Computational Linguistics

A. G. OETTINGER, Editor

ELIZA—A Computer Program For the Study of Natural Language Communication Between Man And Machine

JOSEPH WEIZENBAUM
Massachusetts Institute of Technology, Cambridge, Mass.*

ELIZA is a program operating within the MAC time-sharing system at MIT which makes certain kinds of natural language conversation between man and computer possible. Input sentences are analyzed on the basis of decomposition rules which are triggered by key words appearing in the input text. Responses are generated by reassembly rules associated with selected decomposition rules. The fundamental technical problems with which ELIZA is concerned are: (1) the identification of key words, (2) the discovery of minimal context, (3) the choice of appropriate transformations, (4) generation of responses in the absence of key words, and (5) the provision of an editing capability for ELIZA "scripts". A discussion of some psychological issues relevant to the ELIZA approach as well as of future developments concludes the paper.

Introduction

It is said that to explain is to explain away. This maxim is nowhere so well fulfilled as in the area of computer programming, especially in what is called heuristic programming and artificial intelligence. For in those realms machines are made to behave in wondrous ways, often sufficient to dazzle even the most experienced observer. But once a particular program is unmasked, once its inner workings are explained in language sufficiently plain to induce understanding, its magic crumbles away; it stands revealed as a mere collection of procedures, each quite comprehensible. The observer says to himself "I could have written that". With that thought he moves the program in question from the shelf marked "intelligent", to that reserved for curios, fit to be discussed only with people less enlightened than he.

Work reported herein was supported (in part) by Project MAC, an MIT research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01).

* Department of Electrical Engineering.

The object of this paper is to cause just such a re-evaluation of the program about to be "explained". Few programs ever needed it more.

ELIZA Program

ELIZA is a program which makes natural language conversation with a computer possible. Its present implementation is on the MAC time-sharing system at MIT. It is written in MAD-SLIP [4] for the IBM 7094. Its name was chosen to emphasize that it may be incrementally improved by its users, since its language abilities may be continually improved by a "teacher". Like the Eliza of Pygmalion fame, it can be made to appear even more civilized, the relation of appearance to reality, however, remaining in the domain of the playwright.

For the present purpose it is sufficient to characterize the MAC system as one which permits an individual to operate a full scale computer from a remotely located typewriter. The individual operator has the illusion that he is the sole user of the computer complex, while in fact others may be "time-sharing" the system with him. What is important here is that the computer can read messages typed on the typewriter and respond by writing on the same instrument. The time between the computer's receipt of a message and the appearance of its response is a function of the program controlling the dialogue and of such MAC system parameters as the number of users currently corresponding with the system. These latter parameters generally contribute so little to the overall response time that conversational interaction with the computer need never involve truly intolerable delays.

When in conversation with ELIZA, the user types in some statement or set of statements in natural language using normal punctuation and sentence structures. Only the question mark may not be used, because it is interpreted as a line delete character by the MAC system. The user's statement is terminated by a double carriage return which serves to turn control over to ELIZA. ELIZA then analyzes the user's statement and generates some response which it types out. Control is then again with the user.

A typical conversation is the following:

Men are all alike.
IN WHAT WAY
They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE
Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE
He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED

It's true. I am unhappy.
DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
I need some help, that much seems certain.
WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
Perhaps I could learn to get along with my mother.
TELL ME MORE ABOUT YOUR FAMILY
My mother takes care of me.
WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU
My father.
YOUR FATHER
You are like my father in some ways.
WHAT RESEMBLANCE DO YOU SEE
You are not very aggressive but I think you don't want me to notice that.
WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE
You don't argue with me.
WHY DO YOU THINK I DON'T ARGUE WITH YOU
You are afraid of me.
DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU
My father is afraid of everybody.
WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR FATHER
Bullies.
DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR BOYFRIEND MADE YOU COME HERE

The capitalized lines are the machine responses.

The gross procedure of the program is quite simple; the text is read and inspected for the presence of a *keyword*. If such a word is found, the sentence is transformed according to a *rule* associated with the keyword, if not a content-free remark or, under certain conditions, an earlier transformation is retrieved. The text so computed or retrieved is then printed out.

In detail, of course, the procedure sketched above is considerably more complex. Keywords, for example, may have a RANK or precedence number. The procedure is sensitive to such numbers in that it will abandon a keyword already found in the left-to-right scan of the text in favor of one having a higher rank. Also, the procedure recognizes a comma or a period as a delimiter. Whenever either one is encountered and a keyword has already been found, all subsequent text is deleted from the input message. If no key had yet been found the phrase or sentence to the left of the delimiter (as well as the delimiter itself) is deleted. As a result, only single phrases or sentences are ever transformed.

Keywords and their associated transformation[1] rules constitute the SCRIPT for a particular class of conversation. An important property of ELIZA is that a script is data; i.e., it is not part of the program itself. Hence, ELIZA is not restricted to a particular set of recognition patterns or responses, indeed not even to any specific language. ELIZA scripts exist (at this writing) in Welsh and German as well as in English.

The fundamental technical problems with which ELIZA must be preoccupied are the following:

(1) The identification of the "most important" keyword occurring in the input message.

(2) The identification of some minimal context within which the chosen keyword appears; e.g., if the keyword is "you", is it followed by the word "are" (in which case an assertion is probably being made).

(3) The choice of an appropriate transformation rule and, of course, the making of the transformation itself.

(4) The provision of mechanism that will permit ELIZA to respond "intelligently" when the input text contained no keywords.

(5) The provision of machinery that facilitates editing, particularly extension, of the script on the script writing level.

There are, of course, the usual constraints dictated by the need to be economical in the use of computer time and storage space.

The central issue is clearly one of text manipulation, and at the heart of that issue is the concept of the *transformation rule* which has been said to be associated with certain keywords. The mechanisms subsumed under the slogan "transformation rule" are a number of SLIP functions which serve to (1) decompose a data string according to certain criteria, hence to test the string as to whether it satisfies these criteria or not, and (2) to reassemble a decomposed string according to certain assembly specifications.

While this is not the place to discuss these functions in all their detail (or even to reveal their full power and generality), it is important to the understanding of the operation of ELIZA to describe them in *some* detail.

Consider the sentence "I am very unhappy these days". Suppose a foreigner with only a limited knowledge of English but with a very good ear heard that sentence spoken but understood only the first two words "I am". Wishing to appear interested, perhaps even sympathetic, he may reply "How long have you been very unhappy these days?" What he must have done is to apply a kind of template to the original sentence, one part of which matched the two words "I am" and the remainder isolated the words "very unhappy these days". He must also have a reassembly kit specifically associated with that template, one that specifies that any sentence of the form "I am BLAH" can be transformed to "How long have you been BLAH", independently of the meaning of BLAH. A somewhat more complicated example is given by the sentence "It seems that you hate me". Here the foreigner understands only the words "you" and "me"; i.e., he applies a template that decomposes the sentence into the four parts:

(1) It seems that (2) you (3) hate (4) me

of which only the second and fourth parts are understood. The reassembly rule might then be "What makes you think I hate you"; i.e., it might throw away the first component, translate the two known words ("you" to "I" and "me" to "you") and tack on a stock phrase (What makes you think) to the front of the reconstruction.

