Heuristic Methods for Computer Understanding of Natural Language in Context-Restricted On-Line Dialogues*

KENNETH MARK COLBY AND HORACE ENEA
Department of Computer Science, Stanford University, Stanford, California

Communicated by Kenneth Mark Colby

ABSTRACT

This computer program accepts expressions in natural language as on-line input. It searches each expression for syntactic and semantic patterns. When a pattern match is discovered an appropriate reply is typed out in natural language so that a continuing dialogue develops between person and program. The dialogue is restricted to the context of interpersonal relations such as occurs in a psychiatric interview. The program is an interpreter/supervisor written in SUBALGOL and runs on a 32K IBM 7090 connected via a direct-data device to a PDP-1 and a Philco console.

INTRODUCTION

To clarify the setting in which these heuristic methods apply, we start with an explanation of the term "context-restricted." The context chosen is that of a psychiatric interview being conducted by means of teletype instead of in the usual manner of face-to-face spoken communication. A person types in words, phrases, or sentences in natural language, using his own spellings and punctuation. These inputs express aspects of the individual's thoughts and feelings that he might want to discuss with a psychiatrist or psychotherapist since they bear on some problem with which the person is concerned. When he wants

* This research is supported by Grant MH 06645 from the National Institutes of Health.

a reply, he types a slash and the computer types back its response in natural-language sentences.

Besides being interesting and familiar, this psychiatric context is appropriate for our purposes for a number of methodological and practical reasons. The practical aspects relate to possibilities that computers may eventually alleviate a critical manpower shortage in psychiatry, but these will not be discussed here. The methodological reasons for selecting a psychiatric context pertain to simplifications and constraints characteristic of naturally occurring clinical dialogues. These constraints help to make the situation manageable for current space-limited computers. In psychiatric and psychotherapeutic dialogues a large amount of a patient's output is observed without comment and topics are often changed at the will of the interviewer, who can ask seemingly stupid or naive questions since he is striving to get a picture of the way a particular patient conceives of and finds meaning in his world of experience.

Pragmatic goals of collecting personal information and utilizing it to change a patient's mind determine many of the syntactic and semantic aspects of psychiatric dialogue. Interrogatives are the most common forms used by an interviewer and most of the information processing takes place in the semantic areas of intra- and interpersonal relations.

The contextual use of a dialogue determines its content and form. Other behavioral science contexts have their own problems in gathering information by interview. Psychologists, anthropologists, and political scientists are interested in people's belief systems and how to represent them in computers. Research in belief systems requires specific and comprehensive data. These data can be obtained from questionnaires or documents, but the best method consists of a continuous dialogue, which is flexible and allows rechecking of information during its course. Computer on-line dialogues in natural language will permit us to obtain a more accurate and complete representation of a person's belief system since the person concerned can provide immediate feedback, correction, and corroboration.

By "understanding natural language" we mean, first, that a person's input expressions in everyday English are analyzed and processed in various ways, and second, that the computer outputs expressions in everyday English which indicate that some sort of "understanding" is going on. In an ordinary conversation person A is said to understand person B when A replies appropriately in B's mind to what B has said.

Also, B can ask A questions and if A replies with the right information in accordance with the intent of B's questions, B will believe that A understands him. This is particularly so when B can ask what A has learned from his conversation with B, and A replies with paraphrasings that B agrees capture the meaning of what he said. These tests of understanding are admittedly nonrigorous, but they suffice in both ordinary and clinical dialogues.

Finally, by the term "heuristic methods" we mean that the program contains a collection of procedures arrived at through plausible reasoning and trial and error experience over time. These procedures represent our provisional suggestions. They are nonrigorous and not final. The methods are empirical in that they work without our understanding much about the underlying theoretical principles involved. The program works in the sense that a psychiatric dialogue is conducted, but it does not always work well according to our criteria for a good simulation of the language behavior of a clinician. A description of some of our experiences with three phases of the program follows.

FIRST PHASE

Our first efforts at writing a program capable of conducting on-line clinical dialogues have been described elsewhere [8]. Briefly, this program

PERSON – I DON'T LIKE TO BE COLD.
PROGRAM – WHAT DOES YOUR DISLIKE REMIND YOU OF?
PERSON – TO BE COLD IS TO FEEL REJECTED.
PROGRAM – I SEE.
PERSON – WHAT DO YOU MEAN YOU SEE?
PROGRAM – I MEAN I AM TRYING TO UNDERSTAND WHAT YOU SAY.
PERSON – I DON'T FEEL THAT YOU UNDERSTAND ME VERY WELL.
PROGRAM – WHAT DO YOU FEEL?
PERSON – REJECTED.
PROGRAM – PLEASE GO ON.
PERSON – YOU DON'T SEEM TO UNDERSTAND WHAT I AM TRYING TO TELL YOU.
PROGRAM – WHAT DOES THAT REMIND YOU OF?
PERSON – MY MOTHER.
PROGRAM – TELL ME MORE ABOUT HER.
PERSON – SHE NEVER LISTENED TO ME. SHE WAS WRAPPED UP IN HERSELF.

FIG. 1

worked by scanning input expressions in search of particular patterns of words. When a pattern was found through a straightforward matching procedure, a reply was selected from a group of replies associated with this pattern and typed out as a response to the person's input.

As can be seen in Fig. 1, a dialogue developed between person and program that was at times a good imitation, on both sides, of a clinical interview. We tried it out in an informal way on thirty to forty people, students and friends, who volunteered to participate. The program's failures were more instructive than its successes.

Initially there were two major defects in the program's behavior. Although it maintained continuity fairly well for several sentences, it would then obviously lose the thread of what was being said and give a very inappropriate response.

