HUMAN ENGINEERING FCR APPLIED NATURAL LANGUAGE PROCESSING

Gary G. Herdrix

Artificial Intelligence Carter
SRI INTERNATIONAL
Mmlo Park, California 94025

ABSTRACT

Human engineering features for enhancing the usability of practical natural language systems a^{l}re described. Such features include spelling correction, processing *of* incomplete (elliptic-~l) input?, jntfrrog-tior of th^{p} underlying language definition through English oueries, and ?r rbil.it y for casual users to extrnd the language accepted by the system through the- use of synonyms ana peraphrases. All of 1 h* features described are incorporated in LJFER, -"n r pplieations-orj e nlf d system for^{1} creating natural language j nte*rfaees* between computer programs and casual USERS LJFER's methods for r<"v] izir? the mroe complex human enginering features ? re presented.

1 INTRODUCTION

This pape r depcribes aspect r of a n applieations-oriented system for creating natural langruage interfaces between computer software and Casual users. Like the underlying researen itself, the paper is focused on the human engineering involved in designing practical rnd comfortable interfaces. This focus has lead to the investigation of some generally neglected facets of language processing, including the processing of Ireomplfte inputs, the ability to resume parsing after recovering from spelling errors *a*nd the ability for naive users to input English stat.emert s at run time that, extend and person-lize 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 ol Walt7 (1975). The SIGART newsletter for February, 1977, contains a collection cf 5? short overviews of research efforts in the general area of natural language interfaces.

Tnere has rise been a growing demand for application systems. At SRi's Artificial Irtellugene Center alone, many programs are ripe for the addition of language capabilities, Including systems for data base accessing, industrial automation, automatic programming, deductior, and judgmental reasoning. The appeal cf these systems to builders ana users .-'like 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 reftware systems, SRI has C'velopcd a package of convenient tools, co.llectively called LIFER, which facilitate the rapid construction of natural language interfaces, The Idea behind the LIFER system (Hendrix 19/6, 1977) is to adapt existing computational linguistie technology tc practical applications while extending the technology to rreet 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 cr taken directly** from INTERLISP (Teitelmen 1975), an interactive LISP programming system which is itself an excellent example of thoughtful human engineering in computer software.

11 HUMAN ENGINEERING FOR INTERFACE USERS

Some of LIFER'S human engineering features *ere* 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 *cf* software may also be construeted.

From *s* language processing view, LIFER's most interesting features are the elliptical processor, the spelling corrector, and the paraphraser.

* 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 end 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 Aasoc-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 282-93-5 307
(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          [Note:  the birth date of one
Average is 28.1471        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

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 A8SISTANT 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
     one of the patterns:
     <DEPARTMENT-NAME> ->   BUSINESS ADMINISTRATION
                            COMPUTER SCIENCE
                            HOME ECONOMICS
```

```
10-Use CS for Compsci in 8
PARSED!
(ID 263-42-6062 POSITION Aesist-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