[1] The word "transformation" is used in its generic sense rather than that given it by Harris and Chomsky in linguistic contexts.

A formal notation in which to represent the decomposition template is:

(0 YOU 0 ME)

and the reassembly rule

(WHAT MAKES YOU THINK I 3 YOU).

The "0" in the decomposition rule stands for "an indefinite number of words" (analogous to the indefinite dollar sign of COMIT) [6] while the "3" in the reassembly rule indicates that the third component of the subject decomposition is to be inserted in its place. The decomposition rule

(0 YOU 1 ME)

would have worked just as well in this specific example. A nonzero integer "n" appearing in a decomposition rule indicates that the component in question should consist of exactly "n" words. However, of the two rules shown, only the first would have *matched* the sentence, "It seems you hate and love me," the second failing because there is more than one word between "you" and "me".

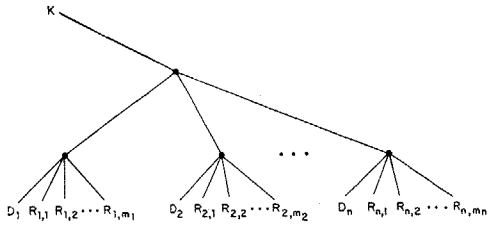

FIG. 1. Keyword and rule list structure

In ELIZA the question of which decomposition rules to apply to an input text is of course a crucial one. The input sentence might have been, for example, "It seems that you hate," in which case the decomposition rule (0 YOU 0 ME) would have failed in that the word "ME" would not have been found at all, let alone in its assigned place. Some other decomposition rule would then have to be tried and, failing that, still another until a match could be made or a total failure reported. ELIZA must therefore have a mechanism to sharply delimit the set of decomposition rules which are potentially applicable to a currently active input sentence. This is the keyword mechanism.

An input sentence is scanned from left to right. Each word is looked up in a dictionary of keywords. If a word is identified as a keyword, then (apart from the issue of precedence of keywords) only decomposition rules containing that keyword need to be tried. The trial sequence can even be partially ordered. For example, the decomposition rule (0 YOU 0) associated with the keyword "YOU" (and decomposing an input sentence into (1) all words in front of "YOU", (2) the word "YOU", and (3) all words following "YOU") should be the last one tried since it is bound to succeed.

Two problems now arise. One stems from the fact that almost none of the words in any given sentence are represented in the keyword dictionary. The other is that of "associating" both decomposition and reassembly rules with keywords. The first is serious in that the determination that a word is not in a dictionary may well require more computation (i.e., time) than the location of a word which is represented. The attack on both problems begins by placing both a keyword and its associated rules on a *list*. The basic format of a typical key list is the following:

$$\begin{array}{l}(K\ ((D_1)\ (R_{1,1})\ (R_{1,2})\ \cdots\ (R_{1,m_1}))\\ \quad\ ((D_2)\ (R_{2,1})\ (R_{2,2})\ \cdots\ (R_{2,m_2}))\\ \quad\ \ \vdots \qquad\qquad\qquad\qquad\ \ \vdots\\ \quad\ ((D_n)\ (R_{n,1})\ (R_{n,2})\ \cdots\ (R_{n,m_n})))\end{array}$$

where K is the keyword, D_i the ith decomposition rule associated with K and $R_{i,j}$ the jth reassembly rule associated with the ith decomposition rule.

A common pictorial representation of such a structure is the tree diagram shown in Figure 1. The top level of this structure contains the keyword followed by the names of lists; each one of which is again a list structure beginning with a decomposition rule and followed by reassembly rules. Since list structures of this type have no predetermined dimensionality limitations, any number of decomposition rules may be associated with a given keyword and any number of reassembly rules with any specific decomposition rule. SLIP is rich in functions that sequence over structures of this type efficiently. Hence programmin problems are minimized.

An ELIZA script consists mainly of a set of list structures of the type shown. The actual keyword dictionary is constructed when such a script is first read into the hitherto empty program. The basic structural component of the keyword dictionary is a vector KEY of (currently) 128 contiguous computer words. As a particular key list structure is read the keyword K at its top is randomized (hashed) by a procedure that produces (currently) a 7 bit integer "i". The word "always", for example, yields the integer 14. KEY(i), i.e., the ith word of the vector KEY, is then examined to determine whether it contains a list name. If it does not, then an empty list is created, its name placed in KEY(i), and the key list structure in question placed on *that* list. If KEY(i) already contains a list name, then the name of the key list structure is placed on the bottom of the list named in KEY(i). The largest dictionary so far attempted contains about 50 keywords. No list named in any of the words of the KEY vector contains more than two key list structures.

Every word encountered in the scan of an input text, i.e., during the actual operations of ELIZA, is randomized by the same hashing algorithm as was originally applied to the incoming keywords, hence yields an integer which points to the only possible list structure which could potentially contain that word as a keyword. Even then, only the tops of any key list structures that may be found there need be interrogated to determine whether or not a keyword has been found. By virtue of the various list

sequencing operations that SLIP makes available, the actual identification of a keyword leaves as its principal product a pointer to the list of decomposition (and hence reassembly) rules associated with the identified keyword. One result of this strategy is that often less time is required to discover that a given word is not in the keyword dictionary than to locate it if it is there. However, the location of a keyword yields pointers to all information associated with that word.

Some conversational protocols require that certain transformations be made on certain words of the input text independently of any contextual considerations. The first conversation displayed in this paper, for example, requires that first person pronouns be exchanged for second person pronouns and vice versa throughout the input text. There may be further transformations but these minimal substitutions are unconditional. Simple substitution rules ought not to be elevated to the level of transformations, nor should the words involved be forced to carry with them all the structure required for the fully complex case. Furthermore, unconditional substitutions of single words for single words can be accomplished during the text scan itself, not as a transformation of the entire text subsequent to scanning. To facilitate the realization of these desiderata, any word in the key dictionary, i.e., at the top of a key list structure, may be followed by an equal sign followed by whatever word is to be its substitute. Tranformation rules may, but need not, follow. If none do follow such a substitution rule, then the substitution is made on the fly, i.e., during text scanning, but the word in question is not identified as a keyword for subsequent purposes. Of course, a word may be both subtituted for and be a keyword as well. An example of a simple substitution is

(YOURSELF = MYSELF).