PERSON – HELLO.
PROGRAM – HOW ARE YOU.
PERSON – FINE.
PROGRAM – WHAT ANIMALS HAVE YOU SEEN AT THE ZOO?
PERSON – I HAVE SEEN SO MANY I CAN'T ANSWER THAT.
PROGRAM – YOU HAVE?
PERSON – YES.
PROGRAM – CAN YOU GIVE ME AN EXAMPLE?
PERSON – NO.
PROGRAM – WHY NOT?
PERSON – GO JUMP IN THE LAKE.
PROGRAM – TELL ME ABOUT A BOOK YOU READ.

FIG. 2

As can be seen in Fig. 2, the dialogue, in this case with a seven-year-old boy, seems reasonable until the person's last statement about go jump in the lake. Instead of ignoring the semantic nature of the input and abruptly changing the topic, the program should somehow recognize the aggressive tone of the remarks, guess at the person's probable feelings at the moment, and reply accordingly.

Such a defect results from the program's failure to grasp the semantic information in the input. Using key words alone is thus not sufficient to provide good continuity. When persons converse, they are able to respond to the connotative as well as to the literal meaning of words. By "meaning" here we mean the network of reference fields to which the word-names have pointers in data structures.

Figure 3 illustrates part of a dialogue with a college student.

PERSON – I MIGHT NEVER BE ABLE TO GIVE MY FAMILY THE SECURITY THEY NEEDED OR THAT I WOULD LIKE TO GIVE THEM.
PROGRAM – REALLY NEVER?
PERSON – I DON'T UNDERSTAND YOUR LAST COMMENT.
PROGRAM – WHAT PUZZLES YOU ABOUT IT?
PERSON – THE WORDS REALLY AND NEVER DO NOT MAKE SENSE IN THIS CONTEXT.
PROGRAM – COULD YOU CLARIFY WHAT YOU MEAN BY REALLY?

FIG. 3

This input sentence is rich in concepts relating to ability, giving, concern about family, security needs, and wishing. But the program selected the key word "never" and made a psychologically trivial response, which indicates that it missed all the important points. Notice how the person recognizes this, asks for some clarification, and receives a reply that would certainly discourage anyone from the hope that further communication would be helpful or even fun.

The second major defect in this phase of the program consisted of an inability to develop an internal model of the person with whom it was communicating. This defect also bears on the continuity problem, but is more important in relation to the problem of machine understanding. When people converse in a manner that indicates mutual understanding, their selection of what to say next and how to say it depends not only on the context and what has just been said, but also on a great deal of background information organized as a knowledge structure or model of the world. A psychiatrist participating in clinical dialogue not only constructs a model of his patient, he has models of

PERSON – WHEN MY FATHER DIED MY MOTHER WAS SICK FOR A LONG TIME.
PROGRAM – TELL ME MORE ABOUT HIM.
PERSON – MY FATHER WAS A WRITER. HE WAS KIND BUT RATHER CYNICAL.
PROGRAM – WHAT ELSE WAS HE LIKE?
PERSON – HE WAS INTELLIGENT AND KNEW A LOT ABOUT PEOPLE.
PROGRAM – IS HE NOW?
PERSON – HE IS DEAD.

FIG. 4

other patients, a model of human behavior in general, and a model of himself to call upon in deciding how to respond to what a patient says. A program capable of conducting reasonable dialogue must have such knowledge structures, cognitive and affective models, if it is to avoid such a horrible blunder as occurs in Fig. 4. The program did not store the simple fact that this person's father was dead in such a way as to avoid asking a ridiculous question.

This sort of nonunderstanding based on ignorance or failure to utilize what it has been told often makes the program unappealing as something with which to communicate. When persons are repeatedly subjected to such crude misunderstanding they become frustrated, angry, and dispirited about any further collaboration. They begin to attack, become sarcastic, resort to joking, and otherwise demonstrate the negative reactions commonly observed in real-life clinical situations in which communication breaks down.

SECOND PHASE

A number of changes, both deletions and additions, were made in Phase 2 of the program, but the most important improvements derived from what we term "stack operations" and from incorporating some memory in the data.

Stack operations preserve greater continuity by directing the dialogue into areas for which certain replies to the computer's output are anticipated; for example, when a "why" type question is asked, a "because" type of response has a high probability of occurring, whereas when a "yes-no" type question is asked, some sort of affirmation or denial can be expected. In the program this was accomplished by using a push-down list that contained arbitrary symbols of our own choice to which key patterns had to be appended to signal that an expected response from the person has occurred and to activate a suitable reply from the program. If the conditions of stack symbols plus key pattern are not satisfied, the question is repeated or rephrased and the program waits for an anticipated response. If the latter still does not occur, the push-down list is popped up and an exit is made to the free flow of the dialogue in which key pattern alone decides what is to be said next by the program. Figure 5 illustrates an example in which the program attempted to explore whether or not a person believes computers think, why he believes

PROGRAM – CAN COMPUTERS THINK?
PERSON – NO THEY CANNOT.
PROGRAM – WHY DON'T YOU BELIEVE COMPUTERS CAN THINK?
PERSON – THEY ARE MANMADE.
PROGRAM – I DON'T UNDERSTAND YOUR REASON. TELL ME AGAIN WHY YOU DOUBT THAT COMPUTERS CAN THINK.
PERSON – BECAUSE THEY ARE MADE OF ELECTRIC WIRES AND WIRES CAN'T THINK.
PROGRAM – IS THAT A GOOD REASON FOR YOUR BELIEF?
PERSON – INDEED.
PROGRAM – I DIDN'T FOLLOW WHAT YOU SAID. LET ME REPHRASE MY QUESTION. IS THE REASON YOU GAVE A GOOD ONE?
PERSON – YES IT IS.
PROGRAM – I THINK I UNDERSTAND NOW WHY YOU BELIEVE COMPUTERS CANNOT THINK.