15-List the compsci secretaries
PARSED!
(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 <EMPL0YEE>
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)
```

However, the usability of LIFER is influfnced net so much by the poner of individual features -s by the aggregate effect of having a number of festunes working together to support the user, it if 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 1NTERLISP (the language in which LIFER is currently implemented) outputs its. prompt characters, the user may type in queries, commands, or assertions to the system ir ordinary English.* There is no needd to call the parser explicitly. Both upper and lower ease 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?" a.fter INTERLISP types the prompt "I-".

h. FEEDBACK

LIFER pnrses typical inputs, such ?s 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 d^iry. LIFER seeks to relieve this anxiety by providing a constant stream of feedback. For example, the CRT cursor cr teletype print head follows the parsing operation as it works through an input from left to right. This feedback is rn important humanizing feature, analogous to eye contact, he?jd 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.

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 autcmnticaiiy deducing possible elliptical (i.e., incomplete) structures from the grammars suppliedfer complete constructions. (See interaction ? of Figure 1.)

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

TRYING ELLIPSIS:

when the elliptic-el analysis routine is invoked, if elliptical analysis is successful, then, as an additional feedback to the user, the system's exprnsion of tne 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 besic sentence pattern, the pattern of input 1. input 2 causes a substitution for the attributes sought. Inputs '* through 5 substitute for the individuals whose attributes *are* sought. Note that input 5 seeks the position and date hired for a whole class of individuals.

A significant consideration when dealing with humen-gencrated inputs is that they often contain spelling errors. Whether the *user* actually misspells *h* word or[1] simply mistypes it, the effect is the same: garbled input. In constructing a. language system for the sake ol 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 ard result in delays and tedious retyping.

LIFER'S spelling correction ability, which maikes use cf IN'IERLISP'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.ERRORMESSAGESaction8illustrateshowLIFERresponds when it cannot successfully interpret an input. Having failed to parse at both the sentence level and the ellipsis level, and being unable[5] 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

* Of course, only ? 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 *v* particular application area.

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

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

***» Eut this operation may be skipped by typing a comma as the first character in an input that is only to be processed elliptic-ally.

another story.) The currert *error message* (ore of several) indicates that LIFER understood whatl ASSOCIATE PROFESSOR IN but then had trouble with ♦ the word compsct It was expecting'DEPARTMENTNAMEAtthispoint,theusermAyrealizethat COKPSCI might net be included in the system's lexicon. Another way of expressing the dcpartmentl name --such is COMPUTER SCIENCE— could be tried. On the other hand, the user may be stumped, having re ides what <DE PARTMFMT-NAr-^ is. This brirgs up the next topic, and interaction 9.

F. inspection OF THE LANGUAGE DEFINATION

LIFER provides easy a'ccess to information . about the underlying language definition through natural language. Sophisticated users ? nd irterface builders rrry usr this mechanism to refresh their memerle* on the uncerlying structures and . naive users . - r i 1 i *vz* t r ' ten j *r.* the last intera rt ion, may peed acess tc the language definition to 'id ir the undr ct *error messages* .

interactior (*j* shows ore type of question that provioes reeses to the underlying structures. The response to this input inaiertes both vords nro phrrsrs thrt rrr-y be substituted lor <DEPARTMENT-NAKE>.

G. EXFLICiT SUBSTITIONS

When *r* user wishes to ASk some simple variant of An errlior ouestior but is rot in the correct context for using ellipsis (*.g., there rre intervening sentences), direct reference may be m.ade tc the- crrlier input, as is jllustre ted by interaction 10. Such references ;<nd substitutions *may save* typing and. so redues both the user's work work the likelihood of typing errors, This is a strnderd fepturc of 1NTERLISP rnd is not unique to LIFER.

H. PPONCKWL REFERENCE

The resolution of ANAPHOIC reference, especially pronouns, presents complex problems for LANGUAGE processing systems.* LIFER has no megic answers to these problems, but does provide frailities for handling seme of the simpler casts. One such case is illustrated by irteruction 11.

1. DEFINING SYNONYMS

In interaction 12, the user agAin attempts to use C0MPSC1 nnd again receives an error messege. It may very well be that he is accustomed to using this abbreviation for computer science rnd does not want tc adept to any of the synonyms currently eccepted by the system. Rather, he wants the system to adapt, to HIs preferences. In interaction 13, the user tells the system tc define C0MPSC1 like CS.** Henceforth, these vords will be synonyms.

Ir. interaction 1^, internet icn 12 is reirvokeo through INIERLiSP's RtDO feature. Tnis time, CO:P.St I is under stood. ir interaction 1*, COt.PSCI is usee in ? new input.

j. DEFINING PARAPHRASES

'The synonym feature prc.se ntec above allcws LiFFR to -drpt to individual users by Jeanning new words. Tne para phrase feature allows LIFER to acapt to new grrmmatical constructions. For example , a user may grow tired of typing syntactically "correct" English gieroes and wish to use ;,r -bbreviPteci format. in interacttor 16, the user attempts to use r ocndirsed format and is confronted vith a n error massage In interaction 17, an ordinary English const ruction is employed to tell the system that the rbbreviated form is ne neeforth tc be accepted as legi t imate . LIFER analyzes the spec i fie parrphr-sc it has been given asanexrmple,seekingtoa pr r:-phr'■ s in.g to other ases . (Recre Will be said rbcut this later-.) Prooucticn rules shewing the results of this generalization are printed for the be re f of t he more so pn i s t i c rt ed user .

In interaction al the new abbreviated forra?t Is tested. interactior. 19 illustr'tes an elliptional expasion' rsion based on the user-aef i nee formrt. Interaction 20 illustrates the foot that LJFER has genenalzed the. or-igirrl paraphrase cxample tc cover' other abbrevirted constructions that are similar.

Interoctions 21 are 22 provice further illustrations of LiFER'sprrpphr-r.seability. Through internet ion 21,

DUMPALL x

c om e s t c have t r. e meaning

1ND1CATE THE ID, POSITION,
DEPARTMENT, AND SALARY CF 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 iilustreted in interaction 2'-, this is ersily done by preceding inputs for INTERLISP with the symbol "!".

* See Grosz (1977) for rn interesting discussion of discourse problems end sophisticpted mechanisms for deling with them.

** Synonyms m:y rlso be defined through the more general concept cf paraphrse . A paraphrase irteracticn equivelent to the use of synonyms in interaction 1; is the following:

