUAGE

The user who knows INTERLISP may wish to mix interactions with the LIFER parser and interactions with INTERLISP. As illustrated in interaction 23, this is easily done by preceding inputs for INTERLISP with the symbol "!".

L. PROVIDING COMFORTABLE LINGUISTIC COVERAGE

In the final analysis, the most important piece of human engineering for users is that of supplying an interface language covering the range of linguistic structures needed to communicate comfortably with the application software. Such features as spelling correction and elliptical processing, although important, can never make up for deficiencies in basic linguistic capabilities.

Given the current state of the art in language processing, it would be futile to attempt to provide a definitive specification of English having sufficient generality to cover all potential applications. LIFER's approach to adequate coverage is not to pursue a definitive specification, but rather to supply the framework, guidance, and mechanisms that allow an interface builder, in a reasonable amount of time, to create a solid, practicable, special purpose language definition, covering the spectrum of linguistic structures most relevant to a particular application.*

No attempt can be made here to detail the particular set of interactive functions that LIFER provides for specifying an application language,** but a few key points may be mentioned:

(1) Interface builders work within the framework of INTERLISP, a powerful and flexible host language with advanced debugging facilities. Lower level languages may have faster execution, but flexibility and programming ease are what count in building workable systems with reasonable amounts of effort.

(2) Extensions and modifications to the language specification may be freely mixed with calls to the parser. There is no grammar compilation phase. This allows interface builders to operate in a rapid, extend-and-test mode, and supports features that modify the language at parse time, such as the paraphraser.

* Special purpose languages are perhaps most easily created with LIFER by adopting the notion of a "semantic grammar," as advocated by Brown and Burton (1975).

** A thorough discussion of this topic is contained in The LIFER Manual (Hendrix, 1977).

(3) The interface builder is isolated from the internal structures that LIFER builds for purposes of increasing parsing efficiency. In particular, the user communicates with LIFER in terms of simple production rules maintained internally as transition networks (Woods 1970).

(4) LIFER has a powerful grammar-editing facility (which uses the INTERLISP editor).

(5) LIFER has a package of functions for grammar interrogation and debugging.

(6) Elliptical constructions are handled automatically and so need never be considered by the interface builder.

(7) There is a reasonable manual describing how to use the system.

III IMPLEMENTATION OF SPECIAL FEATURES

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

A. IMPLEMENTATION OF SPELLING CORRECTION

LIFER uses a left-to-right parser following a simplification of the ATN system of Woods (1970). Each time the 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 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.

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 in a subgrammar that was PUSHed to in a futile attempt to follow a PUSH arc from F. Since G and F are both rightmost failpoints, G represents a stall at the start node of the PUSHed-to subgrammar. (Had any transition been made, G would be to the right of F.) Hence, if F is restarted, G is reattempted as one means of transferring from F. G, therefore, does not need to be considered independently. All dominated rightmost failpoints are dropped from consideration.

Working with the rightmost, dominating failpoints, the spelling corrector examines the associated arcs to find all categories of words that would allow a transition. (For PUSH arcs, this 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 lexical items 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 correction further to the right.) If all restarts fail, other close lexical items are substituted for the "misspelled" word. If these also fail, LIFER prints an error message.

LIFER encourages the use of semantically oriented syntactic categories, such as <EMPLOYEE> and <DEPARTMENT-NAME>, rather than such standard categories as <NOUN>. The use of these more specialized categories greatly facilitates spelling correction by severely restricting the number of possibly valid words at any point in the parse.*

* [See footnote on next page.]