FIG. 5

what he does, and whether or not he thinks his stated reason is a good one for holding his belief on the subject.

This stack device may fail when unusual words are used by a person in expressing his affirmations, denials, or reasons. When this occurs, the program requests simple yes, no, or because type responses. If this fails, the program returns to the mainstream or, even better, it stacks some other set of symbols and changes the topic of conversation to some other area for which response patterns are anticipated. Stack

PERSON – I AM BUILDING A ROBOT.
PROGRAM – WHY ARE YOU BUILDING A ROBOT?
PERSON – I WANT TO GET IT TO WORK.
PROGRAM – WHAT ELSE DO YOU WANT?
PERSON – SAY SOMETHING.
PROGRAM – DID YOU EVER PRETEND YOU WERE A COWBOY?
PERSON – YES WHEN I WAS LITTLE.
PROGRAM – WHEN WAS THAT?
PERSON – 4 YEARS AGO.
PROGRAM – DO YOU HAVE A LOOSE TOOTH?
PERSON – NO BUT I LOST 3 IN THE LAST MONTH.
PROGRAM – TYPE SOME MORE.
PERSON – SOME MORE.
PROGRAM – IF YOU HAD THREE WISHES WHAT WOULD THEY BE.
PERSON – 1. ALL SUPER POWERS
2. ENTURNAL LIFE
3. WHEN I THINK OF A POWER I GET THE POWER.

FIG. 6

operations are now an integral part of the current program. Our experience with this simple method has indicated it is a powerful way of maintaining continuity and avoiding many of the unsettling disruptions that can occur in man-machine dialogue.

To provide a memory that could be used as the basis for an elementary cognitive model of a person, we at first simply added information by hand from one week to the next. For these informal experiments we enlisted the help of a nine-year-old boy (we mistakenly thought this might simplify things), who participated in a dialogue with the program each week for about three months. After each session we would add certain pieces of information to the data base of the program. For example, Fig. 6 shows part of a conversation with this intelligent and witty boy.

From this we, not the program, learn of his interest in robots, that he has lost teeth recently, and something about his wishes. We added this information to the data base so that in subsequent sessions it was utilized as shown in Figs. 7 and 8.

PROGRAM – TELL ME SOME MORE ABOUT THE ROBOT YOU SAID YOU WERE BUILDING, JOHNNY.
PERSON – HE IS GOING TO BE ABOUT 4 FEET TALL.
PROGRAM – TELL ME MORE ABOUT HIM.
PERSON – HE WILL WALK,
HE WILL TALK,
HE WILL MOVE,
BUT HE WON'T DISOBEY ORDERS.

FIG. 7

PROGRAM – WHAT DO YOUR FRIENDS THINK OF YOUR ROBOT?
PERSON – NOTHING. I HAVEN'T TOLD THEM THAT I AM BUILDING ONE AND THEY WOULD NOT BELIEVE ME IF I WAS GOING TO TELL.
PROGRAM – DO YOU THINK THERE ARE ROBOTS ON MARS, JOHNNY?
PERSON – NO BUT YOU KNOW THERE HAS TO BE SOMEONE THERE TO MAKE THEM.
PROGRAM – ARE YOU CERTAIN?
PERSON – WELL ROBOTS DON'T GROW THEY HAVE TO BE MADE.
PROGRAM – CAN ROBOTS REALLY THINK?
PERSON – NOT YET. IN ABOUT THE 35 CENTUREY.

FIG. 8

One interesting consequence of adding the stack and some memory to the program is that no-key patterns are rarely elicited. This implies that if a dialogue can be directed along paths known to be of interest to a person, a program can recognize most of the patterns in the input and can keep the flow of communication going without the abrupt changes of topics that no-key sentences introduce.

The method of adding information by hand is obviously unsatisfactory to anyone who wants the machine to do this work. Hence we turned to methods for picking up and storing information in the form of beliefs; that is, propositions stated by or contained in input sentences. We wanted the program, given some starting knowledge, to build up a model of a person's belief system. To achieve this, it must be possible to parse or analyze sentences in order to find their semantic kernels. The current phase of the program thus involves a theory of syntax and semantics suitable for our purposes in this context.

CURRENT PHASE

The problem can be divided into four major functions or goals. We must be capable of accepting typewritten input in natural English, including nongrammatical sentences and misspelled or mistyped words. Further, we must be able to dissect sentences into conceptual patterns that reflect the relationships that exist between concepts in the subject's sentences. After these patterns are discovered, we must be capable of making appropriate responses and, finally, we want to obtain and store information about beliefs the subject holds.

Physical description of facilities

On-line input facilities at Stanford are currently limited to teletypes and display scopes connected to the THOR time-sharing system on a Digital Equipment Corporation PDP-1. This system allows up to twenty users access to the PDP-1. Because of the large data base desired, the 4K memory allowed each user under THOR is not sufficient for our purposes; therefore, use is made of a 32K IBM 7090 connected via a direct-data device to the PDP-1. Both the 7090 and PDP-1 have access to an IBM 1301 disk file, which permits use of an extremely large data base during on-line interaction. The PDP-1 and its associated teletypes have, for simplicity and flexibility, been made to appear to the 7090

program as the normal input and output tape units (Tapes A2 and A3).* The interpreter/supervisor for the language to be described is written in SUBALGOL.

