Time Limit.

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).

Pretty-Print.

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

```
(CC{C: Closed C})
∧
(REF{C: Closed C}(F))
∧
(LF{C: Closed C})
```

to be printed, whereas TPC G causes

```
(CC G)
∧
(REF G (F))
∧
(LF G)
```

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

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

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

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

"(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

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

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

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

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

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

it will obtain the goal

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

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

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

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

$(\alpha \Rightarrow x_o \in \cup (\quad) \longrightarrow x_o \in A_o)$ 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) $(\alpha \Rightarrow A_o \subseteq \cup (\text{OPEN} \cap \text{Subset } A_o)$ 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 $(H \longrightarrow A)$ and $(H \longrightarrow B)$. Thus the machine might respond

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

PROVED..

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

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

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

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

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

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

$$\wedge$$

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

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 f(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 queries.

```
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 $(H \longrightarrow C_1 \wedge C_2)$, there are top-level sub-calls to IMPLY for $(H \longrightarrow C_1)$ and $(H \longrightarrow C_2)$. DETAIL stops at OPTIONS after $(H \longrightarrow C_1)$ is attempted and then after $(H \longrightarrow C_2)$ is attempted. CNT does not stop until after the attempt to prove $(H \longrightarrow C_1 \wedge C_2)$ is completed. A "B" call stops before $(H \longrightarrow C_1)$ 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 typed 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

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

are skolemized as

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

and

$$(\forall x\, P(x) \rightarrow C)$$

is skolemized as

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

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 $(A \rightarrow B)$ 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

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

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

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

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
	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 θ. 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|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

Ex. A3. $(P(a) \wedge Q(b) \wedge P(c) \wedge Q(c) \rightarrow \exists x(P(x) \wedge Q(x)))$

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.

Ex. A4. $(\forall x\ P(x) \longrightarrow P(a) \wedge P(b))$

(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.

Ex. A5. $\exists x(P(x) \longrightarrow P(a) \wedge P(b))$

(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.

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

(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