Neither "yourself" nor "myself" are keywords in the particular script from which this example was chosen.

The fact that keywords can have ranks or precedences has already been mentioned. The need of a ranking mechanism may be established by an example. Suppose an input sentence is "I know everybody laughed at me." A script may tag the word "I" as well as the word "everybody" as a keyword. Without differential ranking, "I" occurring first would determine the transformation to be applied. A typical response might be "You say you know everybody laughed at you." But the important message in the input sentence begins with the word "everybody". It is very often true that when a person speaks in terms of universals such as "everybody", "always" and "nobody" he is really referring to some quite specific event or person. By giving "everybody" a higher rank than "I", the response "Who in particular are you thinking of" may be generated.

The specific mechanism employed in ranking is that the rank of every keyword encountered (absence of rank implies rank equals 0) is compared with the rank of the highest ranked keyword already seen. If the rank of the new word is higher than that of any previously encountered word, the pointer to the transformation rules associated with the new word is placed on top of a list called the keystack, otherwise it is placed on the bottom of the keystack. When the text scan terminates, the keystack has at its top a pointer associated with the highest ranked keyword encountered in the scan. The remaining pointers in the stack may not be monotonically ordered with respect to the ranks of the words from which they were derived, but they are nearly so—in any event they are in a useful and interesting order. Figure 2 is a simplified flow diagram of keyword detection. The rank of a keyword must, of course, also be associated with the keyword. Therefore it must appear on the keyword list structure. It may be found, if at all, just in front of the list of transformation rules associated with the keyword. As an example consider the word "MY" in a particular script. Its keyword list may be as follows:

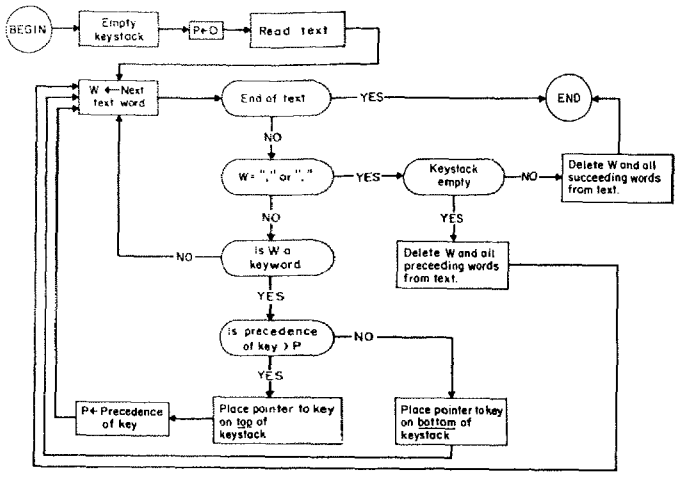

FIG. 2. Basic flow diagram of keyword detection

(MY = YOUR 5 (transformation rules)).

Such a list would mean that whenever the word "MY" is encountered in any text, it would be replaced by the word "YOUR". Its rank would be 5.

Upon completion of a given text scan, the keystack is either empty or contains pointers derived from the keywords found in the text. Each of such pointers is actually a sequence reader—a SLIP mechanism which facilitates scanning of lists—pointing into its particular key list in such a way that one sequencing operation to the right (SEQLR) will sequence it to the first set of transformation rules associated with its keyword, i.e., to the list

$$((D_1)\ (R_{1,1})\ (R_{1,2})\ \ldots\ (R_1, R_{m_1})).$$

The top of *that* list, of course, is a list which serves a decomposition rule for the subject text. The top of the keystack contains the first pointer to be activated.

The decomposition rule D_1 associated with the keyword K, i.e., $\{(D_1), K\}$, is now tried. It may fail however. For example, suppose the input text was:

You are very helpful.

The keyword, say, is "you", and $\{(D_1), \text{you}\}$ is

(0 I remind you of 0).

(Recall that the "you" in the original sentence has already been replaced by "I" in the text now analyzed.) This decomposition rule obviously fails to match the input sentence. Should $\{(D_1), K\}$ fail to find a match, then $\{(D_2), K\}$ is tried. Should that too fail, $\{(D_3), K\}$ is attempted, and so on. Of course, the set of transformation rules can be guaranteed to terminate with a decomposition rule which must match. The decomposition rule

(0 *K* 0)

will match any text in which the word *K* appears while

(0)

will match any text whatever. However, there are other ways to leave a particular set of transformation rules, as will be shown below. For the present, suppose that some particular decomposition rule (D_i) has matched the input text. (D_i), of course, was found on a list of the form

$$((D_i)(R_{i,1})(R_{i,2}) \cdots (R_{i,m_i})).$$

Sequencing the reader which is presently pointing at (D_i) will retrieve the reassembly rule $(R_{i,1})$ which may then be applied to the decomposed input text to yield the output message.

Consider again the input text

You are very helpful

in which "you" is the only key word. The sentence is transformed during scanning to

I are very helpful

$\{(D_1), \text{you}\}$ is "(0 I remind your of 0)" and fails to match as already discussed. However, $\{(D_2), \text{you}\}$ is "(0 I are 0)" and obviously matches the text, decomposing it into the constituents

(1) empty (2) I (3) are (4) very helpful.

$\{(R_{2,1}), \text{you}\}$ is

(What makes you think I am 4)

Hence it produces the output text

What makes you think I am very helpful.

Having produced it, the integer 1 is put in front of $(R_{2,1})$ so that the transformation rule list in question now appears as

$$((D_2)1(R_{2,1})(R_{2,2}) \cdots (R_{2,m_2})).$$

Next time $\{(D_2), K\}$ matches an input text, the reassembly rule $(R_{2,2})$ will be applied and the integer 2 will replace the 1. After (R_{2,m_2}) has been exercised, $(R_{2,1})$ will again be invoked. Thus, after the system has been in use for a time, every decomposition rule which has matched some input text has associated with it an integer which corresponds to the last reassembly rule used in connection with that decomposition rule. This mechanism insures that the complete set of reassembly rules associated with a given decomposition rule is cycled through before any repetitions occur.

The system described so far is essentially one which selects a decomposition rule for the highest ranking keyword found in an input text, attempts to match that text according to that decomposition rule and, failing to make a match, selects the next reassembly rule associated with the matching decomposition rule and applies it to generate an output text. It is, in other words, a system which, for the highest ranking keyword of a text, selects a specific decomposition and reassembly rule to be used in forming the output message.