The language

A complete syntactical description in BNF of the language we have developed to satisfy, in part, our goals is contained in the Appendix to this article. In the following description, various parts of the complete description will be reproduced as needed. The language has three main sections: the transformations, property lists, and key routines. Each section will be taken up in order.

Transformations

Syntax

⟨transform⟩ ::= ⟨header⟩ ⟨tlist⟩*
⟨header⟩ ::= TRANSFORMS ⟨name⟩ ⟨number⟩
⟨tlist⟩ ::= ⟨astring⟩ = ⟨astring⟩ $
::= ⟨astring⟩ = ⟨astring⟩ $ ⟨tlist⟩
⟨astring⟩ ::= ⟨name⟩ ⟨astring⟩

Example

```
TRANSFORMS FORMAL 2
  DAD = FATHER $
  I'M = I AM $
*
```

FIG. 9

When the string on the left of the equal sign (=) (see Fig. 9) is found, it is replaced by the string on the right. The function of transformations is to expand contractions, correct common misspellings, standardize idiomatic colloquial English, and so forth. Another similar list is used to reverse pronouns and correct agreement between linking verbs and pronouns (so that I AM TIRED might become WHY ARE YOU TIRED?). A third special use is to delete the rest of a sentence when any member of the left side is found. This is useful in compound sentences (e.g., I AM SICK AND I DON'T KNOW WHAT TO DO). These special

* The programs that handle communication between machines and those that handle teletype input/output were written by John Sauter, Stanford Computation Center.

functions of the transforms are controlled by the interpreter/supervisor; in future versions of the system it would be desirable to have more control by the user of the language.

Property lists

Syntax

```
⟨property list⟩ ::= PROPERTIES ⟨plist⟩ *
        ⟨plist⟩ ::= ⟨name⟩ = ⟨prop list⟩ $
                ::= . ⟨name⟩ = ⟨prop list⟩ $
                ::= ⟨name⟩ = ⟨prop list⟩ $ ⟨plist⟩
                ::= . ⟨name⟩ = ⟨prop list⟩ $ ⟨plist⟩
    ⟨prop list⟩ ::= ⟨name⟩, ⟨prop list⟩
```

Example

```
PROPERTIES
  .NOG = NO, OF COURSE NOT, NOPE $
  PNG = I, YOU, HE, SHE, IT, WE, THEY $
*
```

FIG. 10

Words or phrases on the right of the equal sign are members of the group named on the left. The period (.) before the name of the property group indicates that it is a key name and that any of the words or phrases on the right will be treated as that key name when selecting the pivotal key. (See description of key routines following.)

Key routines

Syntax

```
⟨key routines⟩ ::= KEYS ⟨pages⟩ *
       ⟨pages⟩ ::= -⟨key name⟩ ⟨priority⟩ ⟨code⟩
               ::= ⟨key name⟩ ⟨priority⟩ ⟨code⟩ ⟨pages⟩
    ⟨priority⟩ ::= ⟨number⟩
```

Example

```
KEYS
  – TEST 1
    SAY (*TESTING 1 2 3 . *) $
  – NOG 2
    SAY (*WHY NOT? *) $
```

FIG. 11

Syntax

⟨code⟩ ::= ⟨statement⟩
::= ⟨statement⟩ $ ⟨code⟩
⟨statement⟩ ::= ⟨label⟩ ⟨statement⟩
::= ⟨decomposition⟩
::= ⟨recomposition⟩
::= ⟨reply to subject⟩
::= ⟨belief⟩
::= ⟨clear⟩
::= ⟨test⟩
::= ⟨dump⟩
::= ⟨go to⟩
::= ⟨transfer⟩
::= ⟨rekey⟩
::= ⟨restore⟩
::= ⟨move⟩
::= ⟨slide 1⟩
::= ⟨subroutines⟩
::= ⟨return⟩
::= ⟨set⟩
::= ⟨unset⟩
::= ⟨flag⟩
::= ⟨comment⟩
⟨label⟩ ::= ⟨local name⟩
⟨local name⟩ ::= ⟨name⟩

FIG. 12

On input the interpreter scans through each input sentence, checking each word against its dictionary of key words and phrases. This dictionary is composed of those words that appear in the ⟨key name⟩ position and those words or phrases that were listed in the property lists as being equivalent to the ⟨key name⟩ (that is, a period preceded the name of the property group). Expansion of contractions and idioms and similar operations also take place now.

The word or phrase with the highest ⟨priority⟩ is selected as the pivotal key of this communication, and the page of code corresponding to that ⟨key name⟩ is retrieved. Control now passes to the program statements of that page. This normal method of selecting a key is circumvented under three conditions: (1) The key that would normally

be selected is temporarily on a special ignore list; (2) A key of normally low priority is temporarily on the high priority list; or (3) There are key names in the stack (the stack is described later). The pages are kept on a 1301 disk file, which allows quite a large number of key names and associated code to be stored.

Statements

The syntax of Fig. 12 shows the various statements allowed. Each type will be explained in order. Note that any statement may be labeled (this is a local label and can be reached only from within the page).

General information

SHELF ALLOCATION

Shelf	*Use*
0	Selected input sentence
1 – 20	Used during decomposition
21 – 30	Contains first ten input sentences
31 – 40	Temporary storage
41 – 49	Long-term storage
50	Undefined, use as a work area by decomposition
51 – 100	Beliefs

FIG. 13

There are 101 shelves available for storage in the system; each shelf may contain up to 50 words. The shelves are allocated as shown in Fig. 13.