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

would be copied as

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

(Similarly, universally quantified conjunctions in the hypothesis

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

should be copied as

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

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).

Comment.

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

The theorem prover is designed to prove the validity of formulas in first order predicate calculus. This appendix will show that if the prover returns a substitution for some formula then that formula is valid.

When a closed formula, E, is given to the prover, it skolemizes (see Appendix 1) the formula into an open formula, S. If $v_1, v_2, \ldots, v_n$ are the free variables in S, then the original formula E is equivalent to

(1) $$\exists v_1 \exists v_2 \ldots \exists v_n \; S.$$

If some substitution θ exists such that $S\theta$ is ground (contains no free variables) and $S\theta$ is true then (1) is true. Likewise if there is some set of substitutions $\theta_1, \theta_2, \ldots, \theta_m$ such that

$$S\theta_1 \vee S\theta_2 \vee \ldots \vee S\theta_m$$

is both ground and true then (1) is true.

We will show that if the prover returns some non-NIL substitution θ then for any substitution ψ such that $S\theta\psi$ is ground then $S\theta\psi$ is true. From this it follows that (1) and the original input are true.

A formula, P, will be termed <u>ground-true</u> if for any substitution Ψ such that $P\Psi$ is ground, $P\Psi$ is true[1].

We will indicate that a formula P is ground-true by the notation

$$\forall\Psi P\Psi.$$

<u>Lemma 1</u>. Properties of Ground-Truth. If P and Q are any formulas and θ and λ are any substitutions then we know

1.1 $\forall\Psi P\Psi \leftrightarrow P$ when P is ground.

1.2 $\forall\Psi P\Psi \rightarrow \forall\Psi P\theta\Psi$.

1.3 $\forall\Psi P\Psi \rightarrow \forall\Psi(P\theta \wedge P\lambda)\Psi$. *

1.4 $\forall\Psi P\Psi \rightarrow \forall\Psi(P\theta \vee P\lambda)\Psi$.

1.5 $\forall\Psi(P \wedge Q)\Psi \leftrightarrow \forall\Psi P\Psi \wedge \forall\Psi Q\Psi$.

1.6 $(\forall\Psi P\Psi \vee \forall\Psi Q\Psi) \rightarrow \forall\Psi(P \vee Q)\Psi$.

1.7 $(\forall\Psi P\Psi \wedge \forall\Psi(P \rightarrow Q)\Psi) \rightarrow \forall\Psi Q\Psi$.

Remember that to show ground-truth we need only show that $P\Psi$ is true for all Ψ such that $P\Psi$ does not contain tree variables.

To show that the prover is sound we will show that the rules used in the prover are each sound. Induction will be used where the prover is recursive. Most of the rules used permit straight forward proofs. For instance, Rule 13 requires the proof that

(2) $\forall\Psi((A \rightarrow C) \wedge (B \rightarrow C))\theta\Psi \rightarrow \forall\Psi((A \vee B) \rightarrow C)\theta\Psi$.

[1] Equivalently, P is ground-true if and only if $\forall v_1 \forall v_2 \ldots \forall v_n$ P is true where $v_1, v_2, \ldots, v_n$ are the tree variables in P.

* '&' is the same as '$\wedge$' in the paper

The left hand side is the induction hypothesis, i.e., that IMPLY(NIL, (A -> C) & (B -> C)) returns a substitution θ such that ((A -> C) & (B -> C))θ is ground-true. The conclusion of (2) is what must be shown: that the returned substitution θ must be such that the original form, ((A v B) -> C), is ground-true after θ is applied. Lemma 1.7 and the fact that the two forms in (2) are equivalent proves (2).

The rules concerning AND-SPLIT, Forward Chaining and Back Chaining are more difficult to prove. We shall prove these by first proving something more general by describing some changes to the rules and proving the soundness of the modified system. In doing so we will generalize the notion of a substitution, θ, allowing symbolic disjunctions

