PROVER

An Elementary Theorem-Proving Program

(In partial fulfillment of requirements
for graduation with honors in mathematics.)

William W. Berghel

May 1983

Acknowledgement

The author wishes to thank his advisor, Dr. Robert L. Wilson, for his continuing support and help in this project. The author also wishes to express his thanks to Dr. Robert S. Johnson for the use of his course materials for Mathematics 301.

INTRODUCTION

Ever since the advent of computers, there have been attempts to write theorem-proving programs for various reasons. One of the first theorem-proving programs was written in 1956 by Allen Newell, J.C. Shaw, and H.A. Simon for the RAND Corporation and Carnegie Institute of Technology. The program, Logic Theorist, was designed to translate propositional calculus using Whitehead and Russell's Principia Mathematica. Its methods were mostly substitution and replacement.(1)

The following year, the trio devised the General Program Solver, which was, to some extent, an expanded form of Logic Theorist, except that it used means-ends analysis in order to reduce the possible number of methods of deriving a solution.(2) The third major program of the fifties was Herbert Gelernter's Geometry Theorem-Proving Machine. Written in 1959 at the IBM Research Center, its goal was to solve high school geometry problems.(3)

(1) Barr and Feigenbaum, The Handbook of Artificial Intelligence, Volume I, pp. 109-110.
(2) Ibid., pp. 112-113.
(3) Ibid., p. 119.

Since those days, much work has progressed in automatic theorem-proving, but little of it has actually surfaced in the form of "program listings." The reason for this secrecy is that theorem-proving is a very competitive area of computer science, for theorem-proving goes far beyond the usual contexts of symbolic logic. A theorem-proving program could be used for many other special chores, including mathematics (assuming that mathematics is separate from symbolic logic). However, a far greater consequence of having a theorem-prover is that it could be used to test program correctness; it could possibly be used to see if a given algorithm worked properly, if a given program had endless loops, or even if a given set of code was the most efficient implementation of an algorithm on a particular computer system. So, one can see the long-range goals of companies to develop theorem-provers; a program written abstractly enough could be applied to nearly every possible situation.

Much progress has been made in theorem-proving techniques in the past twenty years by the previously-mentioned pioneers, as well as by Chang and Lee, Boyer and Moore, and W.W. Bledsoe, to name a few. However, many of their efforts have materialized in the form of "expert systems" (expert systems imitate human actions rather than attempt to exhibit creativity). These programs recognize types of problems and then move to the appropriate method of proof. Thus, to some extent they lack the abstractness of some of the earlier programs; in their favor is the fact that they can handle most of the present types of proof, including proof by contradiction and proof by induction.

At this point, I should mention that, originally, the ultimate goals of these people were to write theorem-provers; today, the goals tend toward writing theorem-proving assistants. The distinction is important. A theorem-prover is self-sufficient; it is given the initial data and the conclusion, to which it adds all of the necessary steps. On the other hand, while a theorem-proving assistant can exhaustively perform insertions, replacements, and implications, it still has the flexibility of the human -- it can be given help in the middle of a problem. As an example, a self-contained program may be able to do induction, but it must decide on what it should induct; an assistant program can be given the induction part separately.

My goal is to write a theorem-proving assistant that can, to some extent, take a hypothesis (that works on IF-THEN rules) and make some progress toward a realizable goal. However, since I need a more concrete problem set, I have restricted my goal to writing a program that might, when given minimal help by the user, be able to pass Mathematics 301, "Fundamental Concepts of Mathematics," with flying colors. So, this program should be able to perform some tasks with symbolic logic, as well as prove that, if $x<y$, then $y>x$.

PROVER -- An Elementary Theorem-Proving Program

PROVER is the title of my theorem-prover, or, more accurately, my theorem-proving assistant. It was developed over the 1982-1983 school year in order to satisfy the requirements of my Honors Thesis. For my programming language, I had two choices: LISP and BASIC. LISP is the favorite among the designers of most theorem-proving programs; however, since I am most fluent in BASIC, I wrote PROVER in the latter language. A beneficial side-effect of this decision is that PROVER will run on a home computer as it is now written, without modifications. The listing, explanation of sections, variable lists, and minimal directions are in the back of this report. For now, let me demonstrate some of the proofs which PROVER can and cannot do.

