D. B. GILLIES, Editor

Experiments with a Deductive Question-Answering Program

JAMES R. SLAGLE
Lawrence Radiation Laboratory, University of California, Livermore, California

As an investigation in artificial intelligence, computer experiments on deductive question-answering were run with a LISP program called DEDUCOM, an acronym for DEDUctive COMmunicator. When given 68 facts, DEDUCOM answered 10 questions answerable from the facts. A fact tells DEDUCOM either some specific information or a method of answering a general kind of question. Some conclusions drawn in the article are: (1) DEDUCOM can answer a wide variety of questions. (2) A human can increase the deductive power of DEDUCOM by telling it more facts. (3) DEDUCOM can write very simple programs (it is hoped that this ability is the forerunner of an ability to self-program, which is a way to learn). (4) DEDUCOM is very slow in answering questions. (5) DEDUCOM's search procedure at present has two bad defects: some questions answerable from the given facts cannot be answered and some other answerable questions can be answered only if the relevant facts are given in the "right" order. (6) At present, DEDUCOM's method of making logical deductions in predicate calculus has two bad defects: some facts have to be changed to logically equivalent ones before being given to DEDUCOM, and some redundant facts have to be given to DEDUCOM.

1. Introduction

Researchers in artificial intelligence try to induce intelligent machine behavior. *Webster's New Collegiate Dictionary* gives these two definitions of intelligence: (a) the power of meeting any situation, especially a novel situation, successfully by proper behavior adjustments; and (b) the ability to apprehend interrelationships of presented facts in such a way as to guide action toward a desired goal.

The author takes the heuristic programming approach to artificial intelligence. A *heuristic program* is a computer program that uses "educated guessing" to discover the solutions to intellectually difficult problems. Heuristic programs have been constructed that solve fairly difficult problems [4]. However, some people claim that these programs fail to satisfy the above dictionary definitions of intelligence because the programs are too narrow in scope and do little or no learning. The author thinks that writing a deductive, question-answering program may be a way to get a program that has breadth and can learn.

The author's work described in the present article is based largely on the ideas and theoretical work of McCarthy [5, 7] on his "Advice Taker" and on the SIR program of Raphael [9]. Of course, they should not be held responsible for the faults of the work reported here. Black [2] has written an Advice Taker program. Cooper [3] has also written a deductive, question-answering program. Simmons [11] has reviewed 15 operating question-answering systems.

The author wishes to thank Robert E. Millstein, whose help has included debugging and partially rewriting the author's original program.

Work performed under the auspices of the US Atomic Energy Commission.

2. Purposes of the Investigation

The author hopes that some day intelligent machines will solve important and intellectually difficult problems. Since an intelligent computer should be able to learn and to answer a wide variety of questions, the author chose to work on a deductive, question-answering program, which he calls DEDUCOM, an acronym for DEDUctive COMmunicator. As seen later in this paper, DEDUCOM has answered six kinds of questions, four of which had been answered previously by the use of four separate programs.

The author wants eventually to get a computer to learn from its experience at least as well as a human does. In particular, the computer should learn how to learn better.

One conceivable way in which the computer might learn is the following: Since the program in the computer is also represented by easily manipulable facts, i.e., data, the program could compile the facts into a replica of itself. By modifying these facts and recompiling itself, the computer learns. As seen later, DEDUCOM has written some very simple programs.

If a program is ever to learn a fact from its experience, it must be able to learn by being told that fact. When relevant facts are told to DEDUCOM, its performance improves. Perhaps a deductive question-answering program someday can be told how to improve itself.

3. General Nature of the Investigation

DEDUCOM is written in the LISP programming language [1, 6] and runs on an IBM 7094 computer. The program occupies 2183 registers. After the author told DEDUCOM many facts, the program answered six kinds of questions answerable from these facts. Due to some of DEDUCOM's limitations described in Section 14, considerable care had to be taken to give the facts in the "right" order and in proper form. This article describes DEDUCOM and its performance when given these facts and questions.

4. Preliminary Description of the Operation of DEDUCOM

As initial input to DEDUCOM, the program is told facts in a simple, English-like language which it understands. From these, it deduces an answer to a question in the following way (a much more detailed description is given in later sections): DEDUCOM reduces the original question to a (hopefully) simpler question, which it then reduces to a still simpler one, and so on, until it has a question it can answer directly.