$$\theta = (\theta_1 \text{ v } \theta_2)$$

to be returned from IMPLY.

Generalized Substitutions.

Definition. θ is a generalized substitution if

(i) θ is an ordinary substitution, or

(ii) θ has the form

$$(\theta_1 \text{ v } \theta_2) \text{ or } (\theta_1 \text{ \& } \theta_2)$$

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_1 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) $A(\theta_1 \wedge \theta_2) = A\theta_1 \wedge A\theta_2$.

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

$$(\theta_1 \ \& \ \theta_2)(\lambda_1 \ v \ \lambda_2)$$

$$= (\theta\lambda_{11} \ \& \ \theta\lambda_{21}) \ v \ (\theta\lambda_{12} \ \& \ \theta\lambda_{22})$$

$$= (\theta\lambda_{11} \ v \ \theta\lambda_{12}) \ \& \ (\theta\lambda_{11} \ v \ \theta\lambda_{22})$$

$$\& \ (\theta\lambda_{21} \ v \ \theta\lambda_{12}) \ \& \ (\theta\lambda_{21} \ v \ \theta\lambda_{22}) .$$

Properties of Generalized Substitutions.

Lemma 2. If θ and λ are generalized substitutions, λ is a pure disjunction, and A and B are formulas then λ' is a pure conjunction and

2.1 $(\theta')' = \theta$

2.2 $\sim(A\theta) = \sim B\theta'$

2.3 $(A \ v \ B)\lambda = A\lambda \ v \ B\lambda$

2.4 $(A \ \& \ B)\lambda' = A\lambda' \ \& \ B\lambda'$

2.5 $(A \rightarrow B)\lambda = (A\lambda' \rightarrow B\lambda)$

Proof. .1 and .2 follow directly from the definition of θ' and the properties of ~. .3 and .4 follow from the associativity of v and of &. 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 HUA.

The changes we propose in the program apply at those points at which two subgoals are combined to prove a goal. These are the rules concerning AND-SPLITs, forward chaining, and back chaining (rules 4, 7.1, 7.2 of IMPLY and 7, 7E of HUA). Two changes are required in proving an AND-SPLIT of the form

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

(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.

Tables A-I and A-II give these changes for IMPLY rules 4, 7.1, and 7.2 and HUA rules 7 and 7E.

TABLE A-1

IF	ACTION	RETURN
4. θ = A & B	Put θ := IMPLY(H,A)	
4.1 θ = NIL		NIL
4.2 $\theta \neq$ NIL	Put λ := IMPLY(H,Bθ')	
4.3 λ = NIL		NIL
4.4 $\lambda \neq$ NIL		$\theta\lambda$ v λ
7.1.1 (Forward chaining)		
A = (P -> Q)	Put θ := ANDS(H,P)	
7.1.2 θ = NIL	Go to 8	
7.1.3 $\theta \neq$ NIL	Put λ := IMPLY(H & A & Qθ, Bθ)	
7.1.4 λ = NIL	Go to 8	
7.1.5 $\lambda \neq$ NIL		$\theta\lambda$ v λ

TABLE A-11

	IF	ACTION	RETURN
7.	(Back-chaining)		
	B = (A -> D)	Put θ := ANDS(D,C)	
7.1	θ = NIL	Go to 7E	
7.2	θ ≠ NIL	Put λ := IMPLY(H,Aθ')	
7.3	λ = NIL		NIL
7.4	λ ≠ NIL		θλ v λ
7E.	B = (A -> A=B)	Put θ := NOA(A=B,C)	
7E.1	θ = NIL		NIL
7E.2	θ ≠ NIL	Put λ := IMPLY(H,Aθ')	
7E.3	λ = NIL		NIL
7E.4	λ ≠ NIL		θλ v λ

Notice that IMPLY and NOA always return pure disjunctions.

Soundness.

We are now in a position to prove the soundness of our extended system. We will do so only for Rule 14. Proofs for other rules are 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 $\theta\lambda \vee \lambda$ is returned for

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

Lemma 3. If θ and λ are pure disjunctions then

3.1 $(H \to C)\theta\lambda \leftrightarrow (H\theta'\lambda' \to C\theta\lambda)$

3.2 $\forall\overline{x}(H \to C)\theta\overline{x} \to \forall\overline{x}(H\theta'\lambda' \to C\theta\lambda)\overline{x}$

3.3 $\forall\overline{x}(H \to C)\theta\overline{x} \to \forall\overline{x}(H\theta'\lambda' \to C\theta\lambda')\overline{x}$

3.4 $\forall\overline{x}(H \to C)\theta\overline{x} \to \forall\overline{x}(H\theta'\lambda \to C\theta\lambda)\overline{x}$

Proof.

3.1 $(H \to C)\theta\lambda \iff (H\theta' \to C\theta)\lambda$

$\iff (H\theta'\lambda' \to C\theta\lambda)$

3.2 $\forall\mathfrak{x}(H \to C)\theta\mathfrak{x} \to \forall\mathfrak{x}(H \to C)\theta\lambda\mathfrak{x}$

$\to \forall\mathfrak{x}(H\theta'\lambda' \to C\theta\lambda)\mathfrak{x}$