```
1?-Let "How msny associate professors are there in
     the C0MPSC1 department" bc a paraphrase cf
     "How many associate professors are there in
     the CS department"
PARSED!
KAY LIFER ASSUME THAT "COhPSCi" MAY ALWAYS BE USED
     FOR "CS"
(TYPE YES OR NO)
YES
<DEPARTMENT-NAME>  => CS
finished
m- ...
```

L. PROVJDING COMFORTABLE LINGUSTIC COVERAGE

In the fin?] ana-lysis, the most important piece of human engineering for users is thAt of supplying an interface language covering the range of linguistic structures nceded to communicate comfortably vith the application software, such features as spelling correction and elliptical procrssinc, although important, can never make up for deficiencies in basic linguistic capabilities.

Given the current state of the art in language processing, it would br futile to attempt to provide a definitive specification of English havingsufficientgeneralitytr,overailpotenttialapplications.LIFER'sapproachtoadequate cover age is not to pursue ?- definitive specification, but rather tc 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 ?. particular applieation.*

No attempt can be made here to de-tail the particular set of interactive functions that LIFER provides for specifying ?n application language,** but a few key points may be me-rticned:

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

(?) Extensions and modificptions to the language specification may be freely mixed with cfclis to the parser. There is no grammar compiltion phase. This allows interface builders to operate in a rapid, extend-and-test mode, pnd supports features thpt modify the language at. parse time, such as the parpphrpser.

(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) LJFER has a powerful grammar-editing facility (which uses the 1NTERLISP editor).

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

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

(7) There is a reasonable manual describing how tc user the system.

all IMPLEMENTATION OF SPECIAL FEATURED

This section presents An overview of LJFER's impltion of the spelling correction, elliptic] processor, and pata phraser.

A. *On 0F SPELLING CORRECTION'*

LIFER *USES A* 1eft-To-rightparserfoOWING.iriga simp! ificrtior of the ATN system of woods 1970. Each time t he pa.rser discovers that it can re longer follow transitions a long the current path, it records the failure on a faiipoint list. Each, entry on this list indicrtes the state of the system when the failure occurre-c (i.e., the position in the" transiticr. net and the values of various stacks and registers) and the curren.t position ir the input string. Local ambiguities and false paths make- it quite norm* 1 for failpoints tc be noted ever when a perfectly acceptable irput is processed.

If a complete parse is found for an input, the- failpoints are ignored. Put if an input carrot be parsed, the list, of faiipoints is used by thf spelling corrector, which selects these faiipoir.tr. 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 tht words at those positions must have beer successful for- there to be faiipoints to their- right .

The spelling corrector further restricts the rightmost faiipoints by locking for cases in which a rightmost faiipoint G is dominated by another rightmost failpoirt F. C is dominated by F if G is a faiipoint in a subgrammar that was PUSHed tc in a futile attempt to follow a PUSH *bro* from F. Since G and F *pre* both rightmost faiipoints, G represents a stall pt the start node of the PUSHe.d-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 tc be. considered independently. All dominated rightmost faiipoints are dropped from consideration.

Working with the rightmost, dominating faiipoints, 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 faiipoints is compared with the lexical items of thf categories Just found. If the "misspelled" word is sufficiently similar to any of these lexical items, the closest match is substituted. Faiipoints 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

[*] 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).

to the right.) If all restarts fail, other close lexical items are substituted for the "misspelled" word. If these also fail, LIFER prints ar error mrssage.