PROVER is very good at following straightforward IF-THEN rules to their logical conclusion. For instance, if one enters:

```
VAR xyz                  /
IF x=y y=z THEN x=z     /   for an explanation of capitalized
GIVEN a=b              (    words, refer to the documentation
GIVEN b=c               \   in the back of this report
IS a=c                   \
```

PROVER will respond in an affirmative manner. Likewise, if one enters:

```
VAR xyzuv
IFF x>y THEN x=y+u
IFF x<y THEN y=x+u
IF x=y THEN y=x
GIVEN a>b
IS b<a
```

PROVER will again respond YES! to your question. If you would like to see a list of deductions that it has formed, you may enter TELL to see them.

We have now seen some kinds of proofs that PROVER can perform. Are there any others? Aside from the "obvious" kinds, yes and no. Yes, it can also perform two more types of proof -- proof by contradiction and proof by induction. However, at this point, the fact that PROVER is a theorem-proving assistant, and not a theorem-prover, plays a crucial role. In order to prove by contradiction, one must enter the theorem "backwards" -- i.e., one must assume the contrapositive also. In the case of proof by induction, one must enter the "1" case and prove that it is in the set, and then he must enter the "n" case and prove that "n+1" is in the set. So, PROVER is capable of these tasks, but it is not really performing them on its own -- it needs expert help from the user.

PROVER can, as I noted above, detect contradictions. However, when it says that is has a contradiction, it does not necessarily mean that you have a proof by contradiction; it means that the rules and the givens collide to give a deduction contrary to another deduction (I should state here that givens and deductions are treated by PROVER in the same way, so I shall use the terms interchangeably). If your method of proof is proof by contradiction, then you have succeeded somewhere; otherwise, you have probably entered a rule and/or a given incorrectly.

I am amazed at how many methods of proof PROVER seems to be unable to do. PROVER has no replacement command, as in, "Replace all ((x)) with (x)." For this reason, it often gets trapped in terminology. One such example is associativity. One cannot simply say, "Get rid of all the parentheses." One must give a rule to follow. Unfortunately, there are so many special cases for the associativity rules that it is easier to simply manually remove the unnecessary parentheses. While this method is not particularly appealing, no better way has yet been suggested (remember: since this program is meant to also perform various proofs in fields other than mathematics, I cannot assume that the parentheses mean "do this first" as they do in mathematics).

Another problem with PROVER is that it cannot perform commutativity easily. For example, if $x>y$ and $u>v$, then $x=y+q1$ and $u=v+q2$. But then $x+u=y+q1+v+q2$. The program has no way of telling that q1 and q2 should just be moved to the end of the statement to make $x+u=y+v+q1+q2$, thus giving $x+u>y+v$. One may, however, "single step" the program through, using, for example, a convention that "all variables are listed in alphabetical order" and changing the givens as appropriate.

CONCLUSION

I am, to some extent, impressed by what my program can prove. It probably seems quite trivial to the casual observer, but believe me, it is not easy to implement such a process. Still, I am even more surprised by what PROVER cannot do; I never thought I would have such a hard time just because of some parentheses! When writing a program like this, one learns about the special cases and the restrictions that one must force on the user of the program.

While this program may be able to prove little more than if $x<y$ then $y>x$, still, few students in Mathematics 301 proved that theorem. While my program may be overshadowed by the achievements of others over the past thirty years, and deservedly so, one must start somewhere. I set out to do what looked like a simple task, and I ended up, to some extent, showing why others have not done this simple task. I suppose that success in a project like this is partly in understanding the parts that are unsuccessful.

PROVER -- The Documentation

PROVER is a program designed to run in most dialects of BASIC. The commands are as follows:

AUTOMATIC: Turns automatic proving mode on and off. When the mode is on, PROVER will prove one "single step" and then try to prove another. When the mode is off, PROVER will prove one "single step" and then will stop.

COMMAND: Gives the user a list of commands and the formats.

DELETE: Deletes the rules and/or givens. The range will be asked for both cases on the next line.

END: Terminates proving session.

GIVEN: Adds what follows to the "given" list. Note: if the first three letters are "NOT", then PROVER assumes the negation of the rest of the line.

IF: Adds a rule. See GIVEN.

IFF: Adds a rule and its converse. See GIVEN, IF.

IS: Check to see if what follows is a valid deduction.

NEW: Clears workspace. This has the same effect as starting over from scratch.

NUMBER: This gives the number of rules, the number of deductions, and the various modes that are on.

TELL: Gives a list of the rules, deductions, and variables.