3.3 Proof is by induction on the structure of λ.

Case 1. λ is ordinary.

$\lambda=\lambda'$ so .2 applies.

Case 2. $\lambda = \lambda_1 \vee \lambda_2$. We use the induction hypotheses:

$$\forall\mathfrak{x}(H \to C)\theta\mathfrak{x} \to \forall\mathfrak{x}(H\theta'\lambda_1' \to C\theta\lambda_1')\mathfrak{x} ,$$

$$\forall\mathfrak{x}(H \to C)\theta\mathfrak{x} \to \forall\mathfrak{x}(H\theta'\lambda_2' \to C\theta\lambda_2')\mathfrak{x} .$$

So we have

$$\forall\mathfrak{x}(H \to C)\theta\mathfrak{x} \to (\forall\mathfrak{x}(H\theta'\lambda_1' \to C\theta\lambda_1')\mathfrak{x} \,\&\, \forall\mathfrak{x}(H\theta'\lambda_2' \to C\theta\lambda_2')\mathfrak{x})$$

$$\to \forall\mathfrak{x}[(H\theta'\lambda_1' \to C\theta\lambda_1') \,\&\, (H\theta'\lambda_2' \to C\theta\lambda_2')]\mathfrak{x}$$

$$\to \forall\mathfrak{x}[(H\theta'\lambda_1' \,\&\, H\theta'\lambda_2') \to (C\theta\lambda_1' \,\&\, C\theta\lambda_2')]\mathfrak{x}$$

$$\to \forall\mathfrak{x}[H\theta'(\lambda_1' \,\&\, \lambda_2') \to C\theta(\lambda_1' \,\&\, \lambda_2')]\mathfrak{x}$$

$$\to \forall\mathfrak{x}(H\theta'\lambda' \to C\theta\lambda')\mathfrak{x}$$

3.4 Proof is by induction on the structure of λ.

Case 1. λ is ordinary.

$\lambda=\lambda'$ so .2 applies.

Case 2. $\lambda = \lambda_1 \vee \lambda_2$. We use the induction hypotheses:

$$\forall\Psi(H \to C)\theta\Psi \to \forall\Psi(H\theta'\lambda_1 \to C\theta\lambda_1)\Psi ,$$

$$\forall\Psi(H \to C)\theta\Psi \to \forall\Psi(h\theta'\lambda_2 \to C\theta\lambda_2)\Psi .$$

So we have

$$\forall\Psi(H \to C)\theta\Psi \to (\forall\Psi(H\theta'\lambda_1 \to C\theta\lambda_1)\Psi \;\&\; \forall\Psi(H\theta'\lambda_2 \to C\theta\lambda_2)\Psi)$$

$$\to \forall\Psi[(H\theta'\lambda_1 \to C\theta\lambda_1) \;\&\; (H\theta'\lambda_2 \to C\theta\lambda_2)]\Psi$$

$$\to \forall\Psi[(h\theta\lambda_1 \vee h\theta\lambda_2) \to (C\theta\lambda_1 \vee C\theta\lambda_2)]\Psi$$

$$\to \forall\Psi[h\theta(\lambda_1 \vee \lambda_2) \to C\theta(\lambda_1 \vee \lambda_2)]\Psi$$

$$\to \forall\Psi(H\theta\lambda \to C\theta\lambda)\Psi$$

Lemma 4. If θ and λ are pure disjunctive generalized substitutions then

$$\forall\Psi[(A\theta\lambda' \;\&\; B\theta'\lambda) \to (A \;\&\; B)\theta\lambda]\Psi$$

Proof. Proof is by induction on the sturcture of λ.

Case 1. λ is ordinary.

Since $\lambda=\lambda'$ we need to establish

(3) $$\forall\Psi[(A\theta\lambda \;\&\; B\theta'\lambda) \to (A \;\&\; B)\theta\lambda.]\Psi$$

This is shown by induction on the structure of θ.

Case 1.1. θ is ordinary.

$$(A\theta\lambda \;\&\; B\theta'\lambda) = (A\theta\lambda \;\&\; B\theta\lambda) = (A \;\&\; B)\theta\lambda.$$

Case 1.2. $\theta = \theta_1 \vee \theta_2$. We will use the induction hypotheses:

$$\forall\Psi[(A\theta_1\lambda \;\&\; B\theta'\lambda) \to (A \;\&\; B)\theta_1\lambda]\Psi \ .$$

$$\forall\Psi[(A\theta_2\lambda \;\&\; B\theta'\lambda) \to (A \;\&\; B)\theta_2\lambda]\Psi \ .$$

we need to show

$$((A\theta\lambda \;\&\; B\theta'\lambda) \to (A \;\&\; B)\theta\lambda)\Psi \tag{4}$$

for all Ψ's such that (4) is ground[2]. we will show it for an arbitrary Ψ satisfying that condition.

$$A\theta\lambda\Psi \;\&\; B\theta'\lambda\Psi \to A(\theta_1 \vee \theta_2)\lambda\Psi \;\&\; B(\theta_1' \;\&\; \theta_2')\lambda\Psi$$

$$\to (A\theta_1\lambda\Psi \vee A\theta_2\lambda\Psi) \;\&\; B\theta_1'\lambda\Psi \;\&\; B\theta_2'\lambda\Psi$$

$$\to (A\theta_1\lambda\Psi \;\&\; B\theta_1'\lambda\Psi \;\&\; B\theta_2'\lambda\Psi)$$

$$\vee\ (A\theta_2\lambda\Psi \;\&\; B\theta_1'\lambda\Psi \;\&\; B\theta_2'\lambda\Psi)$$

$$\to (A\theta_1\lambda\Psi \;\&\; B\theta_1'\lambda\Psi) \vee (A\theta_2\lambda\Psi \;\&\; B\theta_2'\lambda\Psi) \tag{5}$$

by induction hypotnesis ((5) is ground since (4) is)

$$\to (A \;\&\; B)\theta_1\lambda\Psi \vee (A \;\&\; B)\theta_2\lambda\Psi$$

$$\to (A \;\&\; B)(\theta_1 \vee \theta_2)\lambda\Psi$$

[2] The substitution Ψ can be presumed to be an ordinary substitution.

$$\rightarrow (A \,\&\, B)\theta\lambda\Psi$$

So (4) is shown for an arbitrary Ψ.

So (3) is established.

Case 2. $\lambda = \lambda_1 \vee \lambda_2$. We use the induction hypotheses:

$$\forall\Psi[(A\theta\lambda_1' \,\&\, B\theta'\lambda_1) \rightarrow (A \,\&\, B)\theta\lambda_1]\Psi ,$$

$$\forall\Psi[(A\theta\lambda_2' \,\&\, B\theta'\lambda_2) \rightarrow (A \,\&\, B)\theta\lambda_2]\Psi .$$

We need to show

(6) $$[(A\theta\lambda' \,\&\, B\theta'\lambda) \rightarrow (A \,\&\, B)\theta\lambda]\Psi$$

for all Ψ for which (6) is ground. We will show it for an arbitarary Ψ satisfying that condition.

$$A\theta\lambda'\Psi \,\&\, B\theta'\lambda\Psi \rightarrow A\theta(\lambda_1' \,\&\, \lambda_2')\Psi \,\&\, B\theta'(\lambda_1 \vee \lambda_2)\Psi$$

$$\rightarrow A\theta\lambda_1'\Psi \,\&\, A\theta\lambda_2'\Psi \,\&\, (B\theta'\lambda_1\Psi \vee B\theta'\lambda_2\Psi)$$

$$\rightarrow (A\theta\lambda_1'\Psi \,\&\, A\theta\lambda_2'\Psi \,\&\, B\theta'\lambda_1\Psi)$$

$$\vee (A\theta\lambda_1'\Psi \,\&\, A\theta\lambda_2'\Psi \,\&\, B\theta'\lambda_2\Psi)$$

(7) $$\rightarrow (A\theta\lambda_1'\Psi \,\&\, B\theta'\lambda_1\Psi) \vee (A\theta\lambda_2'\Psi \,\&\, B\theta'\lambda_2\Psi)$$

By induction hypothesis ((7) is ground since (6) is)

$$\rightarrow (A \,\&\, B)\theta\lambda_1\Psi \vee (A \,\&\, B)\theta\lambda_2\Psi$$

$$\rightarrow (A \,\&\, B)\theta(\lambda_1 \vee \lambda_2)\Psi$$

$$\rightarrow (A \,\&\, B)\theta\lambda\psi$$

QED.

Soundness Theorem for Generalized ADD-SPLIT

Theorem 1. If θ and λ are pure disjunctive substitutions then

(8) $$\forall\psi(H \rightarrow A)\theta\psi$$

(9) $$\&\ \forall\psi(H \rightarrow B\theta')\lambda\psi$$

$$\rightarrow \forall\psi(H \rightarrow A \,\&\, B)(\theta\lambda \vee \lambda)\psi$$

Proof. We need to show

(10) $$(H \rightarrow A \,\&\, B)(\theta\lambda \vee \lambda)\psi$$

for every substitution ψ such that (10) is ground.

Rewriting this we get

$$(H\theta'\lambda'\psi \,\&\, H\lambda'\psi) \rightarrow ((A \,\&\, B)\theta\lambda\psi \vee (A \,\&\, B)\lambda\psi)$$

Now

$H\theta'\lambda'\psi \,\&\, H\lambda'\psi \rightarrow$	$A\theta\lambda'\psi \,\&\, H\lambda'\psi$	by (8) and Lemma 3.3
$\rightarrow$	$A\theta\lambda'\psi \,\&\, B\theta'\lambda\psi$	by (9)
$\rightarrow$	$(A \,\&\, B)\theta\lambda\psi$	by Lemma 4
$\rightarrow$	$(A \,\&\, B)\theta\lambda\psi \vee (A \,\&\, B)\lambda\psi$	

QED.

The only non-trivial rules left are Back Chaining and Forward Chaining. The generalized soundness theorems for these are

Back Chaining:

$$\forall\forall(B \rightarrow C)\theta\forall$$

$$\& \ \forall\forall(H \rightarrow A\theta')\lambda\forall$$

$$\rightarrow \forall\forall((H \ \& \ (A \rightarrow B)) \rightarrow C)(\theta\lambda \vee \lambda)\forall$$

Forward Chaining:

$$\forall\forall(H \rightarrow A)\theta\forall$$

$$\& \ \forall\forall((B \ \& \ B\theta) \rightarrow C\theta)\lambda\forall$$

$$\rightarrow \forall\forall((H \ \& \ (A \rightarrow B)) \rightarrow C)(\theta\lambda \vee \lambda)\forall$$

These require the lemmas

$$\forall\forall(H \rightarrow C)\theta'\lambda'\forall \rightarrow \forall\forall(H\theta'\lambda \rightarrow C\theta'\lambda)\forall \ ,$$

$$\forall\forall(H \rightarrow C)\theta'\lambda'\forall \rightarrow \forall\forall(H\theta\lambda' \rightarrow C\theta\lambda')\forall \ .$$

Soundness Results for the Implemented Prover

The actual implementation does not include generalized substitutions. Instead we observe restrictions that allow us to return an ordinary substitution. In order to show how we do this, we first need to formalize substitutions.

Definition. A substitution θ is a set $\{a_i/x_i : 1 \leq i \leq n\}$ where the x_i's are variables and the a_i's are terms and $a_i \neq x_i$ and $i \neq j \rightarrow x_i \neq x_j$.

Definition. If A is an expression and $\theta = \{a_i/x_i : 1 \leq i \leq n\}$ is a substitution then $A\theta$ is the expression obtained by replacing all the x_i's in A by the corresponding a_i's.

Definition. A composititon, $\theta\lambda$, of two substitutions $\theta = \{a_i/x_i : 1 \leq i \leq n\}$ and $\lambda = \{b_i/y_i : 1 \leq i \leq m\}$ is defined to be the set $\theta\lambda = \{a_i\lambda/x_i : 1 \leq i \leq n\} \cup \{b_i/y_i : 1 \leq i \leq m \text{ and } 1 \leq j \leq n \rightarrow y_i \neq x_j\}$

A composition is clearly a substitution. Composition is associative. $(A\theta)\lambda = A(\theta\lambda)$

Definition. The domain of a substitution $\theta = \{a_i/x_i : 1 \leq i \leq n\}$ is the set $\{x_i : 1 \leq i \leq n\}$. The range of θ is the set $\{a_i : 1 \leq i \leq n\}$. We will say a variable occurs in the range of θ if it belongs to the range or if it occurs in one of the elements of the range.

Definition. A substitution θ is called *normal* if no element of its domain occurs in its range.

If σ is normal then $\sigma\sigma = \sigma$. If θ is normal then $A\theta$ contains no element in the domain of θ.

Definition. Two substitutions θ and λ are said to conflict if their domains are not disjoint.

Lemma 5. If θ and λ are normal and non-conflicting then $\theta\lambda$ is normal if and only if no element in the domain of θ occurs in the range of λ.

Proof. Proof of ->

Suppose x is in the domain of θ and occurs in b which occurs in the range of λ. But by the definition of composition x is in the domain of $\theta\lambda$ and b occurs in the range of $\theta\lambda$. So $\theta\lambda$ is not normal.

Proof of <-

Suppose $\theta\lambda$ is not normal. Then there is an x in the domain of $\theta\lambda$ that occurs in some c in the range of $\theta\lambda$. By the definition of composition either x is in the domain of θ or in the domain of λ. Likewise either c is in the range of λ or there is some a in the range of θ such that $a\lambda = c$.

Suppose x is in the domain of λ. Since λ is normal, if c were in the range of θ then x can not occur in c. Likewise x can not occur in $a\lambda$. So x can not be in the domain of λ.

So x must be in the domain of θ. Since θ is normal, x can not occur in any a in the range of θ. So x can occur in $a\lambda$ only if x occurs in the range of λ. So no matter where c comes from, x would occur in the range of λ.

QED

Lemma 6. If θ, λ, and $\theta\lambda$ are normal and θ and λ do not conflict, then

$$\lambda\theta\lambda = \theta\lambda.$$

Proof. $\theta\lambda$ and $\lambda\theta\lambda$ only differ for elements in the domain of λ. Suppose v is some such variable. By the previous lemma, no element of the domain of θ occurs in the range of λ. Since $v\lambda$ is in the range of λ, $v\lambda\theta = v\lambda$. So $v\lambda\theta\lambda = v\lambda\lambda = v\lambda$ (since λ is normal $\lambda\lambda = \lambda$). Since v is in the domain of λ and since θ and λ do not conflict, v is not in the domain of θ. So $v\theta = v$ and $v\theta\lambda = v\lambda$. So $v\theta\lambda = v\lambda\theta\lambda$. So $\theta\lambda = \lambda\theta\lambda$.

Corollary. $\forall\Psi(H\theta\lambda \rightarrow H\lambda(\theta\lambda))\Psi \;\&\; \theta\lambda = \theta\lambda(\theta\lambda)$ if θ, λ, and $\theta\lambda$ are normal and θ and λ do not conflict.

Theorem 2. If Φ is some substitution,

(11) $\forall\Psi(H \rightarrow A)\theta\Psi$

(12) $\& \forall\Psi(H \rightarrow B\theta)\lambda\Psi$

(13) $\& \forall\Psi(H\theta\lambda \rightarrow H\lambda\Phi)\Psi$

(14) $\& \forall\Psi(B\theta\lambda\Phi \rightarrow B\theta\lambda)\Psi$

$\rightarrow \forall\Psi(H \rightarrow A \& B)\theta\lambda\Psi$

Proof. We need to show that if σ is some substitution such that $(H \rightarrow A \& B)\theta\lambda\sigma$ is ground then