LIFER encourages the use of semanticaily oriented syrtactic categories, such as <EMPLCYEE> nnd <DEPAR?MENT-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 vords at any point in the parse.*

LIFER'S mechanism for treating elliptical inputs takes advantage of the assumption that specifications for application languages tend to encode a corside-rpble amount of semantic irformatior in the syntactic categories. Thus, similar syntactic constructions tend to be similar semanticnlly. LIFER'S treatment of ellipsis is based on this notion of similarity. During elliptical processing, LIFER is prepared to accept any string of vords 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.

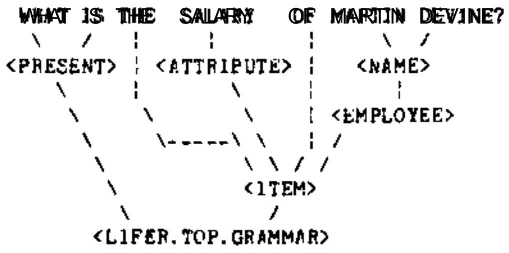

FIGURE *2:* A Syntax Tree

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

OF MARTIN DEV1NE,

using the syntax tree** shown in Figure 2 for a previous input, WHAT IS THE SALARY OF MARTIN DEV1NE Since the MARTIN DEVINE portion cf the substring is a. complete expansion of <NAME>, the substring is rewritten as OF <NAME>. Similarly, since <EMPLCYEE> expands to <NAME>, the substring is rewritten as CF <EMPLOYEE>. Since no other portions of the substring ere complete expansions of other syntactic categories in the tree, the process stops and 0F <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 <EMPL0YEE>, 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 <EMPLCYEE> 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 <EMPL0YEE> to <NAME> but by expanding <EMPLCYEE> to EMPLOYEE <ID-NUMPER>.

To compute responses for elliptical inputs matching OF <EMPLCY£E>, LIFER wcrks its way back through the old syntax tree from the common parent of OF <EMPL0YEE> toward the root. First, the routine for computing the value cf an <1TEM> from constituents of the production

<1TEM> => THE <AT'IRiEU'U>

is invoked, using the new value of <EKPL0YE£> (which appeared in the current elliptical input) and the old value of <ATTRIBUTE> from the last sentence. Then, using the newly computed value for <1TEM> and the old value for <PRESENT>, a new value is similarly computed for <L1FER.T0P.GRAMMAR>, the root of the syntax tree.

Seme other substrings with their associated analogy patterns ere shown below, along with possible new elliptical inputs matching the pat terns.

substring: THE SALARY
pattern: THE <ATTR1EUTE>
a match: THE AGE AND DATE HIRED

substring: SALARY OF MARTIN DEV1NE
pattern: <ATTR1BUTE> OF <EMPL0Y£E>
a match: AGE OF CS SECRETARIES

substring: WHAT IS THE SALARY
Pattern: <PRESENT> THE <ATTRIEUTE>
a match: PRINT THE DATE HIRED

substring: WHAT IS THE SALARY OF MARTIN DEV1NE
pattern: <L1FER.TCP.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 equivelent to attempting parses based on the analogy patterns

[11] An example LIFER system (describees by Sacerdoti, 1977) has a vocabulary of over 1000 words, excluding numbers and coded symbols. This vocabulary is divided among 1?1 categories, 113 of which contein 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."

* [Footnote is printed on next pege.]

of each of the contiguous substrings of the last input.

C. <u>IMPLEMENTATION</u> OF <u>PARAPHRA</u>SE

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

1. The Basic Method

In particular, the paraphraser invokes the parser to produce a syntax tree cf the model. Using this tree, the paraphraser determines all proper subphrases of the model, i.e., P11 substrings that are complete expansions of one of the syntactic categories listed in the tree. Any of these model subphrases that also ?ppear 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.

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 phrese.s, and for two phrases of the same length, the leftmost is taken first. For example, the sorted phrases for the tree of Figure 2 *ere*

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 DEV1NE, both <NAME> and <EKPLOYEE> account for the same subphrast. For such cases, only the most general syntactic category (<EMPLGYEE>) 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 mptch 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 matche. s a sequence of words in this updated paraphrase string. The associated substitution yields

FOR <EMPLOYEE> GIVE ME THE <ATTR1BUTE>

Since there are no more subphrases to try, the structure

<LIFER.TOP.GRAMMAR> =>
FOR <EMPLOYEE> GIVE ME THE <ATTR1BUTE>