After the sentence containing the pivotal key is known, a copy of it is placed on shelf 0. The first ten input sentences are retained in order of input on shelves 21–30. These shelves are protected during subroutines (see subsequent description of the role of shelves). The function of the other shelves will be mentioned as needed. Except as indicated, control moves from statement to statement within a page.

Decomposition

Syntax

⟨decomposition⟩ ::= DEC (⟨dec list⟩)
⟨dec list⟩ ::= ⟨dec part⟩
::= ⟨dec part⟩ ⟨dec list⟩

⟨dec part⟩ ::= ⟨number⟩
::= ⟨name⟩
::= / ⟨name⟩

Examples
(a) DEC (0 1 /LINKINGVERG 0)
(b) DEC (0 I 1 YOU 0)

FIG. 14

The purpose of the decomposition statement, or DEC, is to match the input sentence to a known pattern. The number 0 will match anything. A literal BCD string will match only itself; a nonzero number will match that number of words from the input string, no more or less. A word preceded by a slash will match any input word or phrase in that word's property group. Thus in Example (a), Fig. 14, /LINKINGVERB would match any linking verb that had been listed with /LINKINGVERB. For example:

.LINKINGVERB = AM, FEEL, IS $

Sentences that will match Example (a) given the foregoing LINKING-VERB list include:

I AM TIRED.
SOMETIMES I FEEL LIKE DYING.

Sentences that will match Example (b) are:

I HATE YOU.
NO MATTER WHAT YOU SAY, I FIND YOU ATTRACTIVE.

Matching proceeds from left to right. If a match is found, the matching parts are distributed to corresponding shelves and control passes to the next statement. If the DEC doesn't match, the next state-

Shelf	*Contents*
0	Undefined
1	Empty
2	I
3	FEEL
4	TIRED
5–20	Empty
20–100	What they were before, except 50 is undefined

ment is skipped. For example, given the sentence I FEEL TIRED, and the DEC in Fig. 14, Example (a), after the match the shelves would be as shown in the table at the bottom of page 14. Shelves 1–20 are cleared before each DEC is tried.

Recomposition

Syntax

⟨recompose⟩ ::= REC ⟨(number⟩, ⟨rec list⟩)
⟨rec list⟩ ::= ⟨rec part⟩
::= ⟨rec part⟩ ⟨rec list⟩
⟨rec part⟩ ::= ⟨quoted string⟩
::= ⟨number⟩
⟨quoted string⟩ ::= *⟨string⟩*
⟨string⟩ ::= ⟨name⟩
::= ⟨punctuation⟩
::= ⟨name⟩ ⟨string⟩
::= ⟨punctuation⟩ ⟨string⟩
⟨punctuation⟩ ::= (all special characters except * or $)

Example

REC (1, *WHY DO YOU SAY * 2 3 4 * ? *)

FIG. 15

The purpose of recomposition is to recombine the shelves.

SHELVES BEFORE REC

Shelf	*Contents*
0	Undefined
1	WELL
2	I
3	HATE
4	YOU
5–20	Unknown
21–100	Unknown

Note that pronouns and linking verbs have been changed. These changes are specified by the programmer with a transform list. If the shelf number on which recomposition is to occur is greater than 30, the

SHELVES AFTER REC

Shelf	*Contents*
0	Undefined
1	WHY DO YOU SAY YOU HATE ME?
2	I
3	HATE
4	YOU
5–20	Unchanged
21–100	Unchanged, except 50 is undefined

transform list is not applied. This convention, which allows for the necessary reversal of pronouns before a reply is made, also allows data to be stored correctly.

Replying to subject

Syntax

⟨reply to subject⟩ ::= SAY (⟨rec part⟩)
⟨rec part⟩ ::= ⟨number⟩
::= ⟨quoted string⟩

Examples

SAY (1)
SAY (*TELL ME ABOUT YOUR FATHER.*)

FIG. 16

SAY allows either a specified shelf or literal string to be typed on the teletype. After the SAY statement is executed, control passes back to the supervisor until a new sentence is typed by the subject.

Belief, clear, test, dump. BELIEF moves the string on the indicated shelf to the next empty shelf above 50.

Syntax

⟨belief⟩ ::= BELIEF (⟨number⟩)
⟨clear⟩ ::= CLEAR (⟨number⟩)
⟨test⟩ ::= TEST (⟨number⟩)
⟨dump⟩ ::= DUMP

Examples

BELIEF (31)
CLEAR (3)
TEST (5)
DUMP

FIG. 17

CLEAR will empty the indicated shelf of all words.

TEST will skip the next statement if the indicated shelf contains one or more words.

DUMP will type out the beliefs collected thus far; that is, the non-empty shelves above 50.

Go, transfer, rekey, restore. A simple GO always transfers to the given label. The labels are local to this page. A "reader" type GO circulates among each of the labels branching to the next each time it is executed. This permits variety in output, even though the input had the same pattern as some previously encountered.

Syntax

⟨go to⟩ ::= ⟨simple go to⟩
::= ⟨reader go to⟩
⟨simple go to⟩ ::= GO ⟨label⟩
⟨reader go to⟩ ::= GO ⟨label list⟩
⟨label list⟩ ::= ⟨label⟩, ⟨label⟩
::= ⟨label⟩, ⟨label list⟩
⟨transfer⟩ ::= TRANSFER ⟨key name⟩
⟨rekey⟩ ::= REKEY
⟨restore⟩ ::= RESTORE

Examples

GO L1
GO START, L3, FIN
REKEY
RESTORE

FIG. 18

TRANSFER fetches a new page from the disk. The page is, of course, referenced by its key name.

REKEY is normally placed at the bottom of a series of DEC's. It causes the pivotal key to be recomputed with the current key removed.