VARIABLE: Enters what follows as a variable list.

Note: All commands may be shortened to three (3) characters.

To a large extent, I use the following variables as described below:

Arrays:

```
C$ [] : Letter equivalences for variables
F  [] : False/Trues for Rule (see G[])
G  [] : Truth of Givens (-1=Undefined , 0=T, 1=F)
G$ [] : Givens/Deductions
I  [] : If-parts that could work (0=No, 1=Yes)
I$ [] : If-parts (ANDed together) of Rules
R  [] : Rule Place (+=Variable, -=Simplified assignment)
S$ [] : Simplified Version of Rule
T$ [] : Then-parts of Rules
W  [] : Where we are in IF composition
Z$ [] : Printing Array (easy to print, say, 20 at once)
```

Scalars:

```
C       :Count (number of If-Parts in rule)
D,D0    :Number of Givens/Deductions
D1,D2   :Givens/Deductions to be deleted
E       :Truth of hypothesis
F       :Truth of deduction being tested
P,P0,P1:Position in string
R,R0    :Number of Rules
R1,R2   :Rules to be deleted
S,S0    :Place in S$[]
T       :Total dimension size for arrays
V       :Number of variables (never used)
W0      :Column for W[]
Y       :Are we through (-1=Yes, 0=No)
I,J,I0 :For-Next Variables (I0 is outside I1, etc.)
```

Strings:

```
E$ : End state to be deduced
L$ : Left three letters of a string
M$ : Middle (or right) part of a string
R$ : Rule
S$ : Sample string so far
T$ : Then-part of Rule
V$ : Variable list
X$ : Input string
Y$ : Often a substitute for X$
Z$ : Anything
```

The line numbers are arranged as follows:

1- 29	Initialization
30- 99	Enter command and interpret
100- 199	NUMBER Command
200- 299	IF/IFF Command
300- 399	TELL Command
400- 499	GIVEN Command
500- 599	DELETE Command
600- 699	VARIABLE Command
700- 799	AUTOMATIC Command
800- 899	(not in use)
900- 999	COMMAND (Help!)
1000-1099	Proof Initialization
1100-1299	Pull apart rule
1300-1499	See which IF's might apply
1500-1599	Find upper IF
1600-1699	See if T/F conditions hold
1700-1999	Piece together Givens
2000-2299	See if trial fits
2300-2599	See if it is already deduced
2600-2699	Add trial as a deduction
2700-2799	Decrement counters and try again
2800-2899	Test for automatic mode

I shall now say a few quick words about PROVER. First, PROVER performs its proofs by trying out possible combinations of the givens to see if they fit the rules. It uses the last rules, then the last givens. Because of this, the most-used rules and givens should be entered last for optimum efficiency. PROVER will work without them in that order, but it will run somewhat slower.

If you ask PROVER a question it cannot answer, it may run quite awhile: at least, it will run until (1) it has gotten every possible deduction from the givens and rules or (2) it has run out of space. This is a problem I cannot solve; it assumes that there is an answer. After all, you could be asking for a 100-line proof from it, so it should not stop after the first few deductions.

One more item: PROVER cannot easily handle existence quantifiers. For x>y, it will give x=y+u; for a>b, it will give a=b+u. Thus, I must restrict the user somewhat: one may enter rules in the program with this type of condition, but beware. No two u's (in this case) are necessarily the same. With this in mind, the user is cautioned to check the program's results. Its purpose is to offer suggestions and possible methods of proofs. It is the duty of the user to check for existence quantifiers.

With no further ado, I present PROVER.

```
( RELEASE ^10.1)    B A S I C - V  LINK-READY COMPILER


 REM T IS THE DIMENSION SIZE
 T=100
0 DIM I$(100),T$(100),G$(100),G(100),Z$(210),S$(50),R(100)
1 DIM I(100),F(10),W(10),C$(26)
0 R=0
1 D=0
2 A=0
3 FOR I=1 TO T
4 I$(I)=""
5 T$(I)=""
6 G$(I)=""
7 G(I)=-1
8 NEXT I
9 V$=""
0 PRINT
5 PRINT "ENTER COMMAND"
0 INPUT X$
5 PRINT
0 L$=LEFT(X$,3)
0 IF L$="IS " THEN 1000
1 IF L$="NUM" THEN 100
2 IF L$="IF " THEN 200
3 IF L$="TEL" THEN 300
4 IF L$="GIV" THEN 400
5 IF L$="DEL" THEN 500
6 IF L$="VAR" THEN 600
7 IF L$="AUT" THEN 700
9 IF L$="COM" THEN 900
0 IF L$="NEW" THEN 20
1 IF L$="IFF" THEN 200
9 IF L$="END" THEN 9999
0 PRINT "INVALID COMMAND (ENTER 'COMMAND' FOR A LIST OF VALID COMMANDS)"
0 GOTO 40
00 PRINT "THERE ARE";R;"RULES AND";D;"DEDUCTIONS"
01 P$="OFFON "
02 PRINT "AUTOMATIC MODE IS NOW ";MID(P$,A*3+1,3)
10 GOTO 30
00 IF R<T THEN 230
10 PRINT "NO ROOM FOR MORE RULES"
20 GOTO 30
30 R=R+1
40 P=INSTR(0,X$," THEN ")
50 T$(R)=RIGHT(X$,P+6)
60 IF L$="IFF" THEN 290
70 I$(R)=MID(X$,4,P-4)
80 GOTO 35
90 I$(R)=MID(X$,5,P-5)
91 X$="IF "+T$(R)+" THEN "+I$(R)
92 L$=LEFT(X$,3)
93 GOTO 200
00 Z$(1)="THESE ARE THE RULES:"
01 Z$(2)=""
```