B. IMPLEMENTATION OF ELLIPSIS

LIFER's mechanism for treating elliptical inputs takes advantage of the assumption that specifications for application languages tend to encode a considerable amount of semantic information in the syntactic categories. Thus, similar syntactic constructions tend 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 backwards 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 MARTIN DEVINE,

using the syntax tree** shown in Figure 2 for a previous input, WHAT IS THE SALARY OF MARTIN DEVINE? Since the MARTIN DEVINE portion of the substring is a complete expansion of <NAME>, the substring is rewritten as OF <NAME>. Similarly, since <EMPLOYEE> expands to <NAME>, the substring is rewritten as OF <EMPLOYEE>. Since no other portions of the

* An example LIFER system (described by Sacerdoti, 1977) has a vocabulary of over 1000 words, excluding numbers and coded symbols. This vocabulary is divided among 131 categories, 113 of which contain 10 or less words. 15 categories contain 11 to 50 words, and the largest contains 144.

** "PRESENT" is used in the sense of "to show for inspection."

substring are complete expansions of other syntactic categories in the tree, the process stops and OF <EMPLOYEE> 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 <EMPLOYEE>, extracted from the last input, may be used to match such current elliptical inputs as

OF ERIC JOHNSON
 OF EMPLOYEE 494-81-7207
 and OF PROFESSORS IN THE MATH DEPARTMENT

Note that the expansion of <EMPLOYEE> need not parallel its expansion in the old input that originated the analogy pattern. For example, OF EMPLOYEE 494-81-7207 is not matched by expanding <EMPLOYEE> to <NAME> but by expanding <EMPLOYEE> to EMPLOYEE <ID-NUMBER>.

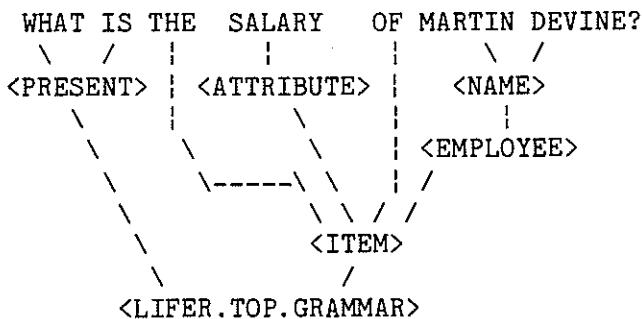


FIGURE 2: A Syntax Tree

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

<ITEM> => THE <ATTRIBUTE>

is invoked, using the new value of <EMPLOYEE> (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 <LIFER.TOP.GRAMMAR>, 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 SALARY
pattern:   THE <ATTRIBUTE>
a match:  THE AGE AND DATE HIRED

substring: SALARY OF MARTIN DEVINE
pattern:   <ATTRIBUTE> OF <EMPLOYEE>
a match:  AGE OF CS SECRETARIES

substring: WHAT IS THE SALARY
pattern:   <PRESENT> THE <ATTRIBUTE>
a match:  PRINT THE DATE HIRED

substring: WHAT IS THE SALARY OF MARTIN DEVINE
pattern:   <LIFER.TOP.GRAMMAR>
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.* Nevertheless, the effect is conceptually equivalent to attempting parses based on the analogy patterns of each of the contiguous substrings of the last input.

C. 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.

* [See footnote on next page.]

1. The Basic Method

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

* [Footnote from last page.] Abstractly, the actual algorithm is as follows. If the last input was parsed by the top-level production

$$\langle \text{LIFER.TOP.GRAMMAR} \rangle \Rightarrow \langle X_1 \rangle \langle X_2 \rangle \dots \langle X_N \rangle$$

then elliptical processing begins by attempting to match the new input to the left portion of the right side of this production. If the new input matches $\langle X_1 \rangle \dots \langle X_j \rangle$, leaving $\langle X_{j+1} \rangle \dots \langle X_n \rangle$ unused, then $\langle X_{j+1} \rangle \dots \langle X_n \rangle$ are assumed from the old input. If the new input does not match the left portion of the pattern $\langle X_1 \rangle \dots \langle X_n \rangle$, then the process restarts, using the left-truncated pattern $\langle X_2 \rangle \dots \langle X_n \rangle$. In general, if the new input matches subpattern $\langle X_i \rangle \dots \langle X_j \rangle$, then the old $\langle X_1 \rangle \dots \langle X_{i-1} \rangle$ and $\langle X_{j+1} \rangle \dots \langle X_n \rangle$ are used to expand the elliptical input into a new top-level sentence.