RESTORE is like REKEY except that the contents of shelf 0 replace the original sentence.

Stack, pop. STACK and POP are used to manipulate the stack mentioned in the description of key routines. When computing the pivotal key, if one or more key names are in the stack, the top key name automatically becomes the pivotal key and its corresponding page of code is fetched from the disk. POP removes the top item from the stack under programmer control. The supervisor will automatically pop an item after it has been used as a pivotal key.

Syntax

⟨stack⟩ ::= STACK (⟨key name⟩)
⟨pop⟩ ::= POP

Examples

STACK (PARENT)
POP

FIG. 19

Move, slide 1. MOVE copies the first shelf indicated onto the second. The first shelf is not destroyed.

Syntax

⟨move⟩ ::= MOVE (⟨number⟩, ⟨number⟩)
⟨slide 1⟩ ::= SLIDE 1 (⟨number⟩, ⟨number⟩)

Examples

MOVE (3,31)
SLIDE 1 (5, 50)

FIG. 20

SLIDE 1 appends the first word (if any) of the shelf indicated by the first argument to the end of the string on the second shelf indicated. The first shelf has one word less on it after the operation.

Subroutines. The SUBROUTINE feature permits commonly repeated blocks of code to be written only once. Recursive subroutine calls are

permitted. The ⟨key name⟩ position indicates which subroutine is desired; the ⟨number⟩ position tells which shelf is to be passed as input. Return is via a RETURN statement and the number states the label to which control should pass when backing up one level; for instance, in Example (b) of Fig. 21 if the RETURN (2) were executed, control would continue at label L2 of the page where subroutine was entered. The first 30 shelves are saved and restored automatically (protected) by the supervisor and, therefore, their contents remain unchanged after a subroutine has been entered.

Syntax

⟨subroutine⟩ ::= SUBR (⟨key name⟩, ⟨number⟩ $ ⟨label list 1⟩)
⟨label list 1⟩ ::= ⟨label⟩
::= ⟨label⟩, ⟨label list 1⟩
⟨return⟩ ::= RETURN (⟨number⟩)

Examples

(a) SUBR (S1, 3 $ L1, L2)
(b) RETURN (2)

FIG. 21

Flags. Seven hundred fifty one-bit flags are available to the programmer (see Fig. 22). SET makes the indicated flag 1; UNSET makes it 0. FLAG will execute the next statement if its flag bit is 1; otherwise, the next statement is skipped.

Syntax

⟨set⟩ ::= SET (⟨number⟩)
⟨unset⟩ ::= UNSET (⟨number⟩)
⟨flag⟩ ::= FLAG (⟨number⟩)

Examples

SET (5)
UNSET (5)
FLAG (500)

FIG. 22

Figure 23 is constructed to illustrate many of the features of the language and is somewhat less detailed than is necessary in practice for

a good dialogue capability. Figure 24 shows a dialogue obtained with this program and Fig. 25 is a list of the beliefs collected.

```
-PARG 30 COMMENT AN EXAMPLE.
    A WORD OR PHRASE WITH THE PROPERTY 'PARENT' IS THE
    PIVOTAL KEY $
LS  DEC (0 MY FATHER /LVG 0) $ GO L1 $
    DEC (0 MY MOTHER /LVG 0) $ GO L2 $
    DEC (0 HE 0) $ GO L3 $
    DEC (0 SHE 0) $ GO L4 $
    REKEY $
L1  COMMENT GET BELIEF ABOUT FATHER $
    REC (31, 2 3 4 5 ) $
    BELIEF (31) $
    COMMENT SET UP TO ASK QUESTION $
    REC (31, *I AM*) $
    REC (32, *LIKE MY FATHER*) $
    STACK (PARG) $
    STACK (GENYESNO) $
    SAY (*ARE YOU LIKE YOUR FATHER?*) $
L2  REC (31, 2 3 4 5 ) $
    BELIEF (31) $
    COMMENT ROTATE BETWEEN L5 AND L6 $
    GO L5, L6 $
L3  COMMENT CHANGE 'HE' TO 'MY FATHER' $
    REC (50, 1 *MY FATHER* 3) $
    COMMENT MUST BE CAREFUL NOT TO REVERSE PRONOUNS $
    MOVE (50, 0) $
    GO LS $
L4  REC (50, 1 *MY MOTHER* 3) $
    MOVE (50, 0) $
    GO LS $
L5  SAY (*TELL ME MORE ABOUT YOUR MOTHER.*) $
L6  COMMENT ASK THIS QUESTION ONLY ONCE $
    FLAG (1) $ GO L5 $
    SET (1) $
    STACK (MOTHERQ1) $
    REC (1, *WHY* 4 2 3 5 * ? * ) $
    SAY (1) $
-GENYESNO 0
    COMMENT GET ANSWER AND THEN TRY FOR MORE INFORMATION $
    DEC (0 /NOG 0 ) $ GO L1 $
    DEC (0 /YESG 0 ) $ GO L2 $
    GO L3, L4 $
L3  REKEY $
    COMMENT DID NOT GET RECOGNIZABLE YES OR NO. $
```

```
L4  FLAG (2) $ GO L3 $
    SET (2) $
    SAY (*TRY AND ANSWER YES OR NO WHENEVER POSSIBLE.*) $
L1  REC (33, 31 *NOT* 32) $
    BELIEF (33) $
    REKEY $
L2  REC (33, 31 32) $
    BELIEF (33) $
    REKEY $
```