(RELEASE ^10.1)   B A S I C - V LINK-READY COMPILER

```
10 FOR I=1 TO R
20 Z$(I+2)="IF "+I$(I)+" THEN "+T$(I)
30 NEXT I
40 Z$(R+3)=""
41 Z$(R+4)="THESE ARE THE GIVENS/DEDUCTIONS:"
42 Z$(R+5)=""
49 P$="TRUE  FALSE "
50 FOR I=1 TO D
60 Z$(I+R+5)=MID(P$,G(I)*6+1,6)+G$(I)
70 NEXT I
71 Z$(R+D+6)=""
72 Z$(R+D+7)="THESE ARE THE VARIABLES: "+V$
80 FOR I=1 TO R+D+7
81 PRINT Z$(I)
82 Z$(I)=""
83 NEXT I
90 GOTO 30
00 IF D<T THEN 430
10 PRINT "NO ROOM FOR MORE GIVENS"
20 GOTO 30
30 D=D+1
40 P=INSTR(0,X$," ")
50 M$=RIGHT(X$,P+1)
60 L$=LEFT(M$,3)
70 IF L$="NOT" THEN 490
80 G$(D)=M$
81 G(D)=0
85 GOTO 30
90 G$(D)=RIGHT(M$,4)
91 G(D)=1
95 GOTO 30
00 PRINT "ENTER FIRST, LAST RULE; FIRST, LAST GIVEN TO DELETE."
01 PRINT "ENTER '1,0' FOR NO DELETION";
10 INPUT R1,R2,D1,D2
20 IF R1>R2 THEN 550
30 R=R+R1-R2-1
40 FOR I=R1 TO R
41 R0=I+R2-R1+1
42 I$(I)=I$(R0)
43 T$(I)=T$(R0)
49 NEXT I
50 IF D1>D2 THEN 580
60 D=D+D1-D2-1
70 FOR I=D1 TO D
71 D0=I+D2-D1+1
72 G$(I)=G$(D0)
73 G(I)=G(D0)
79 NEXT I
80 PRINT "DELETIONS COMPLETED"
90 GOTO 30
00 P=INSTR(0,X$," ")
10 IF P>0 THEN 640
```

```
20 PRINT "THE PROPER FORMAT IS 'VARIABLE XYZ'"
30 GOTO 30
40 V$=RIGHT(X$,P+1)
50 V=LEN(V$)
60 GOTO 30
00 P=INSTR(0,X$," ")
10 IF P=0 THEN 790
20 M$=RIGHT(X$,P+1)
30 IF M$="ON" THEN 770
40 IF M$<>"OFF" THEN 790
50 A=0
60 GOTO 780
70 A=1
80 PRINT "AUTOMATIC MODE IS NOW ";M$;"."
81 GOTO 30
90 PRINT "THE PROPER FORMAT IS 'AUTO ON' OR 'AUTO OFF'"
91 GOTO 30
00 PRINT "THE COMMANDS ARE:"
05 PRINT
10 PRINT "AUTOMATIC ON/OFF: ADJUSTS AUTOMATIC MODE"
15 PRINT "COMMAND: GIVES YOU THIS LIST"
20 PRINT "DELETE: DELETES RULES AND/OR GIVENS/DEDUCTIONS"
25 PRINT "END: TERMINATES PROGRAM"
30 PRINT "GIVEN XXX: ADDS XXX TO THE LIST OF GIVENS/DEDUCTIONS"
35 PRINT "IF XXX THEN YYY: ADDS THIS RULE (AND'S SHOULD BE SPACED)"
40 PRINT "IFF XXX THEN YYY: ENTERS IF XXX THEN YYY AND IF YYY THEN XXX"
45 PRINT "IS XXX: CHECKS TO SEE IF XXX IS A VALID DEDUCTION"
50 PRINT "NEW: CLEARS WORKSPACE (LIKE STARTING ALL OVER)"
55 PRINT "NUMBER: GIVES NUMBER OF RULES AND GIVENS/DEDUCTIONS"
60 PRINT "TELL: TELLS THE RULES AND GIVENS/DEDUCTIONS"
65 PRINT "VARIABLE XXX: ENTERS XXX AS THE LIST OF VARIABLES"
90 PRINT
91 PRINT "ALL COMMANDS MAY BE SHORTENED TO THREE (3) LETTERS"
92 GOTO 30
000 REM DEDUCTION MACHINE.  LINES 1000-2999 ARE
010 REM COPYRIGHT (C) 1983 WILLIAM W. BERGHEL.
020 REM
050 REM E$ IS WHAT WE'RE TRYING TO GET
060 E$=RIGHT(X$,4)
070 PRINT "OBJECT: ";E$
072 E=0
074 IF LEFT(E$,3)<>"NOT" THEN 1080
076 E$=RIGHT(X$,7)
078 E=1
080 D2=0
090 Y=0
100 REM
101 REM FIRST, PULL APART THE RULE INTO USEABLE FORM
102 REM
103 D1=D
105 FOR I0=R TO 1 STEP -1
110 R$=I$(I0)
```

```
111 P1=1
112 C=1
113 F(C)=0
114 M$=MID(R$,P1,3)
115 IF M$<>"NOT" THEN 1120
116 F(C)=1
117 L$=LEFT(R$,P1-1)
118 M$=RIGHT(R$,P1+3)
119 R$=L$+M$
120 P=INSTR(P1,R$," ")
121 IF P=0 THEN 1125
122 C=C+1
123 P1=P+1
124 GOTO 1113
125 S0=0
126 S=1
127 FOR I1=1 TO 50
128 S$(I1)=""
129 NEXT I1
130 FOR I1=1 TO LEN(R$)
140 R(I1)=0
150 M$=MID(R$,I1,1)
160 P=INSTR(0,V$,M$)
170 IF P=0 THEN 1210
180 R(I1)=P
185 IF S>S0 THEN 1240
190 S=S+1
200 GOTO 1240
210 S$=S$(S)
220 S$(S)=S$+M$
230 R(I1)=-S
235 S0=S
240 NEXT I1
