Table of Contents

The UT Interactive Prover

W.W. Bledsoe and Mabry Tyson

1. Introduction

The prover we describe in this paper is a natural deduction type system that proves theorems in first order logic, and some extensions of that by subgoaling, splitting, matching, and rewriting, simplification, and other such procedures. It has been partially described in [1-6] but there remains some uncertainty as to exactly what it does. We will attempt to explain it in a precise manner, but the ultimate explanation is in the LISP program itself, which is available upon request.

There is no attempt here to review all the literature on automatic theorem proving. Suffice it to say that our work is based to a great extent on that of others. The reader is referred to Chang and Lee [7], and Loveland [8] for information and references on resolution type systems, and to the work of Allen and LucKham [9], Guard, et al [10], and Huet [11], on interactive provers. Our prover is in the spirit of Newell, Simon, and Shaw [12], Gelerntner [13], and has much in common with the work of Gentzen [14], Nevins [15-17], Reiter [18], Ernst [19], Bibel [20], Hewitt [21], McDermott and Sussman [22], Wang [23], Maslov [48], and Rulifson, et al [24]. See also Nilsson's Review [26].

In using the interactive prover, the theorem (and subsequent subgoals) are shown on the user terminal's screen in a natural, easy to read form, and the user is provided with several interactive commands (see Section 7) for communicating with the prover. The prover is based upon natural deduction (or is a Gentzen type system [14-17,25,20,49]), as opposed to a "less natural" system such as resolution. When the human user desired to interact directly with the prover, the dialogue is expressed in terms that are (hopefully) natural and convenient for him. The intent is that the computer will act as a support to the user in the proof of a theorem; although, the machine-only system is a powerful prover in its own right.

The interactive policy of the prover is based on the premise that if the prover can construct a proof it will do so fairly quickly. For each theorem or subgoal, a time limit is set; if a proof has not been constructed in that time, the prover stops and waits for interactive direction. The user then has available a number of commands for displaying the theorem and the details of what the prover has done so far. Using these commands the user isolates the difficulty and then can allocate more time, direct the prover into a new line of reasoning, supply additional information (hypotheses, lemmas, definitions) about the whole thing, or simply assume that the current subgoal is true and go on to another part of the proof. Often proofs of subgoals will fail initially because not enough information has been provided. (Failure may well, of course, be due to attempting to prove a non-theorem). A very useful feature of the prover is that these additional hypotheses need not be stated initially, but rather can be supplied at the point in the proof when it is realized that they are necessary. This prevents the objectionable activity of the user having to prove the theorem himself before he asks the prover to do so, in order to determine what additional hypotheses and definitions will be needed.

This system was developed by Bledsoe's group at The University of Texas. While it is a general theorem prover, earlier versions were mainly exercised on theorