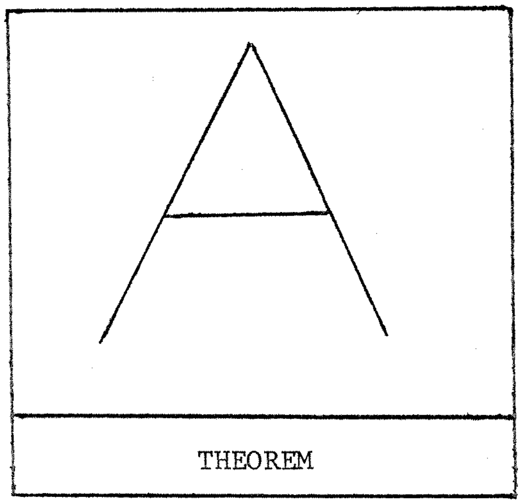
7. Interactive System

Large Data base problem.

One of the irksome things about most automatic theorem proving systems, is that the human user has to prove the theorem before he asks the computer to do so.

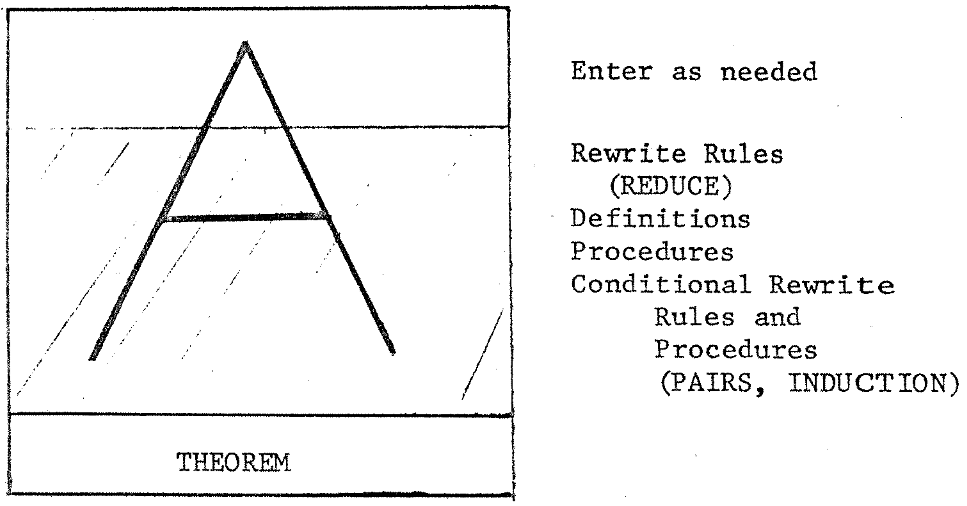


In this figure we depict a theorem to be proved, along with the "Axioms" or reference theorems needed for the proof. If we don't list enough reference theorems then the automatic prover cannot succeed; on the other hand, if we list too many, the prover again cannot succeed because it will be overwhelmed with too much data. So we cannot just list "all known theorems" as hypothesis and expect success, because most provers will then hang up on computing many useless inferences (lemmas) which have nothing to do with the objective at hand.[15]

[15]A few recent programs have attacked the large data base problem [18, 41, 42] with some success.

Thus in order to determine exactly the correct reference theorems needed, the human user is forced to prove the theorem first.

We have partially eliminated this problem by storing information in the form of definitions, rewrite (REDUCE) tables, and procedures, which are used only as needed and in no way clutter up the system (see Section 3).



The remainder of the difficulty is eliminated by having the human user insert references theorems <u>only as they are needed</u>, <u>during</u> the actual proof. Of course he will have to know when to do this, and what to insert.

<u>Hard Theorems</u>.

Equally irksome is the fact that present programs cannot prove very hard theorems. So they don't get involved with very <u>interesting mathematics</u>, and don't come to grips with some of the problems that the computer will have to face if we are to have acceptable computer mathematics.

Man-Machine.

For these reasons, and others, we have decided to include the theorem prover described in Sections 1-6 above, as part of a man-machine interactive, prover.

The System.

The system consists of one or more interactive computer terminals connected to a large digital computer. At present we are using the CDC 6600 and PDP-10 computers at The University of Texas, and the UT time sharing system. A version of the program is also running on the DEC 10 computer at the Information Sciences Institute, USC, Los Angeles.

The system was developed at UT, MIT, and ISI, by the authors, and Bob Boyer, Robert Anderson, Peter Bruell, Mike Ballantyne, Bill Bennett and Larry Fagan.

This system has much in common with earlier programs of Guard, et al [10], Allen and Luckham [9] (especially with recent additions [33]), and Huet [11], but is quite different, (e.g., in its use of DETAIL, PUT, etc., defined below and does not use Resolution).

User Requirements.

We believe that such a system must be built for the convenience of the user and not the programmer. For otherwise, the system will not be used. As long as the pain in using the system exceeds the help obtained, the potential user will stay away.

In order to interact effectively, the user must be able to

(1) *Read* and easily *comprehend* the scope

(2) *Follow* the proof

(3) Help the computer *only* when needed

(4) Use convenient commands

The UT interactive Prover.

In our system the formulas which are being proved appear on the terminal screen in an infix notation. For example, the formula whose internal representation is $(\longrightarrow (\wedge\ A\ B)\ P)$ would appear on the screen as

$$\begin{array}{c} A \\ \wedge \\ B \\ \longrightarrow \\ C \end{array}$$

Larry Fagan has recently developed a package for the DEC 10 at ISI to allow a formula to remain stationary on the scope while other material is scrolled up in a normal fashion. He and Mabry Tyson have adopted it to also operate on the CDC 6600 at UT. Having the theorem (or current subgoal) remain stationary on the scope during the proof is a great help to the user in understanding and following the proof.

The user has at his disposal a set of options which give (interactive) commands to the program. Some of these cause information to be displayed on the terminal screen, while others affect the course of the proof.

Table VII gives a listing of some of the interactive commands being used. A few of these are further explained below. In the following, the work "theorem" is used to represent the current subgoal being proved.

Most of the human input (i.e., the use of the interactive commands) takes place at "IMPLY STOP", a point in the routine IMPLY, near its beginning. The program is a slave to the user, working on tasks assigned it by the user. It halts at IMPLY STOP and reports after such a calculation. It may report

"PROVED"

or

"FAILED"

or other things described below.

Table VII

Some Interactive Commands

Name of command	User types	The machine's response
PRETTY-PRINT	TP	It prints the theorem on the scope in an easily readable form (see example below).
	TP F	If PUT F = ( ) has been used earlier, it prints the theorem on the scope with each occurrence of ( ) replaced by the symbol F.
	TP F G ...	Similarly for F, G, etc.
	TPC F	Similarly for conclusion only.
	TPH F	Similarly for hypothesis only.
	TL	It prints the theorem label.
	TY	It pretty-prints TYPELIST.
	TPR	It pretty-prints the REDUCE table.
ADD-DEFN	ADD-DEFN A ( )	( ) is added to the definition table as the definition of the expression A.
ADD-REDUCE	ADD-REDUCE ( )	( ) is (permanently) added to the REDUCE table.
ADD-PAIRS	ADD-PAIRS ( )	( ) is (permanently) added to the PAIRS table.
DEFN	D A	It replaces all occurrences of A by its (stored) definition.
	DC	It defines the conclusion of the the current goal.
USE	USE N	It fetches theorem number N from memory and adds it to the hypothesis of the current theorem.
	USE ( )	It adds ( ) to the hypothesis.

Name of command	User types	The machine's response
LEMMA	LEMMA ( )	It first proves ( ) and then calls USE ( ).
SUBGOAL	SUBGOAL A	It calls (Lemma $(H \longrightarrow A)$) where H is the current hypothesis.
PROCEED	CONTINUE	It proceeds with the proof with no changes by the user.
	GO	Exit current subgoal with "PROVED" or "FAILED" as was determined by the program.
TIMELIMIT	CNT N	It increases the timelimit on the current subgoal by a factor N.
ASSUME	A	It assumes the current subgoal to be proved and proceeds.
FAIL	F	It fails the current subgoal (i.e., returns NIL).
BACKUP	BACK	It backs up to the previous preset back-up point.
	B	Create a backup point.
REORDER	$(N \longrightarrow M)$	It reorders the goal, placing hypothesis number N first and conclusion number M first.
	$(N1\ N2 \ldots \longrightarrow C)$	It reorders the goal placing hypothesis number N1 N2 ... first in that order.
	$(H \longrightarrow M1\ M2 \ldots)$	Similarly for conclusions M1 M2 ...
	$(N1\ N2 \ldots \longrightarrow M1\ M2 \ldots)$	Similarly for both.

Name of command	User types	The machine's response
DELETE	DELETE N M ...	It deletes hypotheses number N, M,...
PUT	PUT X ( )	The machine replaces each occurrence of x in theorem being proved, by ( ).

<u>Time Limit</u>.

In all cases the program worker under a time limit determined by the user. If it does not prove the current subgoal within that time limit it will halt and report

```
"FAILED TIMELIMIT"
```

The time limit can be increased (or decreased) by use of the command (CNT N) (See Table VII).

<u>Pretty-Print</u>.

The command TP causes the machine to print the current theorem (subgoal) in a parsed, easy to read form. For example, if the theorem is

```
(⟶ (∧ (OC (FSDI))(∧ (REG)(OCLFR)))(∧ (CC G)(∧ (REF G(FSDI))(LF G))))
```

the command TP will cause to be printed on the scope:

```
       (OC (F))
     ∧
           (REG)
         ∧
           (OCLFR)
  ⟶
       (CC G)
     ∧
           (REF G (F))
         ∧
           (LF G)
```

Note that the skolem constant (FSDI) has been printed as (F), though its complete form is retained by the program.

Now if the command

```
PUT G {C: Closed C}
```

is used, the conclusion is altered accordingly. The command TPC if issued now will cause

$$\begin{array}{l} (\mathrm{CC}\{C\colon \mathrm{Closed}\ C\}) \\ \quad\wedge \\ \qquad (\mathrm{REF}\{C\colon \mathrm{Closed}\ C\}(F)) \\ \qquad\wedge \\ \qquad (\mathrm{LF}\{C\colon \mathrm{Closed}\ C\}) \end{array}$$

to be printed, whereas TPC G causes

$$\begin{array}{l} (\mathrm{CC}\ G) \\ \wedge \\ \quad (\mathrm{REF}\ G\ (F)) \\ \quad\wedge \\ \quad (\mathrm{LF}\ G) \end{array}$$

to be printed.

ADD-DEFN, ADD-REDUCE, and ADD-PAIRS allows the user to easily add entries in Tables III, IV and VI.

The command "D A" causes the program to expand the definition of A through the theorem. For example, if the current subgoal is

$$(\mathrm{Oc}\ F \longrightarrow \mathrm{Cover}\ F)$$

and the command "(D Oc)" is issued by the user, then the subgoal is changed to

$$(\mathrm{Open}\ F \wedge \mathrm{Cover}\ F \longrightarrow \mathrm{Cover}\ F)$$

"(DH A)" and "(DC A)" would cause such changes only in the hypothesis or the conclusion respectively.

(USE A) simply allows the user to add the additional hypothesis A, whereas (LEMMA A) requires the prover to prove A first and then add it as a hypothesis, whereas (SUBGOAL A) calls (LEMMA $(H \rightarrow A)$) where H is the current hypothesis.

The commands A (for "ASSUME") and F (for "FAIL") are useful for terminating a long proof or for maneuvering the proof to parts of the theorem that the user is interested in.

The command $(n\ m \longrightarrow i\ j)$ causes the hypotheses and conclusions to be reordered with hypothesis number n first, and number m second, and with conclusion number i first and number j second, etc.. The command (DELETE n m...) causes hypotheses numbers n, m,... to be deleted. For example if the current goal is

$$\begin{array}{ll} & (x_o \in A) \\ \wedge & (x_o \in B) \\ \wedge & (t \in A \rightarrow t \in C) \\ \wedge & \text{Open } A \\ \longrightarrow & \\ & (x_o \in C) \\ \wedge & \text{Open } B \end{array}$$

and the command $(4\ 1\ 3 \longrightarrow 2)$ is issued the goal is changed to

$$\begin{array}{ll} & \text{Open } A \\ \wedge & (x_o \in A) \\ \wedge & (t \in A \rightarrow t \in C) \\ \wedge & (x_o \in B) \\ \longrightarrow & \\ & \text{Open } B \\ \wedge & (x_o \in C)\quad , \end{array}$$

and if the command (DELETE 2 4 1) is now issued the goal is changed to

$$\begin{array}{ll} & (t \in A \rightarrow t \in C) \\ \longrightarrow & \\ & \text{Open } B \\ \wedge & (x_o \in C)\quad . \end{array}$$

PUT is one of the most important commands. It allows us to instantiate a skolem variable with a desired formula. For example, if the prover is trying to prove the theorem

$$\forall\ x(x \in A \longrightarrow \exists B(\text{Open } B \wedge B \subseteq A \wedge x \in B)) \\ \longrightarrow \exists F(F \subseteq \text{OPEN} \wedge A = \text{U } F)$$

it will obtain the goal

$$(1)\qquad ((\overbrace{x \in A_o \longrightarrow \text{Open } B_o \wedge B_o \subseteq A_o \wedge x \in B}^{\alpha}) \\ \longrightarrow F \subseteq \text{OPEN} \wedge A_o = \text{U } F)$$

At this point the machine may be unable to determine the required family F of open sets whose union is A_o. If the user decides to help, he can easily do so by using the command "PUT" to give a value to F. For example the command (PUT F (OPEN $\cap$ Subsets A_o)) will cause (1) to be changed to

(1) $(\alpha \Rightarrow (\text{OPEN} \cap \text{Subsets } A_o) \subseteq \text{OPEN} \wedge \bigcup (\text{OPEN} \cap \text{Subsets } A_o) = A_o)$,

which is easily proved, as follows.

(1 1) $(\alpha \Rightarrow (\text{OPEN} \cap \text{Subsets } A_o) \subseteq \text{OPEN})$ I 4

$(\alpha \Rightarrow (B_o \in (\text{OPEN} \cap \text{Subsets } A_o) \longrightarrow B_o \in \text{OPEN}))$ I 13

$(\alpha \wedge \text{Open } B_o \wedge B_o \subseteq A_o \Longrightarrow \text{Open } B_o)$, TRUE I 7, 5

(1 2) $(\alpha \Rightarrow \bigcup (\text{OPEN} \cap \text{Subsets } A_o) = A_o)$ I 4.2

(1 2 1) $(\alpha \Rightarrow \bigcup (\text{OPEN} \cap \text{Subsets } A_o) \subseteq A_o)$ I 13, I 4

$(\alpha \Rightarrow x_o \in \bigcup (\qquad\qquad) \longrightarrow x_o \in A_o)$ I 13

$(\alpha \Rightarrow (B_o \in (\text{OPEN} \cap \text{Subsets } A_o) \wedge x_o \in B_o \longrightarrow x_o \in A_o))$ I 5

$(\alpha \wedge \text{Open } B_o \wedge B_o \subseteq A_o \wedge x_o \in B_o \wedge x_o \in A_o \Longrightarrow x_o \in A_o)$
TRUE

Forward Chaining was used in the previous step.

(1 2 2) $(\alpha \Rightarrow A_o \subseteq \bigcup (\text{OPEN} \cap \text{Subset } A_o)$ I 4.2

$(\alpha \wedge x_o \in A_o \Rightarrow \text{Open } B \wedge B \subseteq A_o \wedge x_o \in B)$ I 13, 7, 5

$(x_o \in A_o)$ is forward chained into α to get

$(\text{Open } B_o \wedge B_o \subseteq A_o \wedge x_o \in B_o)$. Thus (1 2 2) becomes

(1 2 2) $(\alpha \wedge x_o \in A_o \wedge \text{Open } B_o \wedge B_o \subseteq A_o \wedge x_o \in B_o \Rightarrow \text{Open } B \wedge B \subseteq A_o \wedge x_o \in B)$,

which holds for B_o/B. Thus B_o/B is returned for (1 2 2), (1 2), and (1).

Other examples using PUT and the other interactive commands are given in [1].

The DETAIL command is fully explained in Section 2.1 of [1] and is used in several examples there. It is used to let the machine tell the user which part of the proof it is having trouble with. If the prover fails on a goal of the type

(1 2) $(H \Rightarrow A \wedge B)$

the command "DETAIL" will (in essence) ask for information on the proof of each of the subgoals $(H \rightarrow A)$ and $(H \rightarrow B)$. Thus the machine might respond

(1 2 1) $(H \rightarrow A)$

PROVED..

Then after a user commands "PROCEED", it might respond

(1 2 2) $(H \rightarrow B)$

FAILED..

In this way the user is told which of the two subgoals the prover is having trouble with, and can direct his help accordingly. A further command "DETAIL" would act similarly on subgoals of B (if there are any).

INDUCTION K commands the program to try to prove the current subgoal using mathematical (finite) induction on K. For example if the current goal is

$$P(K)$$

it will rewrite it as

$$P(0) \wedge (P(K) \rightarrow P(K+1)) ,$$

also universally quantifying any free variables in $P(K)$. For example,

$$\sum_{i=0}^{N} i = N(N+1)/2$$

is converted by the command INDUCTION N, to

$$\begin{aligned} &(\sum_{i=0}^{0} i = 0(0+1)/2) \\ \wedge \\ &(\sum_{i=0}^{N} i = N(N+1)/2 \rightarrow \sum_{i=0}^{N+1} i = (N+1)((N+1)+1)/2 \end{aligned}$$

which can now be proved automatically by the use of a simplification routine and REDUCE rewrite rules which convert

$$\sum_{i=0}^{0} f(i)$$

to 0, and convert

$$\sum_{i=0}^{K+1} f(i)$$

to

$$\sum_{i=0}^{K} f(i) + f(K+1) .$$

In many examples the subgoal itself is not a sufficient induction hypothesis. Thus it is necessary to "prove more" in order to get the desired result. To facilitate this, the command (INDUCTION K Q) can be used whereby the user supplies the induction hypothesis Q.

Many other researchers have used induction in their automatic theorem proving programs [40,2,43,44,29]. Boyer and Moore [29] have employed an interesting concept called "generalization" which converts the current subgoal into a more general theorem, but one which then can be proved by induction.

Optional REDUCE.

For some large theorems, for example like those encountered in program verification (see Part I), it is not desirable to call REDUCE each time IMPLY is executed, because this can be very time consuming. Accordingly the program can be operated in a mode that causes it to stop before executing Rule 7 of IMPLY, and print "DO YOU WANT TO REDUCE? TYPE: Y or N". A "N" will cause it to proceed without reducing.

User Equals Substitution.

In Rule 9 of HOA the program applys a "substitution of equals". Given a hypothesis $(a = b)$ it selects either a or b and replaces it by the other throughout H and C. An optional mode of operation is provided that allows the human user to override this process. In this mode the program chooses either a or b and proposes that to the user for him to accept as proposed, reverse, or reject altogether. The program stops and prints

"a Replaced by b?"

The user then says one of:

"Yes"	(means do the substitution)
"R"	(means replace b by a)
"No"	(means do not do any substitution. Proceed)
"Next"	If $b \equiv c + d$ (or $a = e + f$) the program will find the next possible substitution and print "c Replaced by d-a?"

and the whole process repeats.

Most of these interactive commands are retractable. If a command has changed the theorem in any way, the machine displays the changed version and then asks "OK???". The program will then make the change permanent only if the user types "OK".

All the user commands such as PUT, USE, etc., may be called initially without arguments. When this happens the program asks the user for the required arguments. For example the following is a sample dialogue where "h" stands for user input and "c" for machine quiries.

```
c        IMPLY STOP
h        ADD-REDUCE
c        PATTERN:
h        a + 0
c        REWRITE AS:
h        a
c        CONDITIONAL (YES OR NO)?
h        NO
```

Some More Details.

In Section 1 we said that IMPLY was called with five arguments, (TYPELIST, H, C, TL, LT), but only the principal ones, H, C, and the theorem label TL, were discussed in Sections 1-6. TYPELIST is discussed in [27] and LT is a "light" which helps control the man-machine interaction. This is discussed below.

The routine IMPLY has three major sections -- CNTRL (control), OPTIONS, and the features described in Sections 2-6. CNTRL is executed at the entry to IMPLY and is the only section that uses the light (LT). The purpose of LT is to differentiate between calls to DETAIL $(LT = 3)$, CNT $(LT = 5)$, which are described above, and $(LT = B)$. A DETAIL call stops at OPTIONS before returning from the top level sub-calls to IMPLY. A CNT call (which gives a larger time-limit) does not stop at OPTIONS on any sub-call. A "B" call stops at OPTIONS before any proof is attempted. If the theorem is $(H \rightarrow C_1 \wedge C_2)$, there are top-level sub-calls to IMPLY for $(H \rightarrow C_1)$ and $(H \rightarrow C_2)$. DETAIL stops at OPTIONS after $(H \rightarrow C_1)$ is attempted and then after $(H \rightarrow C_2)$ is attempted. CNT does not stop until after the attempt to prove $(H \rightarrow C_1 \wedge C_2)$ is completed. A "B" call stops before $(H \rightarrow C_1)$ is attempted. CNTRL does various things according to the value of LT. If $LT = 3$, CNTRL resets LT to 3 and goes directly down to OPTIONS for human intervention. If $LT = 1$ or $LT > 2$, CNTRL recalls IMPLY with LT one less than its present value. The result of this call ("PROVED", "FAILED", or "PROVED CONDITIONALLY" -- see Section 3 of Part I) is printed and execution continues at OPTIONS. In most cases control passes down to OPTIONS directly.

At OPTIONS human intervention is bypassed if $LT < 1$ or $LT = 2$ and control is passed on to the IMPLY Rules (See Table I). Otherwise "IMPLY-STOP" is printed on the screen and the program waits for the user to enter commands. At present there are about 35 different commands available, including those listed in Table VII. As mentioned earlier, some commands cause information to be printed, some cause special proving methods to be tried (perhaps with a larger "time" limit or with more human intervention). Some allow the user to give prover more information, or to arrange the theorem.

8. Some Applications

This prover has been used to prove theorems in the following areas:

(1) Set Theory [2]

(2) Limit Theorems of Calculus [3]

(3) Topology [1, 38, 39]

(4) Limit Theorem of Analysis [45]

(5) Program Verification [6, 27].

In [45] the methods of non-standard analysis are used whereby the theorem in question is converted automatically to a theorem in non-standard analysis, and then proved in the new setting which seems to be more conductive to automatic proofs. The typing concepts (see Section 1 of [27]) and Reductions (see Section 3) play a major role in handling infinitesimals, and other <u>typed</u> quantities.

Appendix 1

Skolemization -- Elimination of Quantifiers

First we give examples. The formula

$$\exists x\ P(x)$$

is skolemized as

$$P(x)$$

where x is a "skolem variable" which can be replaced by any term during the proof. Similar,

$$Q \rightarrow \exists x\ P(x)\ ,$$

and

$$\exists x (Q \longrightarrow P(x))\ ,$$

are skolemized as

$$Q \longrightarrow P(x)\ ,$$

and

$$(\forall x\ P(x) \longrightarrow C)$$

is skolemized as

$$(P(x) \longrightarrow C)\ ,$$

where x is a skolem variable.

On the other hand, the formulas

$$\forall x\ P(x)\ ,$$
$$Q \longrightarrow \forall x\ P(x)\ ,$$
$$\forall x (Q \longrightarrow P(x))\ ,$$
$$(\exists x\ P(x) \longrightarrow C)\ ,$$

are skolemized as

$$P(x_o)\ ,$$
$$(Q \longrightarrow P(x_o))$$

and

$$(P(x_o) \longrightarrow C)\ ,$$

where x_o is a skolem constant (cannot be replaced).

Finally

$$(\forall x \exists y\ P(x,y) \longrightarrow \exists u \forall v\ Q(u,v))$$

is skolemized as

$$(P(x, g(x)) \longrightarrow Q(u, h(u)))$$

where g and h are skolem functions.

Notice that we do <u>not</u> place the formula being skolemized in prenex form, but skolemize it in place, leave each logical symbol except $\forall$ and $\exists$, in its original position.

We now give the general rules.

Given a formula E, we recursively define[2] as "positive" or "negative" the subformulas of E, as follows:

1. E is positive
2. If $(A \wedge B)$ is positive (negative) then so are A and B
3. If $(A \vee B)$ " " " " " " " " "
4. If $\sim A$ is positive (negative) then A is negative (positive)
5. If $(A \longrightarrow B)$ is positive (negative) then
   A is negative (positive), and
   B is a positive (negative)
6. If $(\forall x\ A)$ is positive (negative) then
   A is positive (negative), and
   $\forall$ is a positive (negative) quantifier
7. If $(\exists x\ A)$ is positive (negative) then
   A is positive (negative), and
   $\exists$ is a negative (positive) quantifier.

For example if E is the formula

$$([H \longrightarrow (C \longrightarrow \sim D)] \longrightarrow [\sim A \vee (B \longrightarrow F)])$$

then E, $[\sim A \vee (B \longrightarrow F)]$, $\sim A$, $(B \longrightarrow F)$, F, H, C and D are positive, while $[H \longrightarrow (C \longrightarrow \sim D)]$, $(C \longrightarrow \sim D)$, $\sim D$, A, and B are negative.

Given a formula E with no free variables, we eliminate the quantifiers of E by deleting each quantifier and each variable immediately after it, and replacing each variable v bound by a positive quantifier with the skolem

[2]See Wang [23].

expression $g(x_1,\ldots,x_n)$ where g is a new function symbol (a "skolem function" symbol) and $x_1,\ldots,x_n$, consists of those variables of E which are bound by negative quantifiers whose scope includes v. The result is called the "skolem form of E".

For example if E is the formula

$$\exists x \forall y \; P(x,y)$$

then $\exists$ is a negative quantifier, $\forall$ is a positive quantifier, and

$$P(x,g(x))$$

is the skolem form of E, whereas the formulas

$$\forall x(P(x) \longrightarrow \exists y \; Q(x,y)) \; ,$$

and

$$\exists x(\forall y \exists z \; P(x,y,z) \longrightarrow \forall w \; Q(x,w))$$

have the skolem forms

$$(P(x_o) \longrightarrow Q(x_o,y)) \; ,$$

and

$$(P(x,y,g(x,y)) \longrightarrow Q(x,h(x)))$$

respectively, and the formula

$$\forall x(\forall y[\exists z \; H(x,y,z) \longrightarrow \exists u(C(x,u) \longrightarrow \sim \forall v \; D(u,v))]$$
$$\longrightarrow \forall w[\sim A(x,w) \vee \exists s(\exists t \; B(s,t) \longrightarrow \forall r \; F(x,r,s,t))])$$

has skolem form

$$([H(x_o,y,z) \longrightarrow (C(x_o,g(y)) \longrightarrow \sim D(g(y),h(y)))]$$
$$\longrightarrow [\sim A(x_o,w_o) \vee (B(s,j(s) \longrightarrow F(x_o,k(s),s,j(s))]) .$$

It should be noted (by those familiar with Resolution proofs) that the formula E is <u>not</u> first negated before the skolem form is derived. This difference reverses the roles of $\forall$ and $\exists$ in the skolemization process.

Appendix 2

Incompleteness of the Prover

As mentioned earlier the prover is incomplete, in that there are theorems that it cannot prove. Of course, it has the usual incompleteness, that humans and other systems possess, of not being able to prove really hard theorems, like Fermat's last theorem (if it is a theorem). But our system is incomplete in three other ways which we refer to under the headings of "using ANDS in backchaining", "trapping", and "multiple copies". We will discuss these and changes we could make to eliminate this incompleteness, and why it is not desirable to do so. See also [46].

Using ANDS in Backchaining.

In Rule 7 of HOA if the hypothesis B has the form $(A \longrightarrow D)$ then we put

(i) $$\theta := \text{ANDS}(D, C)$$

and then try to prove $(H \longrightarrow A\theta)$. A more complete procedure would use

(ii) $$\theta := \text{IMPLY}(D \wedge H,\ C)$$

bringing to bear the whole strength of IMPLY instead of the weaker routine ANDS. The following example, illustrates this inadequacy

Ex. A1. $(A \wedge B \wedge (A \longrightarrow (B \longrightarrow C)) \longrightarrow C)$.

Of course forward chaining would quickly prove this, but let us assume, for the sake of the point we are making, that forward chaining is inoperative. Then we obtain:

(1)	$(A \wedge B \wedge (A \longrightarrow (B \longrightarrow C)) \Rightarrow C)$		I 7
	$(A \Rightarrow C)$	NIL	H 6
	$(B \Rightarrow C)$	NIL	H 6
	$((A \longrightarrow (B \longrightarrow C)) \Rightarrow C)$		H 6
	$\text{ANDS}(B \longrightarrow C,\ C)$	Returns NIL	H 7

So NIL is returned for (1).

If in Rule 7 of HOA, we had used (ii) instead of (i), (Calling it Rule 7'), then the proof would proceed to a successful conclusion as follows:

(1)	$(A \wedge B \wedge (A \longrightarrow (B \longrightarrow C)) \Rightarrow C)$		I 7
	$(A \Rightarrow C)$	NIL	H 6
	$(B \Rightarrow C)$	NIL	H 6
	$(A \longrightarrow (B \longrightarrow C)) \Rightarrow C$		H 6
(1 C)	$((B \longrightarrow C) \wedge A \wedge B \wedge (A \longrightarrow (B \longrightarrow C)) \Rightarrow C)$		H 7'
(1 C C)	$(C \wedge (B \longrightarrow C) \wedge A \wedge B \wedge (A \longrightarrow (B \longrightarrow C)) \Rightarrow C)$		H 7'
		"T"	H 6, 112
(1 C H)	$((B \longrightarrow C) \wedge A \wedge B \wedge (A \longrightarrow (B \longrightarrow C)) \Rightarrow B)$		H 7.2
		"T"	H 6, 2
(1 H)	$(A \wedge B \wedge (A \longrightarrow (B \longrightarrow C)) \longrightarrow A)$		H 7.2
		"T"	H 6, 2

However we do not use Rule 7' because it causes the procedure to spend too much time trying to prove the subgoal IMPLY$(D \wedge H \longrightarrow C)$ in cases where that is impossible, instead of proceeding with other lines of attack. Using ANDS instead of IMPLY prevents this futile attempt at backchaining, allowing it to happen only in cases when D essentially matches C.

Trapping.

In Rule 4 Table I when a conclusion of the form

$$A \wedge B$$

is being proved, we first prove A, getting a substitution θ, and then prove $B\,\theta$. Sometimes, as in the following example, the value of θ returned by IMPLY for the proof of A, will not work for B. We call this "trapping".

Ex. A2. $(P(a) \wedge P(b) \wedge Q(b) \longrightarrow \exists x(P(x) \wedge Q(x)))$

(1)	$(P(a) \wedge P(b) \wedge Q(b) \Rightarrow P(x) \wedge Q(x))$	I 7
(1 1)	$(P(a) \wedge P(b) \wedge Q(b) \Rightarrow P(x))$	I 4
	Returns $a \mid x$ for (1 1)	H 6, H 2
(1 2)	$(P(a) \wedge P(b) \wedge Q(b) \Rightarrow Q(a))$	I 4.2
	Returns NIL for (1 2)	
	Returns NIL for (1)	I 4.3

We could have prevented trapping in this example by trying $Q(x)$ first and then $P(x)$. So in general, when proving $(P(x) \wedge Q(x))$ we might want to first try $(P(x) \wedge Q(x))$ and if that fails then try $(Q(x) \wedge P(x))$. This could be effected by changing Rules 4.3 and 4.4 of Table I to read

4.3'	$\lambda \equiv$ NIL	Put θ': = IMPLY(H,B)	
4.3.1	$\theta' \equiv$ NIL		NIL
4.3.2	$\theta' \equiv$ NIL	Put λ': = IMPLY(H,Aθ')	
4.3.2.1	$\lambda' \equiv$ NIL		NIL
4.3.2.2	$\lambda' \not\equiv$ NIL		$\theta' \circ \lambda'$
4.4'	$\lambda \not\equiv$ NIL		$\theta \circ \lambda$

A similar permutation would have to been provided for in the hypothesis in order to handle an example like

<u>Ex. A3</u>. $(P(a) \wedge Q(b) \wedge P(c) \wedge Q(c) \longrightarrow \exists x(P(x) \wedge Q(x)))$

because there either $P(x)$ or $Q(x)$ first would produce trapping. So it would be necessary to permute the hypothesis to $(Q(b) \wedge P(c) \wedge P(a) \wedge Q(c))$ or one of the other successful configurations. This could be effected by further changing 4.3.2.1 to

4.3.2.1'	$\lambda' \equiv$ NIL	
4.3.2.1.1	H $\not\equiv$ (D $\wedge$ E)	NIL
4.3.2.1.2	H $\equiv$ (D $\wedge$ E)	IMPLY(E $\wedge$ D, A $\wedge$ B).

(In this case we would need to set and test a light to prevent Step 4.3.2.1.2 from being repeated indefinitely).

Unfortunately such changes as these greatly increase the running time (sometimes by orders of magnitude) for all theorems including those that need no such permutations. So our present version has neither change. Most of the examples encountered so far don't require it. Also in our interactive system (see Section 7) such a failure is shown to the human operator who can permute the conclusions and/or the hypotheses with one simple command and thereby achieve the desired result.

Nevins [17] has prevented trapping on an AND-SPLIT

$$(A(x) \wedge B(x))$$

by obtaining the set S_A of all values of x satisfying $A(x)$, and the set S_B of all values of x satisfying $B(x)$, and then intersecting S_A and S_B for the solution of $(A(x) \wedge B(x))$. See also [47]. In using this method one must be careful to provide a special mechanism for cases where S_A or S_B is infinite.

Multiple Copies.

In the following example the hypothesis $\forall x\, P(x)$ is used twice in proving the theorem.

<u>Ex. A4</u>. $(\forall x\, P(x) \longrightarrow P(a) \wedge P(b))$

(1)	$(P(x) \Rightarrow P(a) \wedge P(b))$		I 7
(1 1)	$(P(x) \Rightarrow P(a))$	a/x	I 4, H 2
(1 2)	$(P(x) \Rightarrow P(b))$	b/x	I 4.2, H 2

Returns (b/x, a/x) for (1).

There is no problem here, and also we have no difficulty with the following equivalent version of this example.

<u>Ex. A5</u>. $\exists x(P(x) \longrightarrow P(a) \wedge P(b))$

(1)	$(P(x) \longrightarrow P(a) \wedge P(b))$		I 7

etc. as in the previous example.

However, in the following equivalent version we reach an impass.

<u>Ex. A6</u>. $\exists x[(P(x) \longrightarrow P(a)) \wedge (P(x) \longrightarrow P(b))]$

(1)	$(P(x) \longrightarrow P(a)) \wedge (P(x) \longrightarrow P(b))$		
(1 1)	$(P(x) \Rightarrow P(a))$	a/x	I 4, I 7, H 2
(1 2)	$(P(a) \Rightarrow P(b))$	NIL	I 4.4, I 7

NIL is returned for (1).

If the program was able to convert this example to its equivalent form

$$\exists x(P(x) \longrightarrow P(a) \wedge P(b))$$

then there would be no difficulty. But then a similar theorem such as

Ex. A7. $\forall z(Q(z) \longrightarrow P(z)) \longrightarrow \exists x[(P(x) \longrightarrow P(a)) \wedge (Q(x) \longrightarrow P(b))]$

would be very difficult to convert without eliminating the implication symbol "$\longrightarrow$".

There are two possible ways to cope with this type of difficulty. First we could eliminate all of the implication symbols (using $(\sim A \vee B)$ for $(A \longrightarrow B)$) from the given theorem, and work from there. But this would change the basic nature of our system, and we do not wish to do it for reasons which we give later.

Secondly, we could require that the program make "multiple copies" of existentially quantified disjuncts in the conclusion. For example

$$(H \longrightarrow \exists x \quad P(x))$$

would be copied as

$$(H \longrightarrow P(x) \vee P(x')) \ .$$

(Similarly, universally quantified conjunctions in the hypothesis

$$(\forall x \quad P(x) \longrightarrow C)$$

should be copied as

$$(P(x) \wedge P(x') \longrightarrow C) \quad ,$$

but this is already done by Rule I 4.2). (More generally, we would copy existentially quantified expressions in any "positive" position (see Appendix 1) of the theorem, and any universally quantified expression in a "negative" position.

Thus in Ex. A6, after copying, we get

(1) $[(P(x) \longrightarrow P(a)) \wedge (P(x) \longrightarrow P(b))] \vee [(P(x') \longrightarrow P(a)) \wedge (P(x') \quad P(b))]$

$\sim [(P(x') \longrightarrow P(a)) \wedge (P(x') \longrightarrow P(b))] \Rightarrow [(P(x) \longrightarrow P(a)) \wedge (P(x) \longrightarrow P(b))]$

H 4.2

$(P(x') \wedge \sim P(a)) \vee (P(x') \wedge \sim P(b)) \Rightarrow [(P(x) \longrightarrow P(a)) \wedge (P(x) \longrightarrow P(b))]$

(1 1) $(P(x') \wedge \sim P(a) \Rightarrow [(P(x) \longrightarrow P(a)) \wedge (P(x) \longrightarrow P(b))]$ I 3

(1 1 1) $(P(x') \wedge \sim P(a) \Rightarrow (P(x) \longrightarrow P(a)))$ I 4

$(P(x) \wedge P(a') \wedge \sim P(a) \Rightarrow P(a))$ a/x I 7, H 2

(1 1 2) $(P(x') \wedge \sim P(a) \Rightarrow (P(a) \longrightarrow P(b)))$ b/x' I 7, H 2

(1 2) $(P(b) \wedge \sim P(b) \Rightarrow [(P(a) \longrightarrow P(a)) \wedge (P(a) \longrightarrow P(b))]$

(1 2 1) $(P(b) \wedge \sim P(b) \Rightarrow (P(a) \longrightarrow P(a)))$ "T" I 4, 7, H 2

(1 2 2) $(P(b) \wedge \sim P(b) \Rightarrow (P(a) \longrightarrow P(b)))$ "T" I 7, H 2

So (a/x, b/x') is returned for (1).

A similar attack will succeed for Ex. A7. Of course, we may sometime need more copies than two, and in fact, a procedure would need to be established whereby copies are continually generated, at intervals, until the theorem is proved. Unfortunately this leads to an infinite regression on most non-theorems.

To show what copying does to a non-theorem we try

Ex. A8. $Q(a) \longrightarrow \exists x[(P(x) \longrightarrow P(a)) \wedge (P(x) \longrightarrow P(b)) \wedge Q(x)]$

NOT A THEOREM.

This example also needs copying

(1) $(Q(a) \longrightarrow [(P(x) \longrightarrow P(a)) \wedge (P(x) \longrightarrow P(b)) \wedge Q(x)]$

$\vee [(P(x') \longrightarrow P(a)) \wedge (P(x') \longrightarrow P(b)) \wedge Q(x')])$

$Q(a) \wedge \sim [(P(x') \longrightarrow \cdots] \Rightarrow [(P(x) \longrightarrow P(a)) \cdots]$

$[(Q(a) \wedge P(x') \wedge \sim P(a)) \vee (Q(a) \wedge P(x') \wedge \sim P(b)) \vee (Q(a) \wedge \sim Q(x'))$

$\Rightarrow [\overbrace{(P(x) \longrightarrow P(a)) \wedge (P(x) \longrightarrow P(b)) \wedge Q(x)}^{C}]$

(1 1) $(Q(a) \wedge P(x') \wedge \sim P(a)) \Rightarrow$ C

(1 1 1) $(Q(a) \wedge P(x') \wedge \sim P(a) \Rightarrow (P(x) \longrightarrow P(a))$ a/x

(1 1 2) $(Q(a) \wedge P(x') \wedge \sim P(a) \Rightarrow (P(a) \longrightarrow P(b)) \wedge Q(a))$

(1 1 2 1) $(Q(a) \wedge P(x') \wedge \sim P(a) \Rightarrow (P(a) \longrightarrow P(b)))$ b/x'

(1 1 2 2) $(Q(a) \wedge P(b) \wedge \sim P(a) \Rightarrow Q(a))$ "T"

Returns a/x, b/x' for (1 1)

(1 2) $[(Q(a) \wedge P(x') \wedge \sim P(b)) \wedge (Q(a) \wedge \sim Q(x')](b/x') \Rightarrow C(a/x)$

(1 2 1) $(Q(a) \wedge P(b) \wedge \sim P(b) \Rightarrow C(a/x))$

(1 2 1 1) $(Q(a) \wedge P(b) \wedge \sim P(b) \Rightarrow (P(a) \longrightarrow P(a)))$ "T"

(1 2 1 2) $(Q(a) \wedge P(b) \wedge \sim P(b) \Rightarrow (P(a) \longrightarrow P(b)) \wedge Q(a))$

(1 2 1 2 1) $(Q(a) \wedge P(b) \wedge \sim P(b) \Rightarrow (P(a) \longrightarrow P(b)))$ "T"

(1 2 1 2 2) $(Q(a) \wedge P(b) \wedge \sim P(b) \Rightarrow Q(a))$ "T"

(1 2 2) $(Q(a) \wedge \sim Q(b) \Rightarrow C(a/x))$

(1 2 2 1) $(Q(a) \wedge \sim Q(b) \Rightarrow (P(a) \longrightarrow P(a))$ "T"

(1 2 2 2) $(Q(a) \wedge \sim Q(b) \Rightarrow (P(a) \longrightarrow P(b)) \wedge Q(a))$

(1 2 2 2 1) $(Q(a) \wedge \sim Q(b) \Rightarrow (P(a) \longrightarrow P(b)))$
$(Q(a) \wedge P(a) \Rightarrow P(b) \vee Q(b))$ NIL

So NIL is returned for (1 2). It should also return NIL for (1), since it is false, but it would copy again instead and never return.

Clearly an additional hypothesis $Q(b)$ would make (1) valid in which case "T" would have been returned for (1 2 2 2 1), and (a/x, b/x') returned for (1).

<u>Comment</u>.

We object to the inclusion of "copying" rules and rules to permute hypotheses and conclusions to prevent "trapping" for several reasons.

These rules are not needed on a very large percentage of theorems, and yet they greatly increase the computing time in nearly all cases, sometimes by a large order of magnitude. If we ever expect to prove really difficult theorems in mathematics we must not strangle the mechanism that does it by making sure that it handles every case. Rather we believe (in the spirit of information theory) that it should be allowed to fail on a few cases so that it can succeed on a number of others, especially the hard ones.

The difficulties we point out here are faced by all other proving systems. Resolution systems pay the price by continuing all proofs of the theorem (allowed by the particular restriction on resolution). Gentzen type systems pay it through copying and search. They both remove the implication symbol and work with the result which we feel is very unnatural.

The implication symbol "$\longrightarrow$" or its equivalent plays a crucial role in mathematics. Much of Mathematics consists of stating and proving theorems of the form

$$(H_1 \wedge H_2 \wedge \ldots \wedge H_n \longrightarrow C_1 \wedge C_2 \wedge \ldots \wedge C_m) \quad .$$

Human proof technique, developed over a period of a few thousand years, center around using one or more of the H's to imply one of the C's. We have wanted to keep this same spirit in our prover so that we can easily use some of the powerful heuristics developed by mathematicians, so we can best interact with the prover on a man-machine basis.

Appendix 3

Some Proofs of Soundness

In Section 2 we stated that a call to IMPLY(H,C) would return NIL or a substitution θ such that

(1) $$(H\theta \rightarrow C\theta)$$

is valid. Indeed this is the case if θ has no conflicting entries (entries of the form a/x and b/x, $a \neq b$, or $g(y)/x$ and $f(x)/y$). We wish now to describe a slight change that could be made to IMPLY which would allow it to handle such conflicting entries, and which would permit us to prove a soundness result of the form (1). In doing so we will generalize the notion of a substitution, θ, allowing symbolic disjunctions

$$\theta = (\theta_1 \vee \theta_2)$$

to be returned from IMPLY. In this new setting the formula

(2) $$(H \rightarrow C)\theta$$

will be shown to be valid.

Before considering these generalized substitutions let us firm up our position for ordinary substitutions. Conflicts (of the form mentioned above) are <u>not</u> introduced when two formulas are unified. Indeed the classical unification algorithm prevents that. But conflicts may be introduced at AND-SPLIT's like Rule I4, where a goal of the form

(3) $$(H \Rightarrow A \wedge B)$$

is being proved. If θ is returned for $(H \Rightarrow A)$, and λ is returned for

$(H \Rightarrow B\theta)$, then Rule I4.4 returns $\sigma = \theta \circ \lambda$ for (3). (It would be easy to prevent any such conflict by requiring the subgoal $(H\theta \to B\theta)$ instead of $(H \to B\theta)$, but this would greatly weaken the prover, preventing the proof of such simple theorems as Example A4, Appendix 2.) If a conflict is introduced into σ it in no way implies the invalidity of formula (3). (We confirm this in Theorem 1 below.) It is just that such a σ with a conflict cannot be substituted (in the usual way) into other formulas. We overcome this difficulty by generalized substitutions.

Theorem 1. If θ and λ are ordinary substitutions, each without conflict, then

(4) $$(H \to A)\theta \wedge (H \to B\theta)\lambda \to (H \to A \wedge B) .$$

Proof. What are we trying to prove here? We must show that there are ordinary substitutions $\sigma_1, \sigma_2, \ldots, \sigma_n$ for which

(5) $$(H \to A \wedge B)\sigma_1 \vee \ldots \vee (H \to A \wedge B)\sigma_n$$

is a valid consequence of the hypothesis

(6) $$(H\theta \to A\theta) \wedge (H\lambda \to B\theta\lambda).$$

We will show (5) for $\sigma_1 = \theta$, $\sigma_2 = \lambda$.

The proof is by contradiction. Suppose that

(7) $$(H \to A \wedge B)\theta \vee (H \to A \wedge B)\lambda$$

is false. Then we have

(8) $$H\theta \wedge \sim(A\theta \wedge B\theta) \wedge H\lambda \wedge \sim(A\lambda \wedge B\theta) ,$$

from which we infer by (6), that

$$\begin{aligned}&(A\theta \wedge B\theta\lambda) \wedge \sim(A\theta \wedge B\theta)\\ &\quad = \sim[A\theta \wedge B\theta \longrightarrow A\theta \wedge B\theta\lambda]\\ &\quad = \sim(\text{TRUE}),\end{aligned}$$

a contradiction. Thus our assumption (8) is false and (7) is True. Q.E.D.

It should be noted that in our new notation below, the substitution $(\theta \vee \lambda)$ is returned for (3).

When we do have a conflict, according to Footnote 4, Section 2, we must consider two cases separately and combine the results from these into a resultant substitution.

We now propose instead of this another way of handling conflicting entries. The soundness of this new method is easier to prove, and its soundness implies the soundness of our present system.

Generalized Substitutions.

Definition. θ is a generalized substitution if

(i) θ is an ordinary substitution, or

(ii) θ has the form

$$(\theta_1 \vee \theta_2) \quad \text{or} \quad (\theta_1 \wedge \theta_2)$$

where θ_1 and θ_2 are generalized substitutions.

Some examples are,

$$\theta_1, \quad \theta_1 \vee \theta_2, \quad ((\theta_1 \vee \theta_2) \wedge \theta_3)\ ,$$

where the θ_i are ordinary substitutions.

Definition. If θ is a generalized substitution, then we define θ' by

(i) $\theta' = \theta$ if θ is an ordinary substitution

(ii) $(\theta_1 \vee \theta_2)' = (\theta_1' \wedge \theta_2')$,

(iii) $(\theta_1 \wedge \theta_2)' = (\theta_1' \vee \theta_2')$.

(This definition is for this Appendix only).

Definition. A generalized substitution is said to be a pure disjunction (conjunction) if it contains no $\wedge$ symbols ($\vee$ symbols).

Notice that this definition allows ordinary substitutions to be called pure disjunctions (and pure conjunctions).

Definition. If A is a formula and θ is a generalized substitution, then

$$A\theta$$

is the formula gotten by applying θ from left to right, i.e.,

(i) $A\theta$ is the usual result if θ is an ordinary substitution,

(ii) $A(\theta_1 \vee \theta_2) = A\theta_1 \vee A\theta_2$

(iii) $A(\theta_1 \wedge \theta_2) = A\theta_1 \wedge A\theta_2$.

Definition. For ordinary substitutions θ and λ, the iteration $\theta\lambda$ is defined to be the symbolic disjunction

$$\theta \vee \lambda$$

if θ and λ have conflicting entries (i.e., there are entries a/x in θ and b/x in λ, with $a \neq b$, or $g(y)/x$ and $f(x)/y$), and $\theta \circ \lambda$ otherwise.

Whenever we have an iterated generalized substitution such as

$$\theta\lambda$$

we can convert it into a generalized substitution by applying λ to θ. For example, if θ_i and λ_i are ordinary substitutions then

$$\begin{aligned} &(\theta_1 \wedge \theta_2)(\lambda_1 \vee \lambda_2) \\ &= (\theta_1\lambda_1 \wedge \theta_2\lambda_1) \vee (\theta_1\lambda_2 \wedge \theta_2\lambda_2) \\ &= (\theta_1\lambda_1 \vee \theta_1\lambda_2) \wedge (\theta_1\lambda_1 \vee \theta_2\lambda_2) \\ &\quad (\theta_2\lambda_1 \vee \theta_1\lambda_2) \wedge (\theta_2\lambda_1 \vee \theta_2\lambda_2) \ , \end{aligned}$$

where each of the terms $\theta_i\lambda_j$ is converted into $\theta_i \circ \lambda_j$ or $\theta_i \vee \lambda_j$.

Properties of generalized substitutions.

Lemma 1. If Φ and λ are generalized substitutions, λ is a pure disjunction, and A and B are formulas then λ' is a pure conjunction and

.1 $(\Phi')' = \Phi$

.2 $\sim(A\Phi) = \sim B\Phi'$

.3 $(A \vee B)\lambda = A\lambda \vee B\lambda$

.4 $(A \wedge B)\lambda' = A\lambda' \wedge B\lambda'$

.5 $(A \rightarrow B)\lambda = (A\lambda' \rightarrow B\lambda)$

Proof. .1 and .2 follow directly from the definition of Φ' and the properties of $\sim$. .3 and .4 follow from the associativity of $\vee$ and of $\wedge$. Then .5 follows from .3, .2, and .1, as follows

$$\begin{aligned} (A \rightarrow B)\lambda &= (\sim A \vee B)\lambda \\ &= (\sim A\lambda \vee B\lambda) \\ &= (\sim(A\lambda') \vee B\lambda) \\ &= (A\lambda' \rightarrow B\lambda) \ . \end{aligned}$$

Generalized substitutions in IMPLY and HOA.

The change we propose in the program applies only at AND-SPLITS (Rules 3 and 4 of IMPLY, Rules 7 and 7E of HOA). Two changes are required in proving an AND-SPLIT of the form

$$(H \Rightarrow A \wedge B) .$$

(1) We must state how the substitution θ which is returned for the first subgoal

$$(H \Rightarrow A)$$

is applied to B, before calling IMPLY again on the second subgoal.

(2) And we must state how we combine θ with the substitution λ returned from the second subgoal.

Table A-I gives these changes for IMPLY Rule 4. A similar change is needed for IMPLY Rule 3, and HOA Rules 7 and 7E.

Table A-I

	IF	ACTION	RETURN
4.	$C \equiv A \wedge B$	Put $\theta := \mathrm{IMPLY}(H,A)$	
4.1	$\theta \equiv \mathrm{NIL}$		NIL
4.2	$\theta \not\equiv \mathrm{NIL}$	Put $\lambda := \mathrm{IMPLY}(H,B\theta')$ where $\theta' = \theta$, if θ is an ordinary substitution, $(\theta_1 \vee \theta_2)' = \theta_1' \wedge \theta_2'$, and $(\theta_1 \wedge \theta_2)' = (\theta_1' \vee \theta_2')$.	
4.3	$\lambda \equiv \mathrm{NIL}$		NIL
4.4	$\lambda \not\equiv \mathrm{NIL}$		$\mathrm{COMBINE}(\theta,\lambda)$

COMBINE(θ,λ)

IF	ACTION	RETURN
$\lambda \equiv (\lambda_1 \vee \lambda_2)$		$(\mathrm{COMBINE}(\theta,\lambda_1), \mathrm{COMBINE}(\theta,\lambda_2))$
$\theta \equiv (\theta_1 \vee \theta_2)$		$(\mathrm{COMBINE}(\theta_1,\lambda), \mathrm{COMBINE}(\theta_2,\lambda))$
θ and λ have a conflict		$(\theta \vee \lambda)$
ELSE		$\theta \circ \lambda$

Notice that if θ and λ are pure disjunctions (which may be ordinary substitutions) then so also is COMBINE(θ,λ).

It should be noted that if θ and λ are ordinary substitutions which do not have conflicting entries then $\theta \circ \lambda$ is returned as before. If they do have conflicting entries, then $(\theta \vee \lambda)$, the symbolic disjunction of the two, is returned. If θ (or λ) is already such a disjunction

$$\theta = (\theta_1 \vee \theta_2) ,$$

then λ is combined to θ_1 and θ_2 separately. If λ has no conflict with θ_1 and θ_2 then the result is

$$(\theta_1 \circ \lambda \vee \theta_2 \circ \lambda) .$$

If λ has a conflict with θ_1 the result could be

$$((\theta_1 \vee \lambda) \vee \theta_2 \circ \lambda) ,$$

etc. In this way a pure disjunction is always returned.

The reader should note that by Rule A-I 4.2, when proving

$$(H \Rightarrow A \wedge B)$$

if $(\theta_1 \vee \theta_2)$ is returned for

$$(H \Rightarrow A)$$

then the second subgoal is

$$(H \Rightarrow B(\theta_1 \wedge \theta_2)) .$$

This requires <u>both</u> $B\theta_1$ and $B\theta_2$ to be proved, and is consistent with our Footnote 4, in Section 2, which states that if two conflicting values are returned from the first subgoal, then two cases must be handled.

We could have omitted $\theta \circ \lambda$ altogether even when θ and λ have no conflicts, and always use $\theta \vee \lambda$, but this would be less efficient for most proofs.

Soundness.

We are now in a position to prove the soundness of our extended system. We will do so only for Rule I4. The proof for the other rules is similar.

If we are proving

$$(H \Rightarrow A \wedge B) ,$$

and θ is returned for

$$(H \Rightarrow A)$$

and λ is returned for

$$(H \Rightarrow B\theta')$$

then it will return COMBINE(θ,λ) for

$$(H \Rightarrow A \wedge B) .$$

We show that this is a valid way of proceeding by proving Theorem 4 below.

Lemma 2. If λ is a pure disjunction then

$$(D \rightarrow D\lambda) .$$

Proof. The proof is by induction on the structure of λ.

Case 1. If λ is a ordinary substitution then $D\lambda$ is an instance of D and the result is immediate.

Case 2. (Induction step). If $\lambda = \lambda_1 \vee \lambda_2$, and we assume the induction hypothesis

$$(D \to D\lambda_1)$$

and

$$(D \to D\lambda_2)$$

then we have

$$D \to D\lambda_1 \wedge D\lambda_2 = D(\lambda_1 \vee \lambda_2) = D\lambda \ ,$$

as required.

Lemma 3. If θ is a pure disjunction, then

$$(D\theta \wedge (D \to E)\theta' \to E\theta) \ .$$

Proof. The proof is by induction on the structure of θ.

Case 1. If θ is an ordinary substitution then $\theta' = \theta$, and the result follows by modus ponens.

Case 2. (Induction step). If $\theta = \theta_1 \vee \theta_2$, and we assume the induction hypothesis,

$$D\theta_1 \wedge (D \to E)\theta_1' \to E\theta_1 \ ,$$

and

$$D\theta_2 \wedge (D \to E)\theta_2' \to E\theta_2 \ ,$$

then we have

$$\begin{aligned} & D\theta \wedge (D \to E)\theta' \\ = \ & D(\theta_1 \vee \theta_2) \wedge (D \to E)(\theta_1' \wedge \theta_2') \\ = \ & (D\theta_1 \vee D\theta_2) \wedge (D \to E)\theta_1' \wedge (D \to E)\theta_2' \\ \longrightarrow \ & E\theta_1 \vee E\theta_2 \\ = \ & E\theta \end{aligned}$$

as required. Q.E.D.

Comment. Lemma 2 is not surprising, but Lemma 3 is a curious form of modus ponens, which seems false at first glance. One should not make the mistake of putting $(D \to E)\theta'$ equal to $(D\theta \to E\theta')$. But even if he did the result would still not follow by modus ponens. Lemma 3 is crucial in the proof of Theorem 2 below.

Theorem 2. If θ and λ are pure disjunctions, then

(1) $(H \to A)\theta$

(2) $\wedge\ (H \to B\theta')\lambda$

(3) $\longrightarrow (H \to A \wedge B)(\theta \wedge \lambda)$.

Proof. The proof is by contradiction. Suppose that the hypotheses (1) and (2) hold and the conclusion (3) is false.

Thus using Lemma 1.5 we have

(4) $(H\theta' \to A\theta)$

and

(5) $(H\lambda' \to B\theta'\lambda)$,

and

(6) $\sim[(H \to A \wedge B)(\theta \wedge \lambda)]$

which, using Lemmas 1.5 and 1.2, is equivalent to

$\sim[H(\theta \vee \lambda)' \longrightarrow (A \wedge B)(\theta \vee \lambda)]$

$= \sim[H(\theta' \wedge \lambda') \longrightarrow (A \wedge B)(\theta \vee \lambda)]$

$= \quad H\theta' \wedge H\lambda' \wedge \sim(A \wedge B)(\theta \vee \lambda)'$

$= \quad H\theta' \wedge H\lambda' \wedge (\sim A \vee \sim B)\theta' \wedge (\sim A \vee \sim B)\lambda'$

(7)$= \quad H\theta' \wedge H\lambda' \wedge (A \to \sim B)\theta' \wedge (A \to \sim B)\lambda'$.

It follows from (7) and (4) and (5) (using Modus ponens) that

(8) $A\theta$,

(9) $B\theta'\lambda$,

and from (8), (7) and Lemma 3 that

(10) $\sim B\theta = \sim[B\theta']$

But (10) is in contradiction of (9) by Lemma 2, and hence our assumption (6) is false and the theorem is true. Q.E.D.

Theorem 2 shows that we can dispense altogether wish $\theta \circ \lambda$ using always instead ~~of~~ $\theta \vee \lambda$. However, as mentioned earlier, this would be much less efficient when θ and λ have no conflict.

A weaker version of Theorem 2 would have sufficed in the proof of Theorem 4.

Theorem 3. If θ and λ are ordinary substitutions with no mutual conflicts and H is a formula, then

$$(H\lambda \rightarrow H\theta\lambda) .$$

Proof. We will sketch the proof only for the case where H has only two variables x and y. We write $H = G(x,y)$.

Since θ and λ have no conflict within themselves then they can take only the following possible forms:

θ: $a/x.$ b/y, $g(y)/x$, $f(x)/y$,
$a/x\ b/y$, $a/x\ f(x)/y$, $g(y)/x\ b/y$,

λ: c/x, d/y, $j(y)/x$, $h(x)/y$,
$c/x\ d/y$, $c/x\ h(x)/y$, $j(y)/x\ d/y$,

where a,b,c,d are constants.

Since θ and λ have no mutual conflicts, many combinations such as:

$$\theta = a/x \qquad \lambda = c/x \quad ,$$
$$\theta = g(y)/x \qquad \lambda = f(x)/y \ ,$$

are conflicting and need not be considered. Others such as

$$\theta = a/x \qquad \lambda = h(x)/y \ ,$$
$$\theta = g(y)/x \qquad \lambda = d/y \quad ,$$

are not conflicting, and for these cases the desired conclusion holds. For example if $\theta = a/x$, $\lambda = h(x)/y$, then

$$H\theta\lambda = G(a,h(x)), \quad H\lambda = G(x,h(x)) \ ,$$

and

$$(H\lambda \rightarrow H\theta\lambda)$$

becomes

$$G(x,h(x)) \rightarrow G(a,h(x))$$

which is valid (put a for x).

Theorem 4. If θ and λ are pure disjunctions, then

$$\begin{aligned} & (H \longrightarrow A)\theta \\ \wedge\ & (H \longrightarrow B\theta')\lambda \\ \longrightarrow\ & (H \longrightarrow A \wedge B)\ \mathrm{COMBINE}(\theta,\lambda) \end{aligned}$$

Proof. We will abbreviate $\mathrm{COMBINE}(\theta,\lambda)$ as $C(\theta,\lambda)$ in this proof. The proof is by induction of the structures of λ and θ.

Case 1. λ is an ordinary substitution.

Case 1.1. θ is an ordinary substitution.

Case 1.1.1. λ and θ have no mutual conflict.

In this case $C(\theta,\lambda) = \theta \circ \lambda$, and the desired result follows from Theorem 3.

Case 1.1.2. λ and θ have a conflict.

In this case $C(\theta,\lambda) = \theta \vee \lambda$, and the desired conclusion follows as a special case of Theorem 2.

Case 1.2. $\theta = \theta_1 \vee \theta_2$.

In this case $C(\theta,\lambda) = C(\theta_1,\lambda) \vee C(\theta_2,\lambda)$, and the induction hypotheses are

(1) $$(H \to A)\theta_1 \wedge (H \to B\theta_1')\lambda \to (H \to A \wedge B)C(\theta_1,\lambda) ,$$

(2) $$(H \to A)\theta_2 \wedge (H \to B\theta_2')\lambda \to (H \to A \wedge B)C(\theta_2,\lambda) .$$

Thus

$$(H \to A)\theta \wedge (H \to B\theta')\lambda$$
$$= [(H \to A)\theta_1 \vee (H \to A)\theta_2] \wedge (H \to B\theta_1' \wedge B\theta_2')\lambda$$
$$= [(H \to A)\theta_1 \vee (H \to A)\theta_2] \wedge (H \to B\theta_1')\lambda \wedge (H \to B\theta_2')\lambda$$

since λ is an ordinary substitution

$$\longrightarrow (H \to A \wedge B)C(\theta_1,\lambda) \vee (H \to A \wedge B)C(\theta_2,\lambda)$$

by (1) and (2)

$$= (H \to A \wedge B)[C(\theta_1,\lambda) \vee C(\theta_2,\lambda)]$$
$$= (H \to A \wedge B)C(\theta,\lambda) .$$

Case 2. $\lambda = \lambda_1 \vee \lambda_2$.

In this case, $C(\theta,\lambda) = C(\theta,\lambda_1) \vee C(\theta,\lambda_2)$ and the induction hypotheses are

(3) $(H \rightarrow A)\theta \wedge (H \rightarrow B\theta')\lambda_1 \longrightarrow (H \rightarrow A \wedge B)C(\theta,\lambda_1)$,

(4) $(H \rightarrow A)\theta \wedge (H \rightarrow B\theta')\lambda_2 \longrightarrow (h \rightarrow A \wedge B)C(\theta,\lambda_2)$.

Thus

$$(H \rightarrow A)\theta \wedge (H \rightarrow B\theta')\lambda$$
$$= (H \rightarrow A)\theta \wedge [(H \rightarrow B\theta')\lambda_1 \vee (H \rightarrow B\theta')\lambda_2]$$
$$\longrightarrow (H \rightarrow A \wedge B)C(\theta,\lambda_1) \vee (H \rightarrow A \wedge B)C(\theta,\lambda_2)$$

by (3) and (4)

$$= (H \rightarrow A \wedge B)[C(\theta,\lambda_1) \vee C(\theta,\lambda_2)]$$
$$= (H \rightarrow A \wedge B)C(\theta,\lambda) \ .$$ Q.E.D.

Example.

$$(Q(a) \wedge Q(b) \longrightarrow \exists x[(P(x) \longrightarrow P(a) \wedge P(b)) \wedge Q(x)])$$

(1) $(Q(a) \wedge Q(b) \Rightarrow (P(x) \longrightarrow P(a) \wedge P(b)) \wedge Q(x))$

(1 1) $(Q(a) \wedge Q(b) \Rightarrow (P(x) \longrightarrow P(a) \wedge P(b)))$

(1 1 1) $(Q(a) \wedge Q(b) \wedge P(x) \Rightarrow P(a))$ a/x

(1 1 2) $(Q(a) \wedge Q(b) \wedge P(x) \Rightarrow P(b))$ b/x

Returns $(a/x \vee b/x)$ for (1 1).

(1 2) $(Q(a) \wedge Q(b) \Rightarrow Q(x)(a/x \vee b/x)')$

$(Q(a) \wedge Q(b) \Rightarrow Q(a) \wedge Q(b))$ TRUE

Returns $(a/x \vee b/x)$ for (1).

References

1. W.W. Bledsoe and P. Bruell. A man-machine theorem-proving system. In Adv. Papers 3rd Int. Joint Conf. Artif. Intell., 1973, pp. 55-65; also Artif. Intell., vol. 5, no. 1, pp. 51-72, Spring 1974.

2. W.W. Bledsoe. Splitting and reduction heuristics in automatic theorem proving. Artificial Intelligence 2(1971), 55-77.

3. W.W. Bledsoe, R.S. Boyer, and W.H. Henneman. Computer proofs of limit theorems. Artif. Intell., vol. 3, no. 1, pp. 27-60, Spring 1972.

4. P. Bruell. A description of the functions of the man-machine topology theorem prover. The Univ. of Texas at Austin. Math Dept. Memo ATP8, 1973.

5. W.W. Bledsoe. The sup-inf method in Presburger arithmetic. Dept. Math., Univ. Texas, Austin, Memo ATP 18. Dec. 1974. Essentially the same as: A new method for proving certain Presburger formulas. Fourth IJCAI, Tblisi, USSR, Sept. 3-8, 1975.

6. D.I. Good, R.L. London, and W.W. Bledsoe. An interactive verification system. Proceedings of the 1975 International Conf. on Reliable Software, Los Angeles, April 1975, pp. 482-492, and _IEEE Trans. on Software Engineering_ 1(1975), pp. 59-67.

7. C. Chang and R.C. Lee. _Symbolic logic and mechanical theorem proving_. Academic Press, 1973.

8. Donald Loveland. Forthcoming book on Mechanical theorem proving in first order logic. (Duke University)

9. J. Allen and D. Luckham. An interactive theorem-proving program. Machine Intelligence 5(1970), 321-336.

10. J.R. Guard, F.C. Oglesby, J.H. Bennett and L.G. Settle. Semi-automated mathematics. J. ACM 16(1969), 49-62.

11. G.P. Huet. Experiments with an interactive prover for logic with equality. Rept. 1106, Jennings Computing Center, Case Western Reserve University.

12. A. Newell, J.C. Shaw and H.A. Simon. Empirical explorations of the logic theory machine: a case study in heuristics. RAND Corp. Memo P-951, Feb. 28, 1957. Proc. Western Joint Computer Conf. 1956, 218-239.

    A. Newell, J.C. Shaw and H.A. Simon. Report on a general problem-solving program. RAND Corp. Memo P-1584, Dec. 30, 1958.

13. H. Gelerntner. Realization of a geometry theorem-proving machine. Proc. Int'l. Conf. Information Processing, 1959, Paris UNESCO House, 273-282.

14. G. Gentzen. Untersuchungen uber das logische Schließen I. Mathemat. Zeitschrift 39, 176-210, (1935).

15. Arthur J. Nevins. A human oriented logic for automatic theorem proving. MIT-AI-Lab Memo 268, Oct. 1972. JACM 21(1974), 606-621.

16. Arthus J. Nevins. A relaxation approach to splitting in an automatic theorem prover. MIT-AI-Lab. Memo 302, Jan. 1974. To appear in the AI Jour.

17. Arthur J. Nevins. Plane geometry theorem proving using forward chaining. MIT-AI-Lab Memo 303, Jan. 1974.

18. Raymond Reiter. A semantically guided deductive system for automatic theorem proving. Proc. Third Int'l. Joint Conf. on Art. Intel., 1973, 41-46.

19. George W. Ernst, The utility of independent subgoals in theorem proving. Information and Control, April 1971. A definition-driven theorem prover. Int'l. Joint Conf. on Artificial Intelligence, Stanford, Calif., August 1973, pp. 51-55.

20. W. Bibel and J. Schreiber. Proof search in a Gentzen-like system of first-order logic. Bericht Nr. 7412, Technische Universitat, 1974.

21. Carl Hewitt. Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot. Ph.D. Thesis (June 1971). AI-TR-258 MIT-AI-Lab. April 1972.

22. D.V. McDermott and Gerald J. Sussman. The CONNIVER reference manual. AI Memo 259. MIT-AI-Lab. (May 1972) (Revised July 1973).

23. Hao Wang. Toward mechanical mathematics, IBM J. Res. Dev. 4, 2-22.

24. J.R. Rulifson, J.A. Derksen and R.J. Waldinger. "QA4: a procedural calculus for intuitive reasoning. Standford Res. Inst. Artif. Intell. Center, Standford, Calif., Tech. Note 13, Nov. 1972.

25. D. Prawitz. An imporved proof procedure. Theoria 25, 102-139 (1960).

26. Nils Nilsson. Artificial Intelligence Review of automatic theorem proving. IFIP, Stockholm, Sweden, 1974.

27. W.W. Bledsoe and Mabry Tyson. Typing and proof by cases in program verification. Univ. of Texas Math. Dept. Memo ATP 15, May 1975.

29. R.S. Boyer and J.S. Moore. Proving theorems about Lisp functions. J. Ass. Comput. Mach., vol. 22, pp. 129-144, Jan. 1975.

30. Dallas S. Lankford. Complete sets of reductions for computational logic. Univ. of Texas Math. Dept. Memo ATP-21, Jan. 1975.

31. D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. Computational Problems in Abstract Algebra. J. Leech, Ed., Pergamon Press, 1970, 263-297.

32. J.R. Slagle. Automated theorem-proving for theories with simplifiers, commutativity and associativity. J. ACM, 21(1974), 622-642.

33. N. Suzuki. Verifying programs by algebraic and logical reduction. Proc. Int'l. Conf. on Reliable Software, 1975, 473-481.

34. Mabry Tyson. An algebraic simplifier. Univ. of Texas Math. Dept. Memo ATP-26 (to appear).

35. A.C. Hearn. Reduce 2: A system and language for algebraic manipulation. In Proc. Ass. Comput. Mach., 2nd Symp. Symbolic and Algebraic Manipulation, 1971, pp. 128-133; also Reduce 2 User's Manual, 2nd ed., Univ. Utah, Salt Lake City, UCP-19, 1974.

36. L. Siklossy, A. Rich and V. Marinov. Breadth-first search: some surprising results. A.I. Jour. 4(1973), 1-28.

37. A. Bundy. Doing arithmetic with diagrams. In Adv. Papers 3rd Int. Joint Conf. Artif. Intell., 1973, pp. 130-138.

38. Michael Ballantyne and William Bennett. Graphing methods for topological proofs. Univ. of Texas at Austin, Math. Dept. Memo ATP-7, 1973.

39. Michael Ballantyne, Computer generation of counterexamples in topology. Univ. of Texas at Austin Math. Dept. Memo ATP-24, 1975.

40. J.L. Darlington. Automatic theorem proving with equality substitution and mathematical induction. Machine Intelligence, 3(1968), 113-127.

41. Jack Minker, D.H. Fishman and J.R. McSkimin. The Q^* algorithm -- a search strategy for a deductive question-answering system. A.I. Jour. 4(1973), 225-243.

42. D.H. Fishman. Experiments with a resolution-based deductive question-answering system and a proposed clause representation for parallel search. Ph.D. Thesis, Dept. of Comp. Sci., Univ. of Maryland, (1973).

43. R.J. Waldinger and K.N. Levitt. Reasoning about programs. Artif. Intel. 5(1974). 235-316.

44. J.C. King. A program verifier. Ph.D. dissertation, Carnegie-Mellon Univ., Pittsburgh, Pa., 1969.

45. Michael Ballantyne. Automatic proofs of limit theorems in analysis. Univ. of Texas at Austin Math. Dept. Memo ATP-23, 1975.

46. Donald W. Loveland and M.E. Stickel. A hole in goal trees: some guidance from resolution theory. Proc. third Int'l. Joint Conf. on Art. Intel., Stanford, 1973, 153-161.

47. Raymond Reiter. A paradigm for automated formal inference. To be presented at the IEEE theorem proving workshop, Argonne Nat'l. Lab., Ill., June 3-5, 1975.

48. S. Ju Maslov. Proof-search startegies for methods of the resolution type. Machine Intelligence 6(1971), 77-90.

49. Bernard Luya. Un systeme complet de deduction maturelle. Thesis, University of Paris VII, Jan. 1975.