As an illustration, we give a general description of how DEDUCOM deduced an answer to a particular question. Suppose that DEDUCOM is given (in a suitable notation) the following four facts:

HM1. If there are m x's on a v and if there are n v's on a y, then there are mn x's on a y.
HM2. There are five fingers on a hand.
HM3. There is one hand on an arm.
HM4. There are two arms on a man.

When DEDUCOM is asked "How many fingers on a man?" it answers that there are 10. It gets this answer by first using facts *HM1* and *HM2* to reduce the original question to the question "How many hands on a man?" whose answer DEDUCOM need only multiply by five to obtain the answer to the original question. Similarly, it uses facts *HM1* and *HM3* to reduce the question to "How many arms on a man?" which it can answer from fact *HM4*. It obtains the answer to the original question by multiplication.

5. The Depth-First Search Procedure of the Program

DEDUCOM uses a depth-first search which (despite its limitations) was chosen because it is so simple to program. Various kinds of search procedures are discussed by Newell [8] and Slagle [13]. In the search procedure, illustrated in Figure 1, the questions (represented by the nodes) are tried in numerical order, starting with question 1. The two branches from question 1 mean that there are two ways to try to answer question 1. The second branch will be taken only if the first branch (leading to question 2) fails. Complete attention is therefore turned to question 2. There are three ways of trying to answer question 2. Complete attention is next turned to question 3. Next, complete attention is turned to question 4, from which there is nothing to try. The second branch from question 3 is therefore taken which generates question 5, etc. The

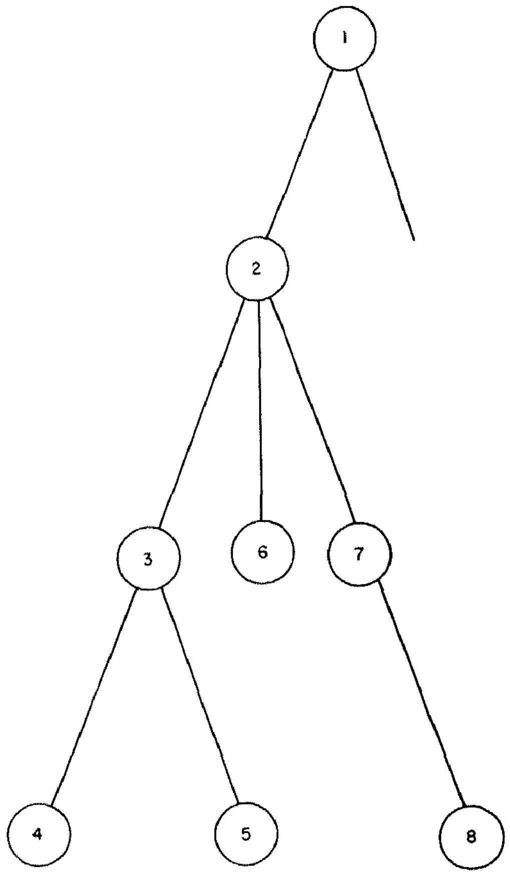

FIG. 1. The depth-first search procedure of the program

process is shown up to the generation of question 8. In the illustration, no question was ever answered. If the top question had been answered, the answer would have been printed out.

6. Some Simple Types of Questions Answered by the Program

To further orient the reader, three simple types of questions are illustrated in this section. Later sections discuss six kinds of questions that are more difficult than the three types discussed in the present section.

Since DEDUCOM is an extension of the ordinary, LISP interpreter, it can perform ordinary computation. For example, when asked the question, times [3, 4], DEDUCOM (or the ordinary, LISP interpreter) gave the answer, 12.

DEDUCOM can answer questions by simple lookup. For example, when DEDUCOM was given the fact:

HM2. There are five fingers on a hand.

and asked: how many[FINGER; HAND], that is, "How many fingers on a hand?", DEDUCOM simply looked up fact *HM2* and answered that there are five.

DEDUCOM can combine ordinary computation and simple lookup. For example, when DEDUCOM was given facts *HM2* and *HM3* and was asked

times[howmany[FINGER; HAND]; howmany[HAND; ARM]],

DEDUCOM gave the answer five.

7. Valueans Questions

Sections 7 and 8 prepare the reader for the flow diagram and detailed example given in Sections 9, 10 and 11. The following sentence defines valueans and ans. If, for some particular value b of the variable v, an answer to the question q is a, then an answer to the question, valueans[v; q], is the ordered pair $(b \cdot a)$, and an answer to the question, ans[v; q] is a. Notice that a value is not specified for the variable v when the question, valueans[v; q], is asked; instead, v takes whatever value is necessary to answer the question q. The following illustrations assume that facts *HM1* through *HM4* are given.