245 S=S0
250 REM
260 REM TRY THE TRUTH OF THE THEN-PART OF THE RULE
270 REM
290 T$=T$(I0)
291 IF LEFT(T$,3)="NOT" THEN 1295
292 F=0
293 GOTO 1300
295 F=1
296 Z$=T$
297 T$=RIGHT(Z$,4)
300 REM
310 REM NOW, SEE WHICH IF'S COULD WORK
320 REM
330 FOR I1=1 TO D
340 I(I1)=1
350 X$=G$(I1)
360 FOR I2=1 TO S
370 IF INSTR(0,X$,S$(I2))>0 THEN 1400
```

```
380 NEXT I2
390 I(I1)=0
400 NEXT I1
500 REM
510 REM SET UP THE PROOF PART
511 REM
520 D0=D1
530 IF I(D0)>0 THEN 1570
540 D0=D0-1
550 IF D0<1 THEN 2800
560 GOTO 1530
570 FOR I1=1 TO C
580 W(I1)=D0
590 NEXT I1
600 REM
610 REM FIRST SEE IF THE CONDITIONS (T/F) HOLD
620 REM
630 FOR I1=1 TO C
640 IF G(W(I1))<>F(I1) THEN 2700
650 NEXT I1
700 REM
710 REM SECOND PIECE TOGETHER THE GIVENS
720 REM
730 X$=G$(W(1))
740 FOR I1=2 TO C
750 Y$=X$+" "+G$(W(I1))
760 X$=Y$
770 NEXT I1
780 PRINT "D=";D;"  R=";I0;"  DL=";D2;"  TR: ";X$;
000 REM
010 REM THIS IS THE BIGGIE !!!!!
020 REM TASK: SEE IF X$ FITS R$, AND IF SO, THEN HOW?
030 REM
040 REM
050 P0=1
051 FOR I1=1 TO 26
052 C$(I1)=""
053 NEXT I1
060 FOR I1=1 TO S
070 P=INSTR(P0,X$,S$(I1))
080 IF P=0 THEN 2700
090 M$=MID(X$,P0,P-P0)
100 IF R(1)=-I1 THEN 2180
110 FOR I2=2 TO LEN(R$)
120 IF R(I2)=-I1 THEN 2140
130 NEXT I2
140 IF LEN(C$(R(I2-1)))>0 THEN 2170
150 C$(R(I2-1))=M$
160 GOTO 2180
170 IF C$(R(I2-1))<>M$ THEN 2700
180 P0=P+LEN(S$(I1))
190 IF P0>LEN(X$) THEN 2260
```

```
200 NEXT I1
204 P1=LEN(R$)
205 IF R(P1)<0 THEN 2700
210 M$=RIGHT(X$,P0)
220 IF LEN(C$(R(P1)))>0 THEN 2250
230 C$(R(P1))=M$
240 GOTO 2260
250 IF C$(R(P1))<>M$ THEN 2700
260 REM
270 REM IT FITS !!!
280 REM
300 REM
310 REM FIGURE OUT WHAT IT GOES TO
311 REM
320 Z$=""
330 FOR I1=1 TO LEN(T$)
340 M$=MID(T$,I1,1)
350 P=INSTR(0,V$,M$)
360 IF P>0 THEN 2385
370 Y$=Z$+M$
380 GOTO 2400
385 IF LEN(C$(P))=0 THEN 2370
390 Y$=Z$+C$(P)
400 Z$=Y$
410 NEXT I1
411 PRINT TAB(50);"DED: ";Z$;
420 REM
430 REM SEE IF IT MATCHES ANYTHING
440 REM
450 FOR I1=1 TO D
460 IF Z$<>G$(I1) THEN 2520
470 REM IT'S ALREADY IN THE LIST !!!
480 REM FORGET IT IF THE TRUTH-PART MATCHES
490 IF G(I1)=F THEN 2700
495 PRINT:PRINT
500 PRINT "WE HAVE A CONTRADICTION!"
510 GOTO 30
520 NEXT I1
600 REM
610 REM WE HAVE A BRAND NEW DEDUCTION!
620 REM
621 REM
622 REM SEE IF IT MATCHES WHAT WE WANT
623 REM
624 IF Z$<>E$ THEN 2640
625 IF F<>E THEN 2630
626 REM IT DOES!!!
627 PRINT:PRINT:PRINT "YES!"
628 Y=-1
629 GOTO 2640
630 PRINT:PRINT:PRINT "NO!"
631 Y=-1
```

```
40 IF D<T THEN 2670
41 REM BUT NOT ENOUGH ROOM!
45 IF Y=-1 THEN 30
49 PRINT:PRINT
50 PRINT "I DON'T KNOW; I RAN OUT OF SPACE!"
60 GOTO 30
70 D=D+1
80 G$(D)=Z$
90 G(D)=F
95 IF Y=-1 THEN 30
00 REM
10 REM DECREMENT COUNTERS
20 REM
21 PRINT
30 W0=C
40 W(W0)=W(W0)-1
50 IF W(W0)<1 THEN 2780
60 IF I(W(W0))=0 THEN 2740
65 FOR I1=1 TO C
70 IF W(I1)>D2 THEN 1600
75 NEXT I1
80 W(W0)=D0
90 W0=W0-1
99 IF W0>0 THEN 2740
00 NEXT I0
05 REM IF WE'RE IN AUTOMATIC MODE, THEN CONTINUE; ELSE, QUIT
07 D2=D1
10 IF A=0 THEN 2840
20 IF D>D1 THEN 1100
30 PRINT "I DON'T KNOW; CERTAINLY NOT FROM WHAT YOU GAVE ME."
35 GOTO 30
40 PRINT "I DON'T KNOW; I DID NOT HAVE TIME TO SEE."
45 PRINT "IF YOU LET ME CONTINUE, I'LL TRY TO FIND OUT."
50 GOTO 30
99 END
```

