ELIZA--A Computer Program For the Study of Natural Language Communication Between Man and Machine

Joseph Weizenbaum
Massachusetts Institute of Technology
Department of Electrical Engineering
Cambridge, Mass..
Communications of the ACM Volume 9, Number 1 (January 1966): 36-35.

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).

Abstract

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 keywords, and
5. the provision of an ending capacity 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 induice 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 that he.

The object of this paper is to cause just such a reevaluation 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 dialog 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 YOUR 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 input is read and inspected for the presence of a *keyword*. When such a word is found, the sentence is transformed according to a *rule* associated

with the keyword, if not a content-free remark or, undercertain 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 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 has 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 rules constitute the SCRIPT for a particular class of conversation. [The word "transformation" is used in its generic sense rather than that given it by Harris and Chomsky in linguistic contexts.] 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 a 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)	(2)	(3)	(4)
It seems that	you	hate	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. 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 "and 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 love and hate me," the second failing because there is more than one word between "you" and "me".

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 limit 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) ony 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 the sentence into

1. all the words in front of "YOU",

2. the word "YOU", and
3. all the 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:

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

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 programming problems are minimized.

An ELIZA script consists mainly of a set of list structures of the type shown. The actual keyword directory is constructed when such a script is read into the hitherto empty program. The basic structural component of the keyword directory 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 *i*th 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 is placed on *that* list.

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 furher 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.

References

1.
2.
3. Weizenbaum, J. Symmetric list processor. Comm. ACM 6, (Sept. 1963), 524-544.
4.
5. Yngve, J. COMIT Programming Manual. MIT Press, Cambridge, Mass., 1961.
6.

APPENDIX. An Eliza Script