An answer to the question, valueans[v; howmany [FINGER; v]], is (HAND·5), where v has taken on the value HAND. Other answers are (ARM·5) and (MAN·10). An answer to ans[v; howmany[FINGER; v]] is 5. Another answer is 10.

An answer to the question

valueans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]

is (HAND·10). Another answer is (ARM·10). An answer to the question,

ans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]

is 10.

8. The Predicate, Ansupset

The predicate, ansupset, has two arguments, q_1 and q_2, which are questions. By definition, ansupset[q_1; q_2] asserts that any answer to q_2 is an answer to q_1. In other words, the set of answers to q_1 is a superset of the set of answers to q_2 (ansupset is an acronym for answer superset).

For example, fact *HM2* is given to DEDUCOM in the following form:

HM2. Ansupset[howmany[FINGER; HAND]; 5]

Fact *HM2* means that any answer to "5" is an answer to the question, howmany[FINGER; HAND]. In other words, an answer to the question, howmany[FINGER, HAND], is 5.

DEDUCOM is told fact *HM1* in the following form:

HM1. Ansupset[howmany[x; y]; ans[v; times[howmany[x; v]; howmany[v; y]]]]

In other words, an answer to the question,

ans[v; times[howmany[x; v]; howmany[v; y]]],

is an answer to the question, howmany[x; y].

9. Flow Diagram of the Program

The flow diagram, Figure 2, is explained by means of an example in Section 11. DEDUCOM, as represented in this diagram, can answer the three simple types of questions discussed in Section 6. So that DEDUCOM can answer the six more difficult kinds of questions (including the example of Section 11) discussed in later sections, it must be given facts on how to answer valueans questions.

10. Facts on How to Answer Valueans Questions

Certain facts describe how to try to answer questions of the valueans and ans type. It would have been easier to build this information into the program proper, but this

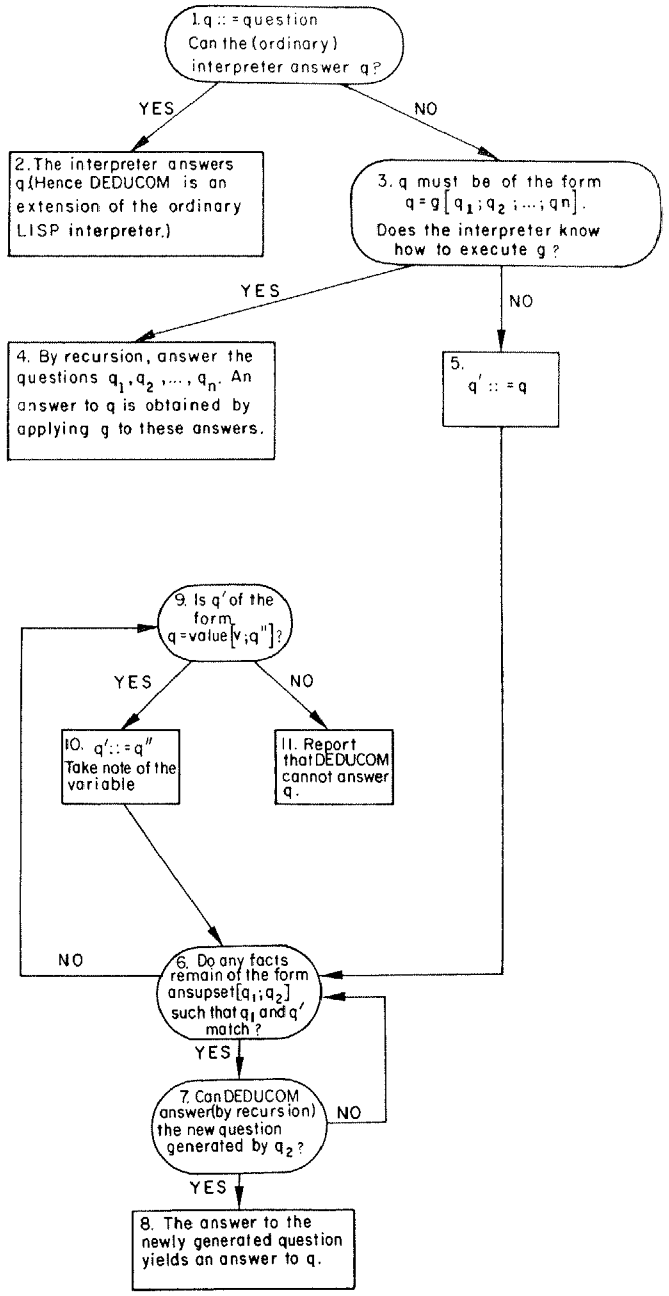
1. q ::= question Can the (ordinary) interpreter answer q?
YES
NO
2. The interpreter answers q. (Hence DEDUCOM is an extension of the ordinary LISP interpreter.)
3. q must be of the form q=g[q1;q2;...;qn]. Does the interpreter know how to execute g?
YES
NO
4. By recursion, answer the questions q1, q2, ..., qn. An answer to q is obtained by applying g to these answers.
5. q' ::= q
9. Is q' of the form q=value[v;q"]?
YES
NO
10. q' ::= q" Take note of the variable
11. Report that DEDUCOM cannot answer q.
NO
6. Do any facts remain of the form ansupset[q1;q2] such that q1 and q' match?
YES
7. Can DEDUCOM answer (by recursion) the new question generated by q2?
NO
YES
8. The answer to the newly generated question yields an answer to q.