Were the system to remain that simple, then keywords that required identical sets of transformation rules would each require that a copy of these transformation rules be associated with them. This would be logically sound but would complicate the task of script writing and would also make unnecessary storage demands. There are therefore special types of decomposition and assembly rules characterized by the appearance of "=" at the top of the rule list. The word following the equal sign indicates which new set of transformation rules is to be applied. For example, the keyword "what" may have associated with it a transformation rule set of the form

((0) (Why do you ask) (Is that an important question) . . .)

which would apply equally well to the keywords "how" and "when". The entire keyword list for "how" may therefore be

(How (=What))

The keywords "how", "what" and "when" may thus be made to form an equivalence class with respect to the transformation rules which are to apply to them.

In the above example the rule "(=what)" is in the place of a decomposition rule, although it causes no decomposition of the relevant text. It may also appear, however, in the place of a reassembly rule. For example, the keyword "am" may have among others the following transformation rule set associated with it:

((0 are you 0) (Do you believe you are 4) . . . (=what) . . .)

(It is here assumed that "are" has been substituted for "am" and "you" for "I" in the initial text scan.) Then, the input text

Am I sick

would elicit either

Do you believe you are sick

or

Why do you ask

depending on how many times the general form had already occurred.

Under still other conditions it may be desirable to

perform a preliminary transformation on the input text before subjecting it to the decompositions and reassemblies which finally yield the output text. For example, the keyword "you're" should lead to the transformation rules associated with "you" but should first be replaced by a word pair. The dictionary entry for "you're" is therefore:

(you're = I'm ((0 I'm 0) (PRE (I AM 3) (=YOU))))

which has the following effect:

(1) Wherever "you're" is found in the input text, it is replaced by "I'm".

(2) If "you're" is actually selected as the regnant keyword, then the input text is decomposed into three constituent parts, namely, all text in front of the first occurrence of "I'm", the word "I'm" itself, and all text following the first occurrence of "I'm".

(3) The reassembly rule beginning with the code "PRE" is encountered and the decomposed text reassembled such that the words 'I AM" appear in front of the third constituent determined by the earlier decomposition.

(4) Control is transferred, so to speak, to the transformation rules associated with the keyword "you", where further decompositions etc. are attempted.

It is to be noted that the set

(PRE (I AM 3) (=YOU))

is logically in the place of a reassembly rule and may therefore be one of many reassembly rules associated with the given decomposition.

Another form of reassembly rule is

(NEWKEY)

which serves the case in which attempts to match on the currently regnant keyword are to be given up and the entire decomposition and reassembly process is to start again on the basis of the keyword to be found in the keystack. Whenever this rule is invoked, the top of the keystack is "popped up" once, i.e., the new regnant keyword recovered and removed from the keystack, and the entire process reinitiated as if the initial text scan had just terminated. This mechanism makes it possible to, in effect, test on key phrases as opposed to single key words.

A serious problem which remains to be discussed is the reaction of the system in case no keywords remain to serve as transformation triggers. This can arise either in case the keystack is empty when NEWKEY is invoked or when the input text contained no keywords initially.

The simplest mechanism supplied is in the form of the special reserved keyword "NONE" which must be part of any script. The script writer must associate the universally matching decomposition rule (0) with it and follow this by as many content-free remarks in the form of transformation rules as he pleases. (Examples are: "Please go on", "That's very interesting" and "I see".)

There is, however, another mechanism which causes the system to respond more spectacularly in the absence of a key. The word "MEMORY" is another reserved pseudo-keyword. The key list structure associated with it differs from the ordinary one in some respects. An example illuminates this point.

Consider the following structure:

```
(MEMORY MY
  (0 YOUR 0 = LETS DISCUSS FURTHER WHY YOUR 3)
  (0 YOUR 0 = EARLIER YOU SAID YOUR 3)
  ⋮
```

The word "MY" (which must be an ordinary keyword as well) has been selected to serve a special function. Whenever it is the highest ranking keyword of a text one of the transformations on the MEMORY list is randomly selected, and a copy of the text is transformed accordingly. This transformation is stored on a first-in-first-out stack for later use. The ordinary processes already described are then carried out. When a text without keywords is encountered later and a certain counting mechanism is in a particular state and the stack in question is not empty, then the transformed text is printed out as the reply. It is, of course, also deleted from the stack of such transformations.

The current version of ELIZA requires that one keyword be associated with MEMORY and that exactly four transformations accompany that word in that context. (An application of a transformation rule of the form

(LEFT HAND SIDE = RIGHT HAND SIDE)

is equivalent to the successive application of the two forms

(LEFT HAND SIDE), (RIGHT HAND SIDE).)

Three more details will complete the formal description of the ELIZA program.

The transformation rule mechanism of SLIP is such that it permits tagging of words in a text and their subsequent recovery on the basis of one of their tags. The keyword "MOTHER" in ELIZA, for example, may be identified as a noun and as a member of the class "family" as follows:

(MOTHER DLIST (/NOUN FAMILY)).

Such tagging in no way interferes with other information (e.g., rank or transformation rules) which may be associated with the given tag word. A decomposition rule may contain a matching constituent of the form (/TAG1 TAG2 ···) which will match and isolate a word in the subject text having any one of the mentioned tags. If, for example, "MOTHER" is tagged as indicated and the input text

"CONSIDER MY AGED MOTHER AS WELL AS ME"

subjected to the decomposition rule

(0 YOUR 0 (/FAMILY) 0)

(remembering that "MY" has been replaced by "YOUR"), then the decomposition would be

(1) CONSIDER (2) YOUR (3) AGED (4) MOTHER (5) AS WELL AS ME.

Another flexibility inherent in the SLIP text manipulation mechanism underlying ELIZA is tha or-ing of matching criteria is permitted in decomposition rules. The above input text would have been decomposed

precisely as stated above by the decomposition rule:

(0 YOUR 0 (*FATHER MOTHER) 0)

which, by virtue of the presence of "*" in the sublist structure seen above, would have isolated either the word "FATHER" or "MOTHER" (in that order) in the input text, whichever occurred first after the first appearance of the word "YOUR".

Finally, the script writer must begin his script with a list, i.e., a message enclosed in parentheses, which contains the statement he wishes ELIZA to type when the system is first loaded. This list may be empty.

Editing of an ELIZA script is achieved via appeal to a contextual editing program (ED) which is part of the MAC library. This program is called whenever the input text to ELIZA consists of the single word "EDIT". ELIZA then puts itself in a so-called dormant state and presents the then stored script for editing. Detailed description of ED is out of place here. Suffice it to say that changes, additions and deletions of the script may be made with considerable efficiency and on the basis of entirely contextual cues, i.e., without resort to line numbers or any other artificial devices. When editing is completed, ED is given the command to FILE the revised script. The new script is then stored on the disk and read into ELIZA. ELIZA then types the word "START" to signal that the conversation may resume under control of the new script.