The process is complicated by the fact that any of the $\langle X \rangle$ may itself have been expanded in the last input by a production

$$\langle X \rangle \Rightarrow \langle Y_1 \rangle \langle Y_2 \rangle \dots \langle Y_m \rangle$$

If the new input does not account for $\langle X_i \rangle$ when attempting the match $\langle X_i \rangle \dots \langle X_n \rangle$, then $\langle Y_1 \rangle \dots \langle Y_n \rangle$ is substituted for $\langle X_i \rangle$, with the hope that the elliptical input may begin somewhere in the middle of the expansion of the old $\langle X_i \rangle$. Only after the $\langle Y \rangle$ have been exhausted by left truncation will $\langle X_{i+1} \rangle$ become the left-most symbol for a matching attempt. Similarly, if $\langle X_i \rangle \dots \langle X_{i+m} \rangle$ has accounted for the left portion of an elliptical input, but $\langle X_{i+m+1} \rangle$ does not match the left part of the remainder of the input, then the expansion of $\langle X_{i+m+1} \rangle$, taken from the last input, is substituted for $\langle X_{i+m+1} \rangle$ and the match continues. As sometimes happens, the elliptical input may end somewhere in the middle of the expansion of $\langle X_{i+m+1} \rangle$.

a. 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 2 are

1. <ITEM> THE SALARY OF MARTIN DEVINE
2. <PRESENT> WHAT IS
3. <NAME> MARTIN DEVINE --not used
4. <EMPLOYEE> MARTIN DEVINE
5. <ATTRIBUTE> SALARY

Since the syntax tree indicates <EMPLOYEE> => <NAME> => MARTIN DEVINE, both <NAME> and <EMPLOYEE> account for the same subphrase. For such cases, only the most general syntactic category (<EMPLOYEE>) is considered.

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 2 is

FOR MARTIN DEVINE GIVE ME THE SALARY

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 <EMPLOYEE> GIVE ME THE SALARY

Subphrase 5 matches a sequence of words in this updated paraphrase string. The associated substitution yields

FOR <EMPLOYEE> GIVE ME THE <ATTRIBUTE>

Since there are no more subphrases to try, the structure

<LIFER.TOP.GRAMMAR> => FOR <EMPLOYEE> GIVE ME THE <ATTRIBUTE>

is created as a new production to account for the paraphrase.

b. Defining a Response Function for the Paraphrase Production

A new semantic function 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 a function for computing the interpretation of associated subphrases from subphrase constituents. The paraphraser reuses selected functions of the model to create a new function for the paraphrase production. The manner in which this is done is best illustrated by example.

Continuing the example of Figure 2, the syntax tree indicates that the production

<LIFER.TOP.GRAMMAR> => <PRESENT> <ITEM>

was used. Associated with this production is a function F1 (not shown in the figure, but referenced in the actual tree) that computes a value for <LIFER.TOP.GRAMMAR> from the values of <PRESENT> and <ITEM>.* Using the notation "#<X>" to indicate "the value of <X>," the role of F1 may be expressed by the equation

#<LIFER.TOP.GRAMMAR> = F1(#<PRESENT>, #<ITEM>)

Another production indicated by the model syntax tree is

<ITEM> => THE <ATTRIBUTE> OF <EMPLOYEE>

This production is associated with a function F2, where

#<ITEM> = F2(#<ATTRIBUTE>, #<EMPLOYEE>)

* Since <LIFER.TOP.GRAMMAR> is the sentence-level syntactic category, this value is, in fact, the response to the total input.