FIG. 2. Flowchart of the DEDUCOM program

temptation was resisted because of the long-range goal, as discussed in Section 2, of representing the program as easily manipulable facts.

DEDUCOM was given fact *ANS*:

ANS. Ansupset[ans[v; q]; cdr[valueans[v; q]]]

(The input to the ordinary LISP program, cdr, is an ordered pair; the output is the second member of the pair.) Thus, any answer to the question, cdr[valueans[v; q]], is an answer to ans[v; q]. This agrees with the definition of ans given in Section 7.

DEDUCOM was next given facts *VA1* through *VA5* about valueans. Facts *VA1* through *VA4* aretoo technical to be described in this article.

VA5. If the variable v does not occur in the question q and if an answer to q is a, then an answer to valueans[v; q] is (ANYTHING·a).

11. Detailed Example

The flow diagram, Figure 2, is now explained by means of an example. Suppose that DEDUCOM was given facts *ANS*, *VA1* through *VA5*, and *HM1* through *HM4*, and is asked the question, howmany[FINGER; MAN]. (The numbers below refer to the flow diagram (Figure 2); the letters refer to Figure 3.)

A. 1. Set q to howmany[FINGER; MAN]. Since the interpreter cannot answer q go to 3.
3. The value of g is howmany. Since the interpreter does not know how to execute howmany, go to 5.
5. Set q' to howmany[FINGER; MAN]. Go to 6.
6. Since there is a fact of the form ansupset[q_1 ; q_2], namely,

HM1. Ansupset[howmany[x; y]; ans[v; times[howmany [x; v]; howmany[v; y]]]],

such that q_1 and q' match, go to 7.

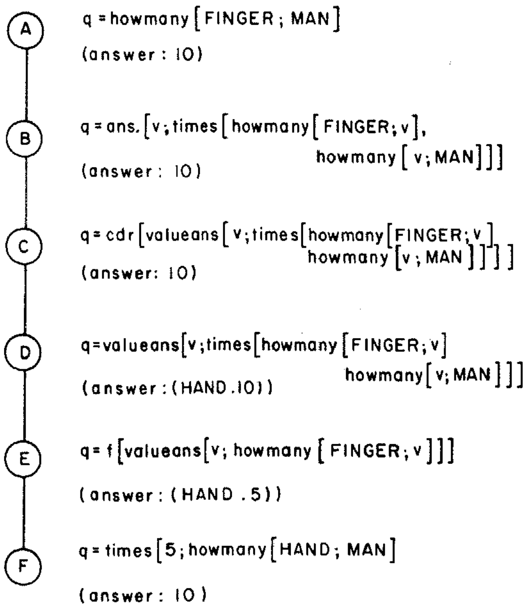
A
q = howmany [FINGER ; MAN]
(answer : 10)
B
q = ans.[v; times [howmany [FINGER; v],
howmany [v; MAN]]]
(answer : 10)
C
q = cdr [valueans [v; times [howmany [FINGER; v]
howmany [v ; MAN]]]]
(answer: 10)
D
q = valueans [v; times [howmany [FINGER; v]
howmany [v; MAN]]]
(answer : (HAND .10))
E
q = f [valueans [v; howmany [FINGER; v]]]
(answer : (HAND .5))
F
q = times [5; howmany [HAND; MAN]
(answer : 10)

FIG. 3. Questions and answers generated while answering the Second Howmany Question.