FIG. 23

PROGRAM – I WOULD LIKE TO KNOW MORE ABOUT YOUR BACKGROUND, ESPECIALLY ABOUT YOUR FAMILY LIFE.
PERSON – MY FATHER AND MOTHER DID NOT GET ALONG WELL.
PROGRAM – TELL ME SOMETHING ELSE YOU THINK IS IMPORTANT ABOUT YOUR FATHER.
PERSON – HE WAS HARD TO LIVE WITH BECAUSE HE WAS SO DOMINEERING.
PROGRAM – ARE YOU LIKE YOUR FATHER?
PERSON – NO, I AM NOT.
PROGRAM – IN SOME WAY THOUGH YOU MUST BE LIKE YOUR FATHER.
PERSON – HE LIKED TO FISH AND I DO TOO.
PROGRAM – DID YOU ADMIRE HIM WHEN YOU WERE YOUNG?
PERSON – I DON'T THINK SO.
PROGRAM – WHY NOT?
PERSON – BECAUSE HE WAS TOO WEAK A PERSON.
PROGRAM – I SEE. NOW TELL ME SOMETHING ELSE ABOUT HIM.
PERSON – WELL HE WAS VERY OPINIONATED.
PROGRAM – WAS THERE MUCH FRICTION BETWEEN YOU AND HIM WHEN YOU WERE GROWING UP?
PERSON – YES. WE FOUGHT ALL THE TIME.
PROGRAM – PERHAPS YOU WERE AFRAID OF HIM.
PERSON – I CERTAINLY WAS. HE USED TO REALLY BLOWUP.
PROGRAM – PERHAPS YOU ARE STILL FEARFUL OF OLDER MEN.
PERSON – LET'S TALK ABOUT SOMETHING ELSE.
PROGRAM – O.K. GO AHEAD. I WILL TRY TO FOLLOW WHAT YOU HAVE TO SAY ABOUT IT.

FIG. 24

BELIEFS –
MY FATHER AND MOTHER DID NOT GET ALONG WELL.
MY FATHER WAS HARD TO LIVE WITH BECAUSE HE WAS DOMINEERING.
I AM NOT LIKE MY FATHER.

MY FATHER LIKED TO FISH AND I DO TOO.
I HAD NOT ADMIRATION FOR FATHER.
BECAUSE FATHER WAS TOO WEAK A PERSON.
MY FATHER WAS VERY OPINIONATED.
I HAD FIGHTS WITH FATHER.
I HAD FEAR OF FATHER.

FIG. 25

DISCUSSION

There are points of similarity and difference between our program and those of others in this area [2, 10, 11, 13]. Rather than discuss these in relation to more general problems of belief systems and natural language [1, 3—9, 12], we will limit our comment to the main deficiencies of our program and how we plan to remedy them.

If a program starts from scratch with no data-base knowledge except what to say in certain situations, the resulting dialogue in time becomes uninteresting and uninformative. To build up a cognitive model of a person with whom it is communicating, the program should have an extensive model of its own to begin with. That is, it should make its replies based on what it knows about human behavior in general, what it wants to find out, and whether or not it has already been told this, directly or indirectly. The ideal program should begin with a large knowledge structure of dictionary words and beliefs in the form of generalizations and implications. An executive or supervisory program then checks this data base, using various subroutines to decide what should be said next in the dialogue. Specific beliefs particular to the person concerned should be added as the dialogue proceeds.

A second improvement concerns how the input expressions are treated. Instead of decomposing an input sentence and then recomposing it in a reply, the program should translate or break down and recombine an input expression into a more explicit form, after which it should reply to this form instead of to the original expression. Natural-language expressions should first be translated into a more formal internal language that will allow the program to know more about what is being said. Much of what a person expresses in on-line dialogue is not in a sentence form. These fragments and ellipses must be expanded into full sentence form in order that information about belief propositions being referred to may be extracted.

Finally we would like to reemphasize the importance of pragmatics in influencing syntactic and semantic features of a context. Writers of programs capable of conducting dialogues must have a clear idea of the goals of the context. They must know what sort of information they are after and how they are going to use it subsequently. When such information appears in the input, it must be recognized as such and when it does not appear, the program must know how to meet this situation and how to guide the dialogue along a goal-directed path. The pragmatic use of the context remains the overriding consideration in writing these conversational programs.

APPENDIX

⟨complete program⟩ ::= ⟨program⟩ FINISH *
⟨program⟩ ::= ⟨program part⟩ | ⟨program⟩ ⟨program part⟩
⟨program part⟩ ::= ⟨transform⟩ | ⟨property list⟩ | ⟨key routines⟩
⟨transform⟩ ::= ⟨header⟩ ⟨tlist⟩ *
⟨header⟩ ::= TRANSFORMS ⟨name⟩ ⟨number⟩
⟨tlist⟩ ::= ⟨astring⟩ = ⟨astring⟩ $ |
⟨astring⟩ = $ |
⟨astring⟩ = ⟨astring⟩ $ ⟨tlist⟩
⟨astring⟩ ::= ⟨name⟩ | ⟨name⟩ ⟨astring⟩
⟨property list⟩ ::= PROPERTIES ⟨plist⟩ *
⟨plist⟩ ::= ⟨name⟩ = ⟨prop list⟩ $ |
.⟨key name⟩ = ⟨prop list⟩ $ |
⟨name⟩ = ⟨prop list⟩ $ ⟨plist⟩ |
.⟨key name⟩ = ⟨prop list⟩ $ ⟨plist⟩
⟨prop list⟩ ::= ⟨name⟩ | ⟨name⟩, ⟨prop list⟩
⟨key routines⟩ ::= KEYS ⟨pages⟩ *
⟨pages⟩ ::= — ⟨key name⟩ ⟨code⟩ |
— ⟨key name⟩ ⟨code⟩ ⟨pages⟩
⟨key name⟩ ::= ⟨name⟩
⟨code⟩ ::= ⟨statement⟩ $ | ⟨statement⟩ $ ⟨code⟩
⟨statement⟩ ::= ⟨label⟩ ⟨statement⟩ |
⟨decompose⟩ | ⟨recompose⟩ |
⟨say⟩ | ⟨belief⟩ | ⟨stack⟩ |
⟨pop⟩ | ⟨comment⟩ | ⟨go to⟩ |
⟨rekey⟩ | ⟨clear⟩ | ⟨move⟩ |
⟨restore⟩ | ⟨test⟩ | ⟨slide 1⟩ |