The paraphraser must define a new function FN for the paraphrase production

LIFER.TOP.GRAMMAR => FOR <EMPLOYEE> GIVE ME THE <ATTRIBUTE>

Moreover, the value computed by FN must be the same as the value computed as a response to the model sentence. Since the categories <EMPLOYEE> and <ATTRIBUTE> appear on the right side of the paraphrase production, the paraphraser assumes that FN is a function of #<EMPLOYEE> and #<ATTRIBUTE>. Since FN must produce the same value as produced by the model call to F1, the paraphraser assumes that

$$FN(\#<EMPLOYEE>, \#<ATTRIBUTE>) = F1(\#<PRESENT>, \#<ITEM>)$$

The syntax tree indicates that the expansion of <PRESENT> is independent of the expansions of <EMPLOYEE> and <ATTRIBUTE>. Hence, the paraphraser assumes #<PRESENT> to be a constant in the computation of FN. That is, the value of <PRESENT> used in the model will always be used as the value of <PRESENT> in computing FN in terms of F1.

In contrast, the syntax tree shows <ITEM> to incorporate both <EMPLOYEE> and <ATTRIBUTE>. Hence, both of these parameters to FN may influence #<ITEM>. Function F2 indicates the nature of this influence. Therefore, in the equation defining FN, the paraphraser replaces #<ITEM> by the expression that computes it:

$$FN(\#<EMPLOYEE>, \#<ATTRIBUTE>) = F1(\#<PRESENT>, F2(\#<ATTRIBUTE>, \#<EMPLOYEE>))$$

This new equation completely specifies FN in terms of constants, formal parameters of FN, and previously defined functions. That is, FN is defined in terms of the constant #<PRESENT> (taken from the original model input), the formal parameters #<EMPLOYEE> and #<ATTRIBUTE>, and the previously defined functions F1 and F2.

2. Greater Generalization

The goal of the paraphrase routine is to account for the paraphrase in the most general terms possible, so that new constructions created to account for a particular paraphrase will cover a maximum

number of new input possibilities. For certain cases, the coverage produced by the basic method presented above is extended by the paraphraser as follows. Suppose some model subphrase S that matches a substring of the paraphrase is associated with the syntactic unit <M> in the model syntax tree. Such an <M>, in turn, will appear in the tree as a direct component of a more general unit <G> such that

$$\langle G \rangle \Rightarrow x \langle M \rangle y$$

where x and y are some (possibly empty) sequences of linguistic units. Since the subphrase for <G> itself was not matched in the paraphrase, either the x or the y or both did not appear in the paraphrase (at least not in the necessary juxtaposition to <M>). Nevertheless, if the grammar allows the production

$$\langle G \rangle \Rightarrow \langle M \rangle$$

and if the value assigned to <G> is the same for both

$$\langle G \rangle \Rightarrow \langle M \rangle$$

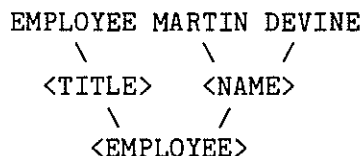
$$\text{and } \langle G \rangle \Rightarrow x \langle M \rangle y$$

then <G> is substituted for <M> in the paraphrase to produce a construction with broader coverage.

For example, suppose that the model input is

WHAT IS THE SALARY OF EMPLOYEE MARTIN DEVINE

and that the syntax tree is like that of Figure 2 except that <EMPLOYEE> expands as



Suppose further that the paraphrase is again

FOR MARTIN DEVINE GIVE ME THE SALARY

Unlike the earlier example in which <EMPLOYEE> was substituted for MARTIN DEVINE, the substitution algorithm of the last section now only allows <NAME> to be substituted. The resultant paraphrase is

FOR <NAME> GIVE ME THE <ATTRIBUTE>

This structure accounts for the given paraphrase, but not for

FOR PROFESSOR MARTIN DEVINE GIVE ME THE AGE

However, using the generalization process just outlined, if the system allows

<EMPLOYEE> => <NAME>

and if the value of <EMPLOYEE> defined in this fashion is the same as the value using

<EMPLOYEE> => <TITLE> <NAME>

then <EMPLOYEE> will be substituted for <NAME> in the paraphrase structure to produce