7. The new question generated from q_2 is:

ans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]

Starting at 1B, we trace how DEDUCOM finds by recursion that an answer to this newly generated question is 10. For the reader's convenience, what happens after DEDUCOM finds this answer, 10, is given in 8, immediately below.

8. Since an answer to the newly generated question (ans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]) is 10, back up to A and get that the answer to the original question is 10.

B. 1. Set q to ans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]. Go to 3.
3. Set g to ans. Go to 5.
5. Set q' to ans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]. Go to 6.
6. Since a match occurs with the fact,

ANS. Ansupset[ans[v; q]; cdr[valueans[v; q]]],

namely, when the value of v is v and the value of q is times[howmany[FINGER; v]; howmany[v; MAN]], go to 7.

7. The newly generated question is

cdr[valueans[v; times[howmany[FINGER; v];

howmany[v; MAN]]]].

Starting at 1C, we trace how DEDUCOM finds by recursion that an answer to this new question is 10. What happens after DEDUCOM finds this answer is given in 8, immediately below.

8. Since an answer to the newly generated question (cdr[valueans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]]) is 10, back up to B and get that an answer to q (ans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]) is 10.

C. 1. Set q to cdr[valueans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]]. Go to 3.
3. Since the interpreter knows how to execute the function cdr, go to 4.
4. Starting at 1D, we trace how DEDUCOM finds by recursion that an answer to the question

valueans[v; times[howmany[FINGER; v];

howmany[v; MAN]]]

is (HAND·10). DEDUCOM backs this answer up to C and finds the answer as follows: Since cdr when applied to (HAND·10) is 10, an answer to q (cdr[valueans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]]) is 10.

D. 1. Set q to valueans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]. Go to 3.
3. Set g to valueans. Go to 5.
5. Set q' to valueans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]. Go to 6.
6. Since *VA5* gives a match, go to 7.
7. Since the attempt to answer the newly generated question fails (not traced in this article), go to 6.
6. Since a match occurs with fact *VA1* (not given in this article), go to 7.
7. The newly generated question is f[valueans[v; howmany[FINGER; v]]]. The nature of the function f is indicated below. Starting at 1E, we trace how DEDUCOM finds by recursion that an answer to this new question is

(HAND·10). What happens after DEDUCOM finds this answer is given in 8, immediately below.

8. Since an answer to the new question (f[valueans[v; howmany[FINGER; v]]]) is (HAND·10), back up to D and obtain that an answer to q (valueans[v; times[howmany[FINGER; v]; howmany[v; MAN]]]) is (HAND·10).

E. 1. Set q to f[valueans[v; howmany[FINGER; v]]]. Go to 3.
3. Since the interpreter knows how to execute f, go to 4.
4. We do not trace in this article how DEDUCOM finds by recursion that an answer to valueans[v; howmany[FINGER; v]] is (HAND·5). DEDUCOM returns to E with this answer and does the following: Applying f to this answer causes the following question to be asked: times[5; howmany[HAND; MAN]]. From this, the reader can get some idea of the nature of the function f.

F. 1. Set q to times[5; howmany[HAND; MAN]]. Go to 3.
3. Since the interpreter knows how to execute times, go to 4.
4. By recursion, DEDUCOM finds that an answer to the question, "5," is 5. By recursion, DEDUCOM finds that an answer to howmany[HAND; MAN] is 2; the trace (not given in this article) of DEDUCOM answering this question is almost the same as the entire trace given above. Since times applied to 5 and 2 is 10, the answer to the original question is 10.

12. Performance of the Program When Given Facts and Questions

As discussed in Section 10, DEDUCOM was first given facts *ANS* and *VA1* through *VA5.*

12.1 *The Howmany Questions.* (This is a modification of a question answered by the SIR program of Raphael [9].) DEDUCOM was given the following facts.

HM1. If there are m x's on a v and if there are n v's on a y, then there are mn x's on a y.
HM2. There are 5 fingers on a hand.
HM3. There is 1 hand on an arm.

The First Howmany Question. DEDUCOM was asked, "How many fingers are there on a man?", which is not answerable from the facts already told to DEDUCOM. DEDUCOM entered an endless loop.