An important consequence of the editing facility built into ELIZA is that a given ELIZA script need not start out to be a large, full-blown scenario. On the contrary, it should begin as a quite modest set of keywords and transformation rules and permitted to be grown and molded as experience with it builds up. This appears to be the best way to use a truly interactive man-machine facility—i.e., not as a device for rapidly debugging a code representing a fully thought out solution to a problem, but rather as an aid for the exploration of problem solving strategies.

Discussion

At this writing, the only serious ELIZA scripts which exist are some which cause ELIZA to respond roughly as would certain psychotherapists (Rogerians). ELIZA performs best when its human correspondent is initially instructed to "talk" to it, via the typewriter of course, just as one would to a psychiatrist. This mode of conversation was chosen because the psychiatric interview is one of the few examples of categorized dyadic natural language communication in which one of the participating pair is free to assume the pose of knowing almost nothing of the real world. If, for example, one were to tell a psychiatrist "I went for a long boat ride" and he responded "Tell me about boats", one would not assume that he knew nothing about boats, but that he had some purpose in so directing the subsequent conversation. It is important to note that this assumption is one made by the speaker. Whether it is realistic or not is an altogether separate question. In any case, it has a crucial psychological utility in that it serves the speaker to maintain his sense of being heard and understood. The speaker further defends his impression (which even in real life may be illusory) by attributing to his conversational partner all sorts of background knowledge, insights and reasoning ability. But again, these are the *speaker's* contribution to the conversation. They manifest themselves inferentially in the *interpretations* he makes of the offered responses. From the purely technical programming point of view then, the psychiatric interview form of an ELIZA script has the advantage that it eliminates the need of storing *explicit* information about the real world.

The human speaker will, as has been said, contribute much to clothe ELIZA'S responses in vestments of plausibility. But he will not defend his illusion (that he is being understood) against all odds. In human conversation a speaker will make certain (perhaps generous) assumptions about his conversational partner. As long as it remains possible to interpret the latter's responses consistently with those assumptions, the speaker's image of his partner remains unchanged, in particular, undamaged. Responses which are difficult to so interpret may well result in an enhancement of the image of the partner, in additional rationalizations which then make more complicated interpretations of his responses reasonable. When, however, such rationalizations become too massive and even self-contradictory, the entire image may crumble and be replaced by another ("He is not, after all, as smart as I thought he was"). When the conversational partner is a machine (the distinction between machine and program is here not useful) then the idea of *credibility* may well be substituted for that of *plausibility* in the above.

With ELIZA as the basic vehicle, experiments may be set up in which the subjects find it credible to believe that the responses which appear on his typewriter are generated by a human sitting at a similar instrument in another room. How must the script be written in order to maintain the credibility of this idea over a long period of time? How can the performance of ELIZA be systematically degraded in order to achieve controlled and predictable thresholds of credibility in the subject? What, in all this, is the role of the initial instruction to the subject? On the other hand, suppose the subject is told he is communicating with a machine. What is he led to believe about the machine as a result of his conversational experience with it? Some subjects have been very hard to convince that ELIZA (with its present script) is *not* human. This is a striking form of Turing's test. What experimental design would make it more nearly rigorous and airtight?

The whole issue of the credibility (to humans) of machine output demands investigation. Important decisions increasingly tend to be made in response to computer output. The ultimately responsible human interpreter of "What the machine says" is, not unlike the correspondent with ELIZA, constantly faced with the need to make credibility judgments. ELIZA shows, if nothing else, how easy it is to create and maintain the illusion of understanding, hence perhaps of judgment

deserving of credibility. A certain danger lurks there.

The idea that the present ELIZA script contains *no* information about the real world is not entirely true. For example, the transformation rules which cause the input

Everybody hates me

to be transformed to

Can you think of anyone in particular

and other such are based on quite specific hypotheses about the world. The whole script constitutes, in a loose way, a model of certain aspects of the world. The act of writing a script is a kind of programming act and has all the advantages of programming, most particularly that it clearly shows where the programmer's understanding and command of his subject leaves off.

A large part of whatever elegance may be credited to ELIZA lies in the fact that ELIZA maintains the illusion of understanding with so little machinery. But there are bounds on the extendability of ELIZA's "understanding" power, which are a function of the ELIZA program itself and not a function of any script it may be given. The crucial test of understanding, as every teacher should know, is not the subject's ability to continue a conversation, but to draw valid conclusions from what he is being told. In order for a computer program to be able to do that, it must at least have the capacity to store *selected* parts of its inputs. ELIZA throws away each of its inputs, except for those few transformed by means of the MEMORY machinery. Of course, the problem is more than one of storage. A great part of it is, in fact, subsumed under the word "selected" used just above. ELIZA in its use so far has had as one of its principal objectives the *concealment* of its lack of understanding. But to encourage its conversational partner to offer inputs from which it can select remedial information, it must *reveal* its misunderstanding. A switch of objectives from the concealment to the revelation of misunderstanding is seen as a precondition to making an ELIZA-like program the basis for an effective natural language man-machine communication system.

One goal for an augmented ELIZA program is thus a system which already has access to a store of information about some aspects of the real world and which, by means of conversational interaction with people, can reveal both what it knows, i.e., behave as an information retrieval system, and where its knowledge ends and needs to be augmented. Hopefully the augmentation of its knowledge will also be a direct consequence of its conversational experience. It is precisely the prospect that such a program will converse with many people and learn something from each of them, which leads to the hope that it will prove an interesting and even useful conversational partner.

One way to state a slightly different intermediate goal is to say that ELIZA should be given the power to slowly build a model of the subject conversing with it. If the subject mentions that he is not married, for example, and later speaks of his wife, then ELIZA should be able to make the tentative inference that he is either a widower or divorced. Of course, he could simply be confused. In the long run, ELIZA should be able to build up a *belief structure* (to use Abelson's phrase) of the subject and on that basis detect the subject's rationalizations, contradictions, etc. Conversations with such an ELIZA would often turn into arguments. Important steps in the realization of these goals have already been taken. Most notable among these is Abelson's and Carroll's work on simulation of belief structures [1].

The script that has formed the basis for most of this discussion happens to be one with an overwhelmingly psychological orientation. The reason for this has already been discussed. There is a danger, however, that the example will run away with what it is supposed to illustrate. It is useful to remember that the ELIZA program itself is merely a translating processor in the technical programming sense. Gorn [2] in a paper on language systems says:

> Given a language which already possesses semantic content, then a translating processor, even if it operates only syntactically, generates corresponding expressions of another language to which we can attribute as "meanings" (possibly multiple—the translator may not be one to one) the "semantic intents" of the generating source expressions; whether we find the result consistent or useful or both is, of course, another problem. It is quite possible that by this method the same syntactic object language can be usefully assigned multiple meanings for each expression . . .