(15) $$H\theta\lambda\sigma \rightarrow A\theta\lambda\sigma \;\&\; B\theta\lambda\sigma.$$

We will do so for an arbitrary σ satisfying that condition.

We begin by assuming

(16) $H\Theta\lambda\Omega$.

By the assumptions on Ω we know that (16) is ground. Combined with (11) we then know

(17) $A\Theta\lambda\Omega$

is ground and true.

Meanwhile we know from (13) and the assumption of (16)

$$\forall\Psi(H\lambda\Phi\Omega)\Psi.$$

Combining this with (12) we know

$$\forall\Psi(B\Theta\lambda\Phi\Omega)\Psi.$$

Further combining this with (14) we get

(18) $B\Theta\lambda\Omega$

which is both ground and true.

Thus by assuming (16) we derived both (17) and (18), thereby proving (15).

QED.

By the previous corollary, it can be seen that the hypotheses about Φ are satisfied by $\Phi = \Theta\lambda$ when Θ, λ, and $\Theta\lambda$ are all normal and Θ and λ do not conflict. This is the normal case.

Theorem_1. If θ, λ, and $\theta\lambda$ are all normal and θ and λ do not conflict, then

$$\forall\vec{x}(h \rightarrow A)\theta\vec{x}$$
$$\& \ \forall\vec{y}(H \rightarrow B\theta)\lambda\vec{y}$$
$$\rightarrow \forall\vec{z}(H \rightarrow A \ \& \ B)\theta\lambda\vec{z}$$

Notes_on_the_Implementation

As we noted earlier, equation (1) is true if there is some θ such that $S\theta$ is ground and true. This means that if the prover returns λ for IMPLY(H)L,S) all that is needed is for some θ to exist such that $S\lambda\theta$ to be ground and true. Specifically we do not need $S\lambda$ to be ground-true. We have used ground-truth to make the inductive proofs tractable. Theorem 1 is much easier to grasp as it was presented than it would be if it were stated as

$$H\theta'\lambda'\vec{x} \rightarrow A\theta\lambda'\vec{y}$$
$$\& \ B\lambda'\vec{y} \rightarrow B\theta'\lambda\vec{z}$$
$$\rightarrow (H \rightarrow A \ \& \ B)(\theta\lambda \vee \lambda)\vec{z}$$

In the actual implementation we relax the requirement that the substitution returned from IMPLY makes the input ground-true. This is necessary for proving such simple theorems as

$$\exists x(P(x) \rightarrow P(a) \ \& \ P(b)).$$

In this case we return the substitution (a/x, b/x). In applying substitutions we check to see if a substitution that has a conflict

is applied to a formula that has the variable for which the conflict occurs. If so, we fail on that subgoal.

Likewise if we have a substitution {a/x, f(x)/y} (which is not normal) we allow it to be used but prohibit its composition with some substitution such as {b/x} while allowing composition with {a/x}.

Examples

Example_1. P(x) => (P(a) & P(b))

This requires the generalized substitution ({a/x} v {b/x}). In the implemented prover the substitution {a/x, b/x} would be returned.

Example_2.

$$P(a,b,z) \;\&\; Q(a,b,d,y) \;\&\; R(a,b,f) \;\&\; R(e,c,d) \Rightarrow (P(x,y,z) \;\&\; Q(a,y,z,c)) \;\&\; R(x,y,z)$$

This is not a theorem. The proof of

$$P(x,y,z) \;\&\; Q(a,y,z,c)$$

requires the composition of the substitutions {a/x, b/y} and {d/z, c/y}. With generalized substitutions we then have to prove

$$R(x,y,z)(\{a/x,\ b/y,\ d/z\} \;\&\; \{d/z,\ c/y\})$$

which translates to

$$R(a,b,d) \;\&\; R(x,c,d).$$

This can not be proved from the hypotheses.

The composition of the two substitutions in the implemented prover would be

$$\{a/x,\ b/y,\ d/z,\ c/y\}.$$

When this substitution is applied to R(x,y,z) the prover notes that there is a conflict in the substitution for y and that y appears in R(x,y,z). Because of this, the prover fails the goal.

Example_3.

$$P(a,b,z) \ \&\ Q(a,b,d,y)$$
$$\&\ R(a,b,d) \ \&\ R(e,c,d)$$
$$\Rightarrow (P(x,y,z) \ \&\ Q(a,y,z,c)) \ \&\ R(x,y,z)$$

This is similar to example 2 but is a theorem. With generalized substitutions it would be proved but the implemented prover would fail it for the same reasons as before.