The Second Howmany Question. DEDUCOM was next given the following fact:

HM4. There are 2 arms on a man.

When DEDUCOM was asked the same question as before, it answered in 14 seconds that there are 10 fingers on a man. This is the detailed example traced in Section 11.

The Third Howmany Question. DEDUCOM was next given the following fact:

HM5. There are 2 hands on a man.

When DEDUCOM was asked the same question as before, it answered in only 11 seconds that there are 10 fingers.

The Fourth Howmany Question. DEDUCOM was next asked the question, "What is the product of the number of fingers on a hand and the number of hands on an arm?" The answer given by DEDUCOM was 5.

12.2 *The Set Question.* (This is a modification of question 23 stated by Cooper [3].) DEDUCOM was next given the following 13 facts:

SET 1. If there is a v such that all v's are y's and such that all x's are v's, then all x's are y's.
SET 2. If all x's are y's, then it is false that all x's are not y's.
SET 3. If there is a v such that no v's are y's and such that all x's are v's, then it is false that all x's are y's.
SET 4. An answer to "Is it true that all x's are y's and that all x's are z's?" is an answer to "Are all x's both y's and z's?"
SET 5. All x's are x's.
SET 6. If x does not exist, then all x's are y's.
SET 7. All x's are things.
SET 8. An answer to "Are all y's x's?" is an answer to "Is whatever not x not y?"
SET 9. Gasoline is a fuel.
SET 10. Gasoline exists.
SET 11. Gasoline is combustible.
SET 12. Combustible things exist.
SET 13. Combustible things burn.

When DEDUCOM was asked "Is gasoline a fuel that burns?" it answered that it is.

Of course, the fact that Cooper's program was unable to answer question 23 should not be taken to mean that his program is inferior to DEDUCOM. Cooper's program does an excellent job in translating English sentences into set notation, whereas DEDUCOM cannot translate at all and must be given facts *SET 1* through *SET 13* and question 23 in set notation. Furthermore, out of 23 questions, Cooper's program answered 19. Due to the weakness of DEDUCOM's search procedure, it answered only 7, namely, questions 1, 2, 3, 5, 6, 11 and 23. Question 23 was given to DEDUCOM only to show that it can answer a wide variety of questions.

12.3 *Facts on How to Answer "What Should x Do So That p?"* The author next wanted DEDUCOM to answer questions like:

1. "What should the computer do so that the computer wins the game?" [7]
2. "What should Koko do so that Koko continues living?" [10]
3. "What should the monkey do so that the monkey has the bananas?" [7]

The ability to answer such questions could be given to DEDUCOM more easily by building into the program than by giving more facts. However, in keeping with the philosophy given in Section 2, this ability was given by means of more facts. The expression, howult[x; p], means "What should x do so that (ultimately) p?"

DEDUCOM was next given the facts *HU1* through *HU15*. If DEDUCOM were stronger in predicate calculus, it could make do with fewer than 15 facts. Only three of the facts are simple enough to be included in this article:

HU1. If there is an r such that r leads to p, then any answer to howult[x; r] is an answer to howult[x; p].

HU2. If p is true, then doing nothing is an answer to howult[x; p].
HU3. If there is a p that implies that x can perform the action a, then appending the action a to an answer to howult[x; p] is an answer to howult[x; a].

12.4 *The End Game Question.* (This is a modification of a question stated by McCarthy [7].) DEDUCOM was next given the following facts:

EG1. Move M_1 or move M_2 must be made by the opponent.
EG2. Move M_1 leads to position P_1.
EG3. Move M_2 leads to position P_2.
EG4. In position P_1, the computer can make move N_1.
EG5. In position P_2, the computer can make move N_2.
EG6. Making move N_1 in position P_1 or making move N_2 in position P_2 leads to a win for the computer.

DEDUCOM was asked "What should the computer do so that the computer wins?" DEDUCOM answered that the computer should wait for the opponent to move; then, if the position is P_1, make move N_1, if the position is P_2, make move N_2.