SAMPLE RUN

The following is a sample run. The CAPITALS are what the program types; the lower case letters are the user entries but would be in capital letters in an actual run. Before every command PROVER types ENTER COMMAND, followed by a question mark. I shall abbreviate this by using only a question mark.

During the process of proving the theorem, PROVER prints four to five items: D, R, DL, TR, and maybe DED. D is the number of deductions already present, R is the rule we are trying to fit, DL is the deduction limit (no test case will be tried if all of its if-parts are below this limit), TR is the trial string, and DED, if present, is the deduction that fits the rules.

```
prover
?variables xyz
?if x=y y=z then x=z
?given a=b
?giv b=c
?is a=c

OBJECT: A=C
D= 2    R= 1    DL= 0    TR: B=C B=C
D= 2    R= 1    DL= 0    TR: B=C A=B
D= 2    R= 1    DL= 0    TR: A=B B=C        DED: A=C

YES!

?tel

THESE ARE THE RULES:

IF X=Y Y=Z THEN X=Z

THESE ARE THE GIVENS/DEDUCTIONS:

TRUE   A=B
TRUE   B=C
TRUE   A=C

THESE ARE THE VARIABLES: XYZ

?new
?var xyzuv
?iff x>y then x=y+u
```

```
?iff x<y then y=x+u
?if x=y then y=x
?giv a>b
?tell

THESE ARE THE RULES:

IF X>Y THEN X=Y+U
IF X=Y+U THEN X>Y
IF X<Y THEN Y=X+U
IF Y=X+U THEN X<Y
IF X=Y THEN Y=X

THESE ARE THE GIVENS/DEDUCTIONS:

TRUE  A>B

THESE ARE THE VARIABLES: XYZUV

?is b<a

OBJECT: B<A
D= 1   R= 1   DL= 0   TR: A>B          DED: A=B+U
I DON'T KNOW; I DID NOT HAVE TIME TO SEE.
IF YOU LET ME CONTINUE, I'LL TRY TO FIND OUT.

?is b<a

OBJECT: B<A
D= 2   R= 5   DL= 0   TR: A=B+U        DED: B+U=A
D= 3   R= 4   DL= 0   TR: A=B+U        DED: B<A

YES!

?tell

THESE ARE THE RULES:

IF X>Y THEN X=Y+U
IF X=Y+U THEN X>Y
IF X<Y THEN Y=X+U
IF Y=X+U THEN X<Y
IF X=Y THEN Y=X

THESE ARE THE GIVENS/DEDUCTIONS:

TRUE  A>B
TRUE  A=B+U
TRUE  B+U=A    {note the unnecessary step -- wwb}
```

```
TRUE   B<A

THESE ARE THE VARIABLES: XYZUV

?del

ENTER FIRST, LAST RULE; FIRST, LAST GIVEN TO DELETE.
ENTER '1,0' FOR NO DELETION?1,0,2,4
DELETIONS COMPLETED

?auto on

AUTOMATIC MODE IS NOW ON.

?num

THERE ARE 5 RULES AND 1 DEDUCTIONS
AUTOMATIC MODE IS NOW ON

?is b<a

OBJECT: B<A
D= 1   R= 1   DL= 0   TR: A>B           DED: A=B+U
D= 2   R= 5   DL= 1   TR: A=B+U         DED: B+U=A
D= 3   R= 4   DL= 1   TR: A=B+U         DED: B<A

YES!

?end
```

BIBLIOGRAPHY

Barr, Avron, and Feigenbaum, Edward A., The Handbook of Artificial Intelligence, Volume I, William Kaufman, Inc., Los Altos, California, 1981.

Boyer, Robert S., and Moore, J. Strother, A Computational Logic, Academic Press, Inc., New York, 1979.

Chang, Chin-Liang, and Lee, Richard Char-Tung, Symbolic Logic and Mechanical Theorem Proving, Academic Press, Inc., New York, 1973.

Dreyfus, Hubert L., What Computers Can't Do: A Critique of Artificial Reason, Harper and Row, Publishers, Inc., New York, 1972.

Johnson, Robert S., Class Notes from Mathematics 301 (Fundamental Concepts of Mathematics) (adapted from Landau, Edmund, Foundations of Analysis, Chelsea Publishing Company, New York, 1966).