Example_4. $(P(f(x)) \ \&\ Q(g(y)) \Rightarrow P(y) \ \&\ Q(x)$

This theorem is proved using the generalized substitution $(\{f(g(y))/y,\ f(y)/x\} \vee \{g(y)/x\})$. The implemented prover returns simply $\{f(g(y))/y,\ g(y)/x\}$ which is not normal.

Example_5. $Q(a,f(b)) \Rightarrow [(P(f(x)) \Rightarrow (P(f(a)) \& P(y))) \ \&\ Q(x,y)]$

The first step in proving this is to prove

$$Q(a,f(b)) \ \&\ P(f(x)) \Rightarrow P(f(a)) \ \&\ P(y).$$

This requires the combination of the substitutions $\{a/x\}$ and $\{f(x)/y\}$. With generalized substitutions we then need to prove

$$(Q(a,f(b)) \Rightarrow Q(x,y)) \{a/x,\ f(x)/y\} \alpha \{f(x)/y\}$$

which becomes

$$Q(a,f(f)) \Rightarrow (Q(a,f(x)) \wedge Q(x,f(x))).$$

This is clearly not true.

The implemented prover combines the two substitutions into {a/x, f(x)/y} (which is not normal). It then succeeds in proving

$$Q(a,f(b)) \Rightarrow Q(a,f(x))$$

with the substitution {b/x}. However when it tries to combine this substitution with the previous one, the prover recognizes its mistake and fails the subgoal.

References

0. W. W. Bledsoe and Mabry Tyson. The UT Interactive Prover. University of Texas, Math Department Memo ATP 17, May 1975

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. Computers and Thought, Feigenbaum and Feldman (Eds) 134-152.

12. 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'1. 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'1. 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'1. 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 improved proof procedure. Theoria 25, 102-139 (1960).

26. Nils Nilsson. Artificial Intelligence. Including a 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 Mat. 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.

50. G.D. Plotkin. Building Equational Theories. Machine Intelligence 7, 1972, 73-89.

51. Dallas S. Lankford. Canonical Algebraic Simplification in Computational Logic. Univ. of Texas Math. Dept. Memo ATP-25, 1975.

52. Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Ph.D. Thesis, MIT. MAC TR-84, Feb. 1971.

53. J.R. Slagle. Automatic Theorem Proving with Built-in Theories of equality, Partial ordering and Sets. JACM, 19(1972), 120-135.

54. J.R. Slagle and L. Norton. Experiments with an Automatic Theorem Prover having partial ordering rules. CACM, 16(1973), 682-688.

55. J.R. Slagle. Automatic Theorem Proving with renamable and semantic resolution. JACM, 14(1967), 687-697.

56. L. Siklossy, A. Rich, and V. Marinov. Breadth-First Search: Some Surprising Results. AI Jour., 4(1973), 1-27.

57. L. Siklossy. Disprover.

58. René Reboh and Earl Sacerdoti. A preliminary QLISP Manual. Stanford Research Inst. AI Center Tech. Note 81, August, 1973.

59. Ira Goldstein. Elementary Geometry Theorem Proving. MIT-AI Lab. Memo. 280, April, 1973.

60. G.J. Sussman, T. Winograd, and E. Charniak. Micro-planner Manual. MIT-AI Lab. Memo. 203A, Dec. 1971.

61. Frank Brown. (Unpublished work on Automatic Theorem Proving). Univ. of Edinburgh, 1975.

62. Marvin Minsky. A framework for representing knowledge. In P. Winston (Ed). The Psychology of Computer vision. New York: McGraw-Hill. in press.

63. W.W. Bledsoe and A.M. Ballantyne. Automatic proofs of theorems in analysis using non-standard techniques. Univ. of Texas Math. Dept. Memo ATP-23, July, 1975.

64. W.W. Bledsoe. A New Method for proving certain Presburger formulas. ISCAI-75, Sept. 1975, Tblisi, USSR.

65. R.E. Kling. A paradigm for reasoning by analogy. AI Jour. 2, (1971), 147-178.

66. R.L. de Carvalho. Some Results in Automatic Theorem-Proving with Applications in Elementary Set Theory and topology. Ph.D. Thesis, Dept. of C.S., Univ. of Toronto, Canada. Tech. Report no. 71, Nov. 1974.

67. D.C. Cooper. Theorem Proving in Computers. Advances in Programming and Non-numeric Computation (L. Fox, ed.), 155-182.