12.5 *The Mikado Question.* (This is a modification of a question stated by Safier [10].) DEDUCOM was next given the following facts:

MIK1. If x is not married, y is not married, x is male, and y is female, then x can do something so that y can marry x.
MIK2. If x can do something so that Katisha can marry x, then x can marry Katisha.
MIK3. Koko is not married.
MIK4. Katisha is not married.
MIK5. Koko is male.
MIK6. Katisha is female.
MIK7. Someone marrying y leads to y being married.
MIK8. Katisha being married leads to Katisha not claiming Nankipoo.
MIK9. Katisha not claiming Nankipoo leads to Katisha not accusing Nankipoo.
MIK10. Katisha not accusing Nankipoo leads to Nankipoo being able to appear and stay alive.
MIK11. If Nankipoo can appear and stay alive, then Koko can produce Nankipoo.
MIK12. Someone producing Nankipoo leads to the Mikado not thinking that Nankipoo is dead.
MIK13. The Mikado not thinking that Nankipoo is dead leads to Koko staying alive.

DEDUCOM was asked "What should Koko do so that Koko can stay alive?" DEDUCOM answered that Koko should marry Katisha.

12.6 *The Monkey Questions.* (This is a modification of a question stated by McCarthy [7].) DEDUCOM was next given the following facts:

MB1. The monkey can move the box to any place.
MB2. Someone moving v to u leads to v being at u.
MB3. The monkey can climb the box.
MB4. v being at u and p climbing v leads to v being at u and p being on v.
MB5. Under the bananas is a place.
MB6. If the box is under the bananas and the monkey is on the box, then the monkey can reach the bananas.
MB7. p reaching x leads to p having x.

The First Monkey Question. DEDUCOM was next asked howult[MONKEY; has[MONKEY; BANANAS]], i.e., "What should the monkey do so that the monkey has the bananas?" DEDUCOM answered:

```
((THE MONKEY SHOULD DO THE FOLLOWING)
(THE MONKEY MOVES THE BOX UNDER THE BANANAS)
(THE MONKEY CLIMBS THE BOX)
(THE MONKEY REACHES THE BANANAS))
```

The Second Monkey Question. DEDUCOM was next asked the more difficult question, value[v; howult[MONKEY; has[v; BANANAS]]]. This question differs from the preceding question in that DEDUCOM must find a value for the variable v so that it can answer the question, howult[MONKEY; has[v; BANANAS]]. DEDUCOM answered:

```
(MONKEY·((THE MONKEY SHOULD DO THE FOLLOWING)
(THE MONKEY MOVES THE BOX UNDER THE BANANAS)
(THE MONKEY CLIMBS THE BOX)
(THE MONKEY REACHES THE BANANAS)))
```

12.7 *The State Description Compiler Question.* (This is a modification of a question answered by the State Description Compiler of Simon [12].) DEDUCOM was next given the following facts:

SDC1. Cell H_5 contains the symbol s_1 followed by the list r_1.
SDC2. Cell H_0 contains the list r_0.
SDC3. The computer can execute $(P1\ s)$.
SDC4. H_0 containing ρ_0, c containing σ_0 followed by ρ_1, and executing $(P1\ \sigma_1)$ leads to H_0 containing σ_1 followed by ρ_0 and c containing σ_0 followed by ρ_1.
SDC5. The computer can execute $(P2\ c)$.
SDC6. The existence of a σ_0 such that c contains σ_0 followed by ρ_1, H_0 containing σ_1 followed by ρ_0, executing $(P2\ c)$ leads to H_0 containing ρ_0 and c containing σ_1 followed by ρ_1.

DEDUCOM was asked "What should the computer do so that H_0 contains r_0 and H_5 contains the symbol J_3 followed by r_1?" DEDUCOM answered

```
((THE COMPUTER SHOULD DO THE FOLLOWING)
(P1 J3)
(P2 H5))
```

12.8 *The Time Taken by the Program to Answer Each Question.* The time taken by DEDUCOM, on the 7094, to answer each question is given in Table I.

TABLE I

Question	*Time*
Second Howmany	14 sec
Third Howmany	11 sec
Fourth Howmany	4 sec
Set	1.6 min
End Game	1.5 min
Mikado	5.7 min
First Monkey	2.9 min
Second Monkey	5.8 min
State Description Compiler	3.4 min

13. Conclusions

When given relevant facts, DEDUCOM can answer a wide variety of questions. The answer to a question can be a program, a number, etc., and need not be only yes or no.

Combining question-answering and ordinary computation allows a relatively small program to answer a wide variety of questions. DEDUCOM is a relatively small program, which is an extension of the ordinary LISP interpreter. A human questioner can mix a great deal of ordinary computation with his question. A simple example is the question, times[howmany[FINGER; HAND]; howmany[HAND; ARM]] in which times is the ordinary LISP multiply function handled by the interpreter.