It is striking to note how well his words fit ELIZA. The "given language" is English as is the "other language", expressions of which are generated. In principle, the given language could as well be the kind of English in which "word problems" in algebra are given to high school students and the other language, a machine code allowing a particular computer to "solve" the stated problems. (See Bobrow's program STUDENT [3].)

The intent of the above remarks is to further rob ELIZA of the aura of magic to which its application to psychological subject matter has to some extent contributed. Seen in the coldest possible light, ELIZA is a translating processor in Gorn's sense; however, it is one which has been especially constructed to work well with natural language text.

REFERENCES

1. Abelson, R. P., and Carroll, J. D. Computer simulation of individual belief systems. *Amer. Behav. Sci. 9* (May 1965), 24–30.
2. Gorn, S. Semiotic relationships in ambiguously stratified language systems. Paper presented at Int. Colloq. Algebraic Linguistics and Automatic Theory, Hebrew U. of Jerusalem, Aug. 1964.
3. Bobrow, D. G. Natural language input for a computer problem solving system. Doctoral thesis, Math. Dept., MIT. Cambridge, Mass., 1964.
4. Weizenbaum, J. Symmetric list processor. *Comm. ACM 6*, (Sept. 1963), 524–544.
5. Rogers, C. *Client Centered Therapy: Current Practice, Implications and Theory.* Houghton Mifflin, Boston, 1951.
6. Yngve, J. *COMIT Programming Manual.* MIT Press, Cambridge, Mass., 1961.