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

b. <u>Defining</u> a <u>Response Function</u> for the Paraphrase Production,

A new semantic function indicating how to respond to inputs matching this paraphrase

----

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

<LIFER.TOP.GRAMMAR> => <X1> <X2> ... <XN>

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 <X1> ... <Xj>, leaving <Xj+1> ... <Xn> unused, then <Xj+1> ... <Xn> are assumed from the old input. If the new input does not match the left portion of the pattern <X1> ... <Xn>, then the process restarts, using the left-truncated pattern <X2> ... <Xn>. In general, if the new input matches subpattern <Xi> ... <Xj>, then the old <X1> ... <Xi-1> and <Xj+1> ... <Xn> are used to expand the elliptical input into a new top-level sentence.

The process is complicated by the fact that any of the <X> may itself have been expanded in the last input by a production

<X> => <Y1> <Y2> ... <Ym>

If the new input does not account for <Xi> when attempting the match <Xi> ... <Xn>, then <Y1> ... <Yn> is substituted for <Xi>, with the hope that the elliptical input may begin somewhere in the middle of the expansion of the old <Xi>. Only after the <Y> have been exhausted by left . truncation will <Xi+1> become the left-most symbol for a matching attempt. Similarly, if <Xi> ... <Xi+m> has accounted for the left portion of an elliptical input, but <Xi+m+1> does not match the left part of the remainder of the input, then the expansion of <Xi+m+1>, taken from the last input, is substituted for <X+m+1> and the match continues. As sometimes happens, the elliptical input may end somewhere in the middle of the expansion of <Xi+m+1>.

production is programmed automatically from information in the syntax tree of the model.]n 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 ? 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.GRAKMAR> => <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 *r.* 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 (0<PRES£NT>, <KITEM>)

Another production indicated by the model syntax tree is

<ITEM> r> THE <ATTRIBUTE> OF <EMPLOYEE>

This production is associated with a function F?, where

#<ITEM> = F?(tf<ATTRIBUTE>, /KEMPLOYEE>)

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

<L1FER.T0P.GRAMMAR> =>
FOR <EMPL0YEE> GIVE ME THE <ATTRJBUTE>

Moreover, the value computed by FN must be the same as the value computed as a response to the model sentence. Since the categories <EMPLOYE£> and <ATTRIBUTE> appear on the right side of the paraphrase production, the paraphraser assumes that FN is s function of *<EMPLOYEE> and tf<ATTR1BUTE>. Since FN must produce the same value as produced by the model call to F1, the paraphraser assumes that

FN(#<EMPLOYEE>, (KATTR1EME>) =
F1(#<PRESENT>, */<IT£M>)

The syntax tree indicates that the expansion of <PRESENT> is independent of the expansions of <EMPLOYEE> and <ATTR1BUTE>. 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 <1TEM> to incorporate both <EMPLOYEE> and <ATTRIBUTE>. Hence, both of these parameters to FN may influence #<TTEM>. Function *¥2* indicates the nature of this influence. Therefore, in the equation defining FN, the paraphraser replaces #<1TEM> by the expression that computes it:

FN(#<EMPLOYEE>, #<ATTRIBUTE>) s
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 #<EMPLCYEE> and #<ATTKIEinE>, and the previously defined functions F1 and F2.

?. Greater Generalizatiion

The goal of the paraphrase routine is to account for the prraphrrse in the most general terms possible, so thrt new constructions created to account for r 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 psraphraser as fcllcwr. Suppose some mod^l subphrase S that matches a substring of the paraphrase is associated with th« syntactic unit <m> in the model syntax tree. Such an <M>, in turn, will appear in the tree a? a direct component of a more general unit <G> such that

<G> => x <M> 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 peraphrase (at least not in the necessary juxtaposition to <M>). Nevertheless, if the grammar allows the production

<G> => <M>

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

<G> => <M>
and <G> => x <M> 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 EKPLGYEE MARTIN DEVINE

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