A human can increase the power of DEDUCOM by simply telling it more facts. After being told a crucial fact, DEDUCOM answers questions which it previously could not answer (e.g., the Second Howmany Question). After being told a relevant fact, DEDUCOM quickly answers questions which it previously answered slowly (e.g., the Third Howmany Question). A fact tells DEDUCOM how to answer a general kind of question or tells some more specific piece of information, although no such distinction is made within the program.

DEDUCOM can write simple programs. For instance, the answer, to the State Description Compiler Question is a set of instructions for a hypothetical computer. Further, the answers to the Mikado, First and Second Monkey, and Endgame Questions state procedures which could conceivably be expanded to programs. It is hoped that this ability is the forerunner of an ability to self-program, which is a way to learn.

DEDUCOM is very slow in answering questions. Since Black [2] has the same complaint about his deductive question-answering program, slowness may be an intrinsic problem for such programs.

14. Future Plans

Some of the planned improvements to DEDUCOM are given below. The author would appreciate other suggestions.

The depth-first search procedure will be replaced by a better search procedure. The weakness of DEDUCOM's search procedure has two bad effects: (1) DEDUCOM cannot answer some questions which are answerable from the given facts. For example, DEDUCOM could not answer 16 of the 23 questions posed by Cooper [3]. (2) Some other questions can be answered only if the relevant facts are given in the "right" order; for example, when DEDUCOM was given fact *HM1* after fact *HM4* and asked "How many fingers are there on a man?" it entered an endless loop.

DEDUCOM will be given an efficient predicate calculus program. The weakness of DEDUCOM in making logical deductions has two bad effects: (1) Before being given to DEDUCOM, some facts have to be changed into logically equivalent ones. (2) Some redundant facts have to be given to DEDUCOM.

DEDUCOM will store and retrieve facts more efficiently than at present. This will become essential when the number of facts gets large. The plan is to use an extension of Stefferud's method [15] of storing and retrieving theorems in the Logic Theory program.

RECEIVED MARCH, 1965

REFERENCES

1. BERKELEY, E., AND BOBROW, D. (Eds.) The programming language LISP: Its operation and applications. Information International, Inc., Cambridge, Mass., 1964.
2. BLACK, F. A deductive question answering system. Thesis, Harvard U., Cambridge, Mass., 1964.
3. COOPER, W. S. Fact retrieval and deductive question answering information retrieval systems. *J. ACM 11* (Apr. 1964), 117–137.
4. FEIGENBAUM, E., AND FELDMAN, J. (Eds.) *Computers and Thought.* McGraw-Hill, New York, 1963.
5. MCCARTHY, J. Programs with common sense. *Symp. Mechanization of Thought Processes.* Nat. Physical Lab., Teddington, Middlesex, England, 1958.
6. MCCARTHY, J., ET AL. *LISP 1.5 Programmer's Manual.* M.I.T. Press, Cambridge, Mass., 1963.
7. MCCARTHY, J. Situations, actions & causal laws. Mem., Stanford Artificial Intelligence Proj., Stanford U., Palo Alto, Calif., July 1963.
8. NEWELL, A. Some problems of basic organization in problem solving programs. In *Self-Organizing Systems*, p. 393, Spartan Books, Washington, D.C., 1962.
9. RAPHAEL, B. A computer program which "understands". Proc. 1964 Fall Joint Comput. Conf., Spartan Books, Washington, D.C., 1964.
10. SAFIER, F. The Mikado as an advice taker problem. Memo, Stanford Artificial Intelligence Proj., Stanford U., Palo Alto, Calif., July 1963.
11. SIMMONS, R. F. Answering English questions by computer: a survey. *Comm. ACM 8* (Jan. 1965), 53–70.
12. SIMON, H. Experiments with a heuristic compiler. *J. ACM 10* (Oct. 1963), 493–506.
13. SLAGLE, J. A heuristic program that solves symbolic integration problems in freshman calculus: symbolic automatic integrator (SAINT). Rep. 5 G-0001, MIT Lincoln Lab., May 10, 1961; available from MIT Microproduction Service, Cambridge, Mass. Ref. No. H-295.
14. ——. The heuristic program that solves symbolic integration problems in freshman calculus. *J. ACM 10* (Oct. 1963), 507–520.
15. STEFFERUD, E. The logic theory machine: a model heuristic program. RAND Corp., Santa Monica, Calif., June 1963.