APPENDIX. An ELIZA Script

```
(HOW DO YOU DO.  PLEASE TELL ME YOUR PROBLEM)
START
(SORRY ((0) (PLEASE DON'T APOLIGIZE)
(APOLOGIES ARE NOT NECESSARY) (WHAT FEELINGS
DO YOU HAVE WHEN YOU APOLOGIZE) (I'VE TOLD YOU
THAT APOLOGIES ARE NOT REQUIRED)))
(DONT = DON'T)
(CANT = CAN'T)
(WONT = WON'T)
(REMEMBER 5
((0 YOU REMEMBER 0) (DO YOU OFTEN THINK OF 4)
(DOES THINKING OF 4 BRING ANYTHING ELSE TO MIND)
(WHAT ELSE DO YOU REMEMBER)
(WHY DO YOU REMEMBER 4 JUST NOW)
(WHAT IN THE PRESENT SITUATION REMINDS YOU OF 4)
(WHAT IS THE CONNECTION BETWEEN ME AND 4))
((0 DO I REMEMBER 0) (DID YOU THINK I WOULD FORGET 5)
(WHY DO YOU THINK I SHOULD RECALL 5 NOW)
(WHAT ABOUT 5) (=WHAT) (YOU MENTIONED 5))
((0) (NEWKEY)))
(IF 3 ((0 IF 0) (DO YOU THINK ITS LIKELY THAT 3) (DO YOU WISH THAT 3)
(WHAT DO YOU THINK ABOUT 3) (REALLY, 2 3)))
(WHAT DO YOU THINK ABOUT 3) (REALLY, 2 3)))
(DREAMT 4 ((0 YOU DREAMT 0)
(REALLY, 4) (HAVE YOU EVER FANTASIED 4 WHILE YOU WERE AWAKE)
(HAVE YOU DREAMT 4 BEFORE) (=DREAM) (NEWKEY)))
(DREAMED = DREAMT 4 (=DREAMT))
(DREAM 3 ((0) (WHAT DOES THAT DREAM SUGGEST TO YOU)
(DO YOU DREAM OFTEN) (WHAT PERSONS APPEAR IN YOUR DREAMS)
(DON'T YOU BELIEVE THAT DREAM HAS SOMETHING TO DO WITH
YOUR PROBLEM) (NEWKEY)))
(DREAMS = DREAM 3 (=DREAM))
(HOW (=WHAT))
(WHEN (=WHAT))
(ALIKE 10 (=DIT))
(SAME 10 (=DIT))
(CERTAINLY (=YES))
(FEEL DLIST(/BELIEF))
(THINK DLIST(/BELIEF))
(BELIEVE DLIST(/BELIEF))
(WISH DLIST(/BELIEF))
(MEMORY MY (0 YOUR 0 = LETS DISCUSS FURTHER WHY YOUR 3)
(0 YOUR 0 = EARLIER YOU SAID YOUR 3)
(0 YOUR 0 = BUT YOUR 3)
(0 YOUR 0 = DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR 3))
(NONE ((0) (I AM NOT SURE I UNDERSTAND YOU FULLY)
(PLEASE GO ON)
(WHAT DOES THAT SUGGEST TO YOU)
(DO YOU FEEL STRONGLY ABOUT DISCUSSING SUCH THINGS)))
(PERHAPS ((0) (YOU DON'T SEEM QUITE CERTAIN)
(WHY THE UNCERTAIN TONE)
(CAN'T YOU BE MORE POSITIVE)
(YOU AREN'T SURE) (DON'T YOU KNOW)))
(MAYBE (=PERHAPS))
(NAME 15 ((0) (I AM NOT INTERESTED IN NAMES)
(I'VE TOLD YOU BEFORE, I DON'T CARE ABOUT NAMES -
PLEASE CONTINUE)) )
PLEASE CONTINUE)) )
(DEUTSCH (=XFREMD))
(FRANCAIS (=XFREMD))
(ITALIANO (=XFREMD))
(ESPANOL (=XFREMD))
(XFREMD ((0) (I AM SORRY, I SPEAK ONLY ENGLISH)))
(HELLO ((0) (HOW DO YOU DO.  PLEASE STATE YOUR PROBLEM)))
(COMPUTER 50 ((0) (DO COMPUTERS WORRY YOU)
(WHY DO YOU MENTION COMPUTERS) (WHAT DO YOU THINK MACHINES
HAVE TO DO WITH YOUR PROBLEM) (DON'T YOU THINK COMPUTERS CAN
HELP PEOPLE) (WHAT ABOUT MACHINES WORRIES YOU) (WHAT
DO YOU THINK ABOUT MACHINES)))
(MACHINE 50 (=COMPUTER))
(MACHINES 50 (=COMPUTER))
(COMPUTERS 50 (=COMPUTER))
(AM = ARE ((0 ARE YOU 0) (DO YOU BELIEVE YOU ARE 4)
(WOULD YOU WANT TO BE 4) (YOU WISH I WOULD TELL YOU YOU ARE 4)
(WHAT WOULD IT MEAN IF YOU WERE 4) (=WHAT))
((0) (WHY DO YOU SAY 'AM') (I DON'T UNDERSTAND THAT)))
(ARE ((0 ARE I 0 )
(WHY ARE YOU INTERESTED IN WHETHER I AM 4 OR NOT)
(WOULD YOU PREFER IF I WEREN'T 4) (PERHAPS I AM 4 IN YOUR
FANTASIES) (DO YOU SOMETIMES THINK I AM 4) (=WHAT))
((0 ARE 0) (DID YOU THINK THEY MIGHT NOT BE 3)
(WOULD YOU LIKE IT IF THEY WERE NOT 3) (WHAT IF THEY WERE NOT 3)
(POSSIBLY THEY ARE 3)) )
(YOUR = MY ((0 MY 0) (WHY ARE YOU CONCERNED OVER MY 3)
(WHAT ABOUT YOUR OWN 3) (ARE YOU WORRIED ABOUT SOMEONE ELSES 3)
(REALLY, MY 3)))
(WAS 2 ((0 WAS YOU 0 )
(WHAT IF YOU WERE 4) (DO YOU THINK YOU WERE 4)
(WERE YOU 4) (WHAT WOULD IT MEAN IF YOU WERE 4)
(WHAT DOES ' 4 ' SUGGEST TO YOU) (=WHAT))
((0 YOU WAS 0)
(WERE YOU REALLY) (WHY DO YOU TELL ME YOU WERE 4 NOW)
(WERE YOU REALLY) (WHY DO YOU TELL ME YOU WERE 4 NOW)
(PERHAPS I ALREADY KNEW YOU WERE 4) )
((0 WAS I 0) (WOULD YOU LIKE TO BELIEVE I WAS 4)
(WHAT SUGGESTS THAT I WAS 4)
(WHAT DO YOU THINK) (PERHAPS I WAS 4)
(WHAT IF I HAD BEEN 4))
((0) (NEWKEY)) )
(WERE = WAS (=WAS))
(ME = YOU)
(YOU'RE = I'M ((0 I'M 0) (PRE (I ARE 3) (=YOU))))
(I'M = YOU'RE ((0 YOU'RE 0) (PRE (YOU ARE 3) (=I))))
(MYSELF = YOURSELF)
(YOURSELF = MYSELF)
(MOTHER DLIST(/NOUN FAMILY))
(MOM = MOTHER DLIST(/ FAMILY))
(DAD = FATHER DLIST(/ FAMILY))
(FATHER DLIST(/NOUN FAMILY))
(SISTER DLIST(/FAMILY))
(BROTHER DLIST(/FAMILY))
(WIFE DLIST(/FAMILY))
(CHILDREN DLIST(/FAMILY))
(I = YOU
((0 YOU (* WANT NEED) 0) (WHAT WOULD IT MEAN TO YOU IF YOU GOT 4)
(WHY DO YOU WANT 4) (SUPPOSE YOU GOT 4 SOON) (WHAT
IF YOU NEVER GOT 4) (WHAT WOULD GETTING 4 MEAN TO
YOU) (WHAT DOES WANTING 4 HAVE TO DO WITH THIS DISCUSSION))
((0 YOU ARE 0 (*SAD UNHAPPY DEPRESSED SICK ) 0)
(I AM SORRY TO HEAR YOU ARE 5) (DO YOU THINK COMING HERE
WILL HELP YOU NOT TO BE 5) (I'M SURE ITS NOT PLEASANT TO
BE 5) (CAN YOU EXPLAIN WHAT MADE YOU 5))
((0 YOU ARE 0 (*HAPPY ELATED GLAD BETTER ) 0)
(HOW HAVE I HELPED YOU TO BE 5)
(HAS YOUR TREATMENT MADE YOU 5) (WHAT MAKES YOU 5 JUST
NOW) (CAN YOU EXPLAIN WHY YOU ARE SUDDENLY 5))
((0 YOU WAS 0) (=WAS))
((0 YOU WAS 0) (=WAS))
((0 YOU (/BELIEF) YOU 0) (DO YOU REALLY THINK SO) (BUT YOU ARE
NOT SURE YOU 5) (DO YOU REALLY DOUBT YOU 5))
((0 YOU 0 (/BELIEF) 0 I 0) (=YOU))
((0 YOU ARE 0)
(IS IT BECAUSE YOU ARE 4 THAT YOU CAME TO ME)
(HOW LONG HAVE YOU BEEN 4)
(DO YOU BELIEVE IT NORMAL TO BE 4)
(DO YOU ENJOY BEING 4))
((0 YOU (* CAN'T CANNOT) 0) (HOW DO YOU KNOW YOU CAN'T 4)
(HAVE YOU TRIED)
(PERHAPS YOU COULD 4 NOW)
(DO YOU REALLY WANT TO BE ABLE TO 4))
((0 YOU DON'T 0)(DON'T YOU  REALLY 4)(WHY DON'T YOU 4)
(DO YOU WISH TO BE ABLE TO 4) (DOES THAT TROUBLE YOU))
((0 YOU FEEL 0) (TELL ME MORE ABOUT SUCH FEELINGS)
(DO YOU OFTEN FEEL 4)
(DO YOU ENJOY FEELING 4)
(OF WHAT DOES FEELING 4 REMIND YOU))
((0 YOU 0 I 0) (PERHAPS IN YOUR FANTASY WE 3 EACH OTHER)
(DO YOU WISH TO 3 ME)
(YOU SEEM TO NEED TO 3 ME)
(DO YOU 3 ANYONE ELSE))
```

```
((0) (YOU SAY 1)
(CAN YOU ELABORATE ON THAT) (DO YOU SAY 1 FOR SOME SPECIAL REASON)
(THAT'S QUITE INTERESTING)))
(YOU = I ((0 I REMIND YOU OF 0) (=DIT))
((0 I ARE 0) (WHAT MAKES YOU THINK I AM 4)
(DOES IT PLEASE YOU TO BELIEVE I AM 4)
(DO YOU SOMETIMES WISH YOU WERE 4)
(PERHAPS YOU WOULD LIKE TO BE 4))
((0 I 0 YOU) (WHY DO YOU THINK I 3 YOU)
(YOU LIKE TO THINK I 3 YOU - DON'T YOU)
(WHAT MAKES YOU THINK I 3 YOU)
(REALLY, I 3 YOU) (DO YOU WISH TO BELIEVE I 3 YOU)
(REALLY, I 3 YOU) (DO YOU WISH TO BELIEVE I 3 YOU)
(SUPPOSE I DID 3 YOU - WHAT WOULD THAT MEAN)
(DOES SOMEONE ELSE BELIEVE I 3 YOU))
((0 I 0) (WE WERE DISCUSSING YOU - NOT ME)
(OH, I 3) (YOU'RE NOT REALLY TALKING ABOUT ME - ARE YOU)
(WHAT ARE YOUR FEELINGS NOW)))
(YES ((0) (YOU SEEM QUITE POSITIVE) (YOU ARE SURE)
(I SEE) (I UNDERSTAND)))
(NO ((0) (ARE YOU SAYING 'NO' JUST TO BE NEGATIVE)
(YOU ARE BEING A BIT NEGATIVE) (WHY NOT) (WHY 'NO')))
(MY = YOUR 2 ((0 YOUR 0 (/FAMILY) 0)
(TELL ME MORE ABOUT YOUR FAMILY) ( WHO ELSE IN YOUR FAMILY
5) (YOUR 4) (WHAT ELSE COMES TO MIND WHEN YOU THINK OF
YOUR 4))
((0 YOUR 0) (YOUR 3) (WHY DO YOU SAY YOUR 3) (DOES THAT
SUGGEST ANYTHING ELSE WHICH BELONGS TO YOU) (IS IT
IMPORTANT TO YOU THAT 2 3)))
(CAN ((0 CAN I 0) (YOU BELIEVE I CAN 4 DON'T YOU) (=WHAT)
(YOU WANT ME TO BE ABLE TO 4) (PERHAPS YOU WOULD LIKE TO BE
ABLE TO 4 YOURSELF)) ((0 CAN YOU 0) (WHETHER OR NOT YOU CAN 4
DEPENDS ON YOU MORE THAN ON ME) (DO YOU WANT TO BE ABLE TO 4)
(PERHAPS YOU DON'T WANT TO 4) (=WHAT)))
(WHAT ((0) (WHY DO YOU ASK) (DOES THAT QUESTION INTERFST YOU)
(WHAT IS IT YOU REALLY WANT TO KNOW)
(ARE SUCH QUESTIONS MUCH ON YOUR MIND)
(WHAT ANSWER WOULD PLEASE YOU MOST)
(WHAT DO YOU THINK) (WHAT COMES TO YOUR MIND WHEN YOU ASK THAT)
(HAVE YOU ASKED SUCH QUESTION BEFORE)
(HAVE YOU ASKED ANYONE ELSE)))
(BECAUSE ((0) (IS THAT THE REAL REASON) (DON'T ANY OTHER REASONS
COME TO MIND) (DOES THAT REASON SEEM TO EXPLAIN ANYTHING ELSE)
(WHAT OTHER REASONS MIGHT THERE BE)))
(WHY ((0 WHY DON'T I 0) (DO YOU BELIEVE I DON'T 5) (PERHAPS I
WILL 5 IN GOOD TIME) (SHOULD YOU 5 YOURSELF) (YOU WANT ME TO 5)
(=WHAT))
(=WHAT))
((0 WHY CAN'T YOU 0) (DO YOU THINK YOU SHOULD BE ABLE TO 5)
(DO YOU WANT TO BE ABLE TO 5) (DO YOU BELIEVE THIS WILL HELP YOU
TO 5) (HAVE YOU ANY IDEA WHY YOU CAN'T 5) (=WHAT))
(= WHAT))
(EVERYONE 2 ((0 (* EVERYONE EVERYBODY NOBODY NOONE) 0 )
(REALLY, 2) (SURELY NOT 2) (CAN YOU THINK OF
ANYONE IN PARTICULAR) (WHO, FOR EXAMPLE) (YOU ARE THINKING OF
A VERY SPECIAL PERSON)
(WHO, MAY I ASK) (SOMEONE SPECIAL PERHAPS)
(YOU HAVE A PARTICULAR PERSON IN MIND, DON'T YOU) (WHO DO YOU
THINK YOU'RE TALKING ABOUT)))
(EVERYBODY 2 (= EVERYONE))
(NOBODY 2 (=EVERYONE))
(NOONE 2 (=EVERYONE))
(ALWAYS 1 ((0) (CAN YOU THINK OF A SPECIFIC EXAMPLE) (WHEN)
(WHAT INCIDENT ARE YOU THINKING OF) (REALLY, ALWAYS)))
(LIKE 10 ((0 (*AM IS ARE WAS) 0 LIKE 0) (=DIT))
((0) (NEWKEY)) )
(DIT ((0) (IN WHAT WAY) (WHAT RESEMBLANCE DO YOU SEE)
(WHAT DOES THAT SIMILARITY SUGGEST TO YOU)
(WHAT OTHER CONNECTIONS DO YOU SEE)
(WHAT DO YOU SUPPOSE THAT RESEMBLANCE MEANS)
(WHAT IS THE CONNECTION, DO YOU SUPPOSE)
(COULD THERE REALLY BE SOME CONNECTION)
(HOW)))
()
```