FOR <EMPLOYEE> GIVE ME THE <ATTRIBUTE>

This more general construction accounts for the inputs

FOR PROFESSOR MARTIN DEVINE GIVE ME THE AGE

FOR EMPLOYEE 205-56-1620 GIVE ME THE DATE HIRED

FOR MATH DEPARTMENT SECRETARIES GIVE ME THE SALARY

3. Confinement to Subgrammars

Consider paraphrases of the form "x y z", where the model is of the form "x S y" and S is a proper subphrase associated with a syntactic category <C>. The paraphraser traps this type of condition and asks the user if y is always a paraphrase of S or is simply a paraphrase in the context of x and z. If the user indicates a context dependency, then processing proceeds as usual. If the user indicates that y is a paraphrase of S in every context, then LIFER will make y a paraphrase of S in the subgrammar accounting for <C>. The influence of this paraphrase will then be felt everywhere that category <C> is used. (For example, see footnote of section II-I.)

IV CONCLUDING REMARKS

During the last year, a number of interfaces have been constructed using LIFER, and the response from users has been enthusiastic. It is worth noting that interfaces for several of the simpler applications took less than a week to create. Most of these simple interfaces were to small, relational data bases. However, interfaces were also constructed for a task scheduling and resource allocating system, a computer-based expert system, and a program that answers questions about the relationships between procedures in a large body of computer code.

LIFER has also been used in creating more ambitious interfaces. One of these, developed with several man-months (but not several man-years) of effort, is the INLAND component of the LADDER system described by Sacerdoti (1977). This system, which incorporates a grammar with over 600 "productions" and a lexicon with over 1000 words (not to mention numbers and numerous coded symbols), provides natural language access to a relatively large collection of data that is distributed among multiple remote computers on the ARPA net.

In summary, the experience with LIFER indicates that genuinely useful natural language interfaces can be created and that the creation process takes considerably less effort than might be expected. Human engineering has played a key role in making this possible. The application of similar engineering to more sophisticated language processing technology, such as that developed in the SRI Speech Understanding Project (Walker 1976), promises to produce practical systems having much greater fluency in their user's natural language.

REFERENCES

- Brown, J. S. and Burton, R. R. Multiple Representations of Knowledge for Tutorial Reasoning. In Bobrow, D. G. and Collins, A. (Eds.) Representation and Understanding, Academic Press, New York, 1975, 311-349.
- Grosz, Barbara J. The Representation and Use of Focus in Dialogue Understanding. Ph.D. Thesis, University of California, Berkeley, California, June 1977.
- Hendrix, G. G. LIFER: A Natural Language Interface Facility. Technical Note 135, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, 1976.
- Hendrix, G. G. The LIFER Manual: A Guide to Building Practical Natural Language Interfaces. Technical Note 138, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, 1977.
- Sacerdoti, E. D. Language Access to Distributed Data with Error Recovery. Adv. Papers of 5th Intl. Joint Conf. on Artificial Intelligence, Cambridge, Massachusetts, August 1977.
- Teitelman, W. INTERLISP Reference Manual. XEROX Palo Alto Research Center, Palo Alto, California, 1975.
- Thompson, F. B. and Thompson, B. H. Practical Natural Language Processing: The REL System Prototype. In Rubinfeld, M. and Yovits, M. C. (Eds.) Advances in Computers, Academic Press, New York, 1975, 109-168.
- Walker, D. E. (Ed.) Speech Understanding Research. Annual Report, Project 3804, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, 1976
- Waltz, D. L. Natural Language Access to a Large Data Base: An Engineering Approach. Adv. Papers 4th Intl. Joint Conf. on Artificial Intelligence, Tbilisi, U.S.S.R., September 1975, 868-872.
- Woods, W. A. Transition Network Grammars for Natural Language Analysis. CACM 13, 10, October 1970, 591-606.
- Woods, W. A., Kaplan, R. M., and Nash-Webber, B. The Lunar Sciences Natural Language System: Final Report. Report No. 2378, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, 1972.