```
EMPLOYEE MARTIN DEVINE
   \        \     /
  <T11LE>    <NAME>
       \      /
      <EMPLOYEE>
```

Suppose further that the paraphrase is again

FOR MARTIN DEVINE GIVE ME THE SALARY

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

FOR <NAME> GIVE ME THE <ATTR1BUTE>

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 <EMPLOYE£> defined in this fashion is the same as the value using

<EMPLOYEE> => <TITLE> <NAME>

then <EMPLOY£E> will be substituted for <NAME> in the paraphrase structure to produce

FOR <EMPLOYEE> GIVE ME THE <ATTR1BUTE>

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

This more general construction recounts for the inputs

FOR PROFESSOR MARTIN DEVJ.NE GIVE ME THE AGE
FOR EMPLOYEE 205-6-1620 GIVE ME THE DATE HIRED
FOR MATH DEPARTMENT SECRETARIES GIVE ME THE SALARY

*. 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 associeted with a syntactic category <C>. The paraphraser traps this type of cordition and asks the user if y is always a paraphrase of S or is simply a paraphrase in the context of x and y. If the user indicates a context deperdern.cy, then processing proceeds as usurl. If the user indicates that y is a paraphrase of S in every context, then LIFER will rnak* y r paraphrase of £ 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 ta.sk scheduling and resource allocating system, a computer-based expert system, and a program that answers questions about the relatjonships 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 (rot 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.

ACKNOWLEDGEMENT

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

REFERENCES

Frown, J. S. and Burton, R. R. Multiple Representations of Knowledge for Tutorial Reasoning. In Pobrow, D. G. and Collins, A. (Eds.) Representation and Understanding, Academic Press, New York, 1975, ?11-349.

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 1;5, 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. 1NTERLISP Reference Manual. XEROX Palo Alto Research Center, Palo Alto, California, 1975.

Thompson, F. B. and Thompson, P. H. Practical Natural Language Processing: The REL System Prototype. In Rubincff, 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 Perk, California, 1976

Waltz, D. L. Natural Language Access to a Large Data Pase: 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 1?, 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.

Advantages of a Transformational Grammar for Question Answering

Fred J. Damerau
IBM Corporation
Thomas J. Watson Research Center
Yorktown Heights, New York

A number of researchers in artificial intelligence, for example, Woods(1975, p.88 ff.), have asserted that transformational grammars are not a satisfactory basis on which to construct natural language understanding systems, primarily because of efficiency considerations. The evidence for such a claim is by no means strong, Petrick(1976), and it can be argued that transfer of new theoretical insights into a language understanding system based on transformational grammar is facilitated, Plath(1973). This note shows that a transformational parser can also simplify problems of relating canonical representations of queries to data base representations.

Consider a data base consisting of a set of company names each with an associated list of employees. A natural question for such a data base is ^{M} How many people does company Y employ?" Our grammar produces an underlying tree structure whose bracketted terminal string is something like (I), from which a Knuth-style semantic interpreter produces a LISP form like (2).

(1) (EMPLOY (company Y) ((how many) person XI)).
(2) (SIZEOF(SETX 'XI '(TESTFCT XI (EMPLOY Y 1977))))

TESTFCT would trigger extraction of names from the data base, SETX would create a set of these names, and SIZEOF would determine the cardinality of that set. So far, this is simple enough and no difficulty arises. The first query system we constructed had a small data base of business statistics of large corporations, Plath(1973), Petrick(1973). Consider in this context a question like "What were GE's 1970 earnings?". The underlying structure was something like (3), where the semantic interpreter produced a LISP form of roughly (4).