⟨print beliefs⟩ | ⟨subroutine⟩ |
⟨return⟩ | ⟨set⟩ | ⟨flag⟩ | ⟨unset⟩ |
⟨transfer⟩

⟨label⟩ ::= ⟨local name⟩
⟨local name⟩ ::= ⟨name⟩
⟨decompose⟩ ::= DEC (⟨dec list⟩)
⟨dec list⟩ ::= ⟨dec part⟩ | ⟨dec part⟩ ⟨dec list⟩
⟨dec part⟩ ::= ⟨name⟩ | ⟨number⟩ | / ⟨name⟩
⟨recompose⟩ ::= REC (⟨number⟩, ⟨rec list⟩)
⟨rec list⟩ ::= ⟨rec part⟩ |⟨rec part⟩ ⟨rec list⟩
⟨rec part⟩ ::= ⟨quoted string⟩ | ⟨number⟩
⟨quoted string⟩ ::= * ⟨string⟩ *
⟨string⟩ ::= ⟨name⟩ | ⟨punctuation⟩ | ⟨name⟩ ⟨string⟩ |
⟨punctuation⟩ ⟨string⟩
⟨punctuation⟩ ::= (all special characters except * or $)
⟨say⟩ ::= SAY (⟨rec part⟩)
⟨belief⟩ ::= BELIEF (⟨number⟩)
⟨stack⟩ ::= STACK (⟨key name⟩)
⟨pop⟩ ::= POP
⟨comment⟩ ::= COMMENT ⟨string⟩
⟨go to⟩ ::= ⟨simple go to⟩ | ⟨reader go to⟩
⟨simple go to⟩ ::= GO ⟨label⟩
⟨reader go to⟩ ::= GO ⟨label list⟩
⟨label list⟩ ::= ⟨label⟩, ⟨label⟩ |
⟨label⟩, ⟨label list⟩
⟨rekey⟩ ::= REKEY
⟨clear⟩ ::= CLEAR (⟨number⟩)
⟨move⟩ ::= MOVE (⟨number⟩, ⟨number⟩)
⟨restore⟩ ::= RESTORE
⟨test⟩ ::= TEST (⟨number⟩)
⟨slide 1⟩ ::= SLIDE 1 (⟨number⟩, ⟨number⟩)
⟨print beliefs⟩ ::= DUMP
⟨subroutine⟩ ::= SUBR (⟨key name⟩, ⟨number⟩ $ ⟨label list 1⟩)
⟨label list 1⟩ ::= ⟨label⟩ | ⟨label⟩, ⟨label list 1⟩
⟨return⟩ ::= RETURN (⟨number⟩)
⟨unset⟩ ::= UNSET (⟨number⟩)
⟨set⟩ ::= SET (⟨number⟩)
⟨flag⟩ ::= FLAG (⟨number⟩)
⟨transfer⟩ ::= TRANSFER ⟨key name⟩

REFERENCES

1 R. P. Abelson and J. D. Carroll, Computer simulation of individual belief systems, *Amer. Behav. Sci.* **8**(1965), 24–30.

2 R. Bellman, M. B. Friend, and L. Kurland, *On the construction of a simulation of the initial psychiatric interview.* Paper presented at annual meeting Amer. Psychiat. Assoc., May, 1964.

3 D. G. Bobrow, Syntactic analysis of English by computer–a survey, *Proc. Fall Joint Computer Conf.* **24**(1963), 365–387.

4 K. M. Colby, Computer simulation of a neurotic process. In *Computer simulation of personality* (S. Tomkins and S. Messick, Eds.), pp. 165–179. Wiley, New York, 1963.

5 K. M. Colby, Experimental treatment of neurotic computer programs, *Arch. Gen. Psychiat.* **10**(1964), 220–227.

6 K. M. Colby and J. P. Gilbert, Programming a computer model of neurosis, *J. Math. Psychol.* **1**(1964), 405–417.

7 K. M. Colby, Computer simulation of neurotic processes. In *Computers in biomedical research* (R. W. Stacey and B. Waxman, Eds.), Vol. I, pp. 491–503. Academic Press, New York, 1965.

8 K. M. Colby, J. Watt, and J. P. Gilbert, A computer method of psychotherapy, *J. Nerv. Ment. Dis.*, **142**(1966), 148–152.

9 K. M. Colby, Computer simulation of change in personal belief systems, *Behav. Sci.*, in press (1967)

10 M. R. Quillian, Semantic memory, Ph. D. thesis, Carnegie Inst. Technol., Pittsburgh, Pennsylvania, 1966.

11 B. Raphael, SIR: A computer program for semantic information retrieval, Ph. D. thesis, M.I.T. (Math. Dept.), Cambridge, Massachusetts, 1964.

12 R. F. Simmons, Answering English questions by computer: a survey, *Commun. ACM* **8**(1965), 53–69.

13 J. Weizenbaum, Eliza–a computer program for the study of natural language communication between man and machine, *Commun. ACM* **9**(1966), 36–45.