RECEIVED SEPTEMBER, 1965

LETTERS—continued from p. 35

The technique consists of translating the code for the letter "O" to the code for the numeral 0 whenever it is encountered in the input character string. If the string consists only of items such as numbers and names and it is necessary to sort alphabetically on names, the occurrence of an alphabetic character within a name field is used to cause the code for zero to be retranslated to the code for the letter "O" by a rescan of the characters in the name field.

If no sorting is required, the retranslation can be avoided, provided that delimiters such as FORMAT or GO TO are spelled with zero within the recognizer segment of a translator. It is also necessary to redefine identifier as

⟨identifier⟩ ::= ⟨letter⟩ | ⟨identifier⟩ ⟨letter⟩ | ⟨identifier⟩ ⟨digit⟩ | ⟨0⟩ ⟨identifier⟩

where it is understood that the letter "O" is removed from the standard definition of letter as in ALGOL 60. The redefinition permits the inclusion of identifiers such as ODD or OOPS but prevents the use of an identifier consisting only of the repeated mark O.

This technique requires consistency of use and might result in chaos in a warehousing operation in which the letter "O" is used in parts labels with check digits.

L. RICHARD TURNER
NASA Lewis Research Center
Cleveland, Ohio

Comments on a Problem in Concurrent Programming Control

Dear Editor:

I would like to comment on Mr. Dijkstra's solution [Solution of a problem in concurrent programming control. *Comm ACM 8* (Sept. 1965), 569] to a messy problem that is hardly academic. We are using it now on a multiple computer complex.

When there are only two computers, the algorithm may be simplified to the following:

```
Boolean array b(0; 1) integer k, i, j,
comment This is the program for computer i, which may be
  either 0 or 1, computer j ≠ i is the other one, 1 or 0;
C0: b (i) := false;
C1: if k ≠ i then begin
C2: if not b(j) then go to C2;
    else k := i; go to C1 end;
    else critical section;
    b(i) := true;
    remainder of program;
    go to C0;
    end
```

Mr. Dijkstra has come up with a clever solution to a really practical problem.

HARRIS HYMAN
Munitype
New York, New York