(3) (EQUAL (the X5 (GROSS GE X5 1970)) (some amount XI)).
(4) (SETX 'XI '(FORATLEAST 1 'X7 (SETX X5 (TESTFCT X5 (GROSS GE 1970)) (EQUAL X7 XI))))

FORATLEAST implements the default quantifier, and TESTFCT finds GE's gross income. This data base also contained the total, number of employees for each company. If we were to ask

(5) How many employees does GE have?

the system would produce an underlying structure related to (1), leading to a retrieval program like (2). Unfortunately, we need a retrieval program like (4), with "EMPLOYEE" substituted for "GROSS". We could of course modify the SIZEOF function to be sensitive to the data field it dominates and return the set rather than the cardinality of the set in appropriate cases, but this is aesthetically unattractive (although this is in fact what we did in our very first system). We could also modify our translation equations and semantic interpreter so as to be sensitive to this situation. While this might be satisfactory in one or two cases, the number of special cases can become very large.

In our present application, which is an English query system for the planning files of a small city near our laboratory, there are many more situations of this kind. One can ask

(6) In what zone/planning area/ census tract/ etc., is parcel 5 located?

For each of these questions, the underlying structure has a top level verb of "LOCATED", where the translator would prefer "ZONE" or "PLANNING AREA" etc. Again, one could make the LOCATED function sensitive to its arguments, or insert the appropriate equations into the translator, but the complexity of either solution is much greater than before.

Transformational grammars customarily have two sets of rules, *cyclic* rules, which apply successively to each level of embedding, and *postcyclic* rules, which apply globally to the entire sentence. Our grammar has an additional set of rules, called *string transformations,* Plath (1974), which apply to strings of lexical trees. The transformational parsing program calls each of these sets of rules separately. Since the parser is basically a tree processor, it can be applied, via an additional set of rules, to underlying structures like those for (5) and (6), and modify the structures in such a way that the semantic interpreter can produce correct code without data base specific modifications. In the case of (5), the output of the new processing phase, called the *precycle,* is a structure like (3) instead of a structure like (1), with a data identification of "EMPLOYEE" rather than "GROSS". At the cost of an additional call to the transformational parser, we have insulated both the semantic interpreter and the data base functions from the organization of the data base, confining the necessary modifications to a single table of rules. We have not yet found a class of structural changes we wished to make because of the data base which required more than one rule. Therefore, the cost of writing new rules has been much less than the cost of generating new programs for these special cases would have been.

While I am sure other system developers are able to solve this general problem, as they must in order to proceed in their work, we have nonetheless been pleased to note that our decision to use a transformational approach on linguistic grounds has had additional benefits on practical grounds.

References:

Petrick, Stanley R. 1973. Semantic Interpretation in the REQUEST System. IBM Research Report RC 4457, IBM Corp., Yorktown His., NY.

Petrick, Stanley R. 1976. On Natural Language Based Computer Systems. IBM Journal of Research and Development, vol. 20, No. 4, pp. 314-325. July, 1976.

Plath, Warren J. 1973. Transformational Grammar and Transformational Parsing in the REQUEST System. IBM Research Report RC 4396, IBM Corp., Yorktown Hts., NY.

Plath, Warren J. 1974. String Transformations in the REQUEST System. American Journal of Computational Linguistics, Microfiche 8, 1974.

Woods, William A. 1975. Comment on a paper by Petrick, in Directions in Artificial Intelligence, R. Grishman, ed., New York University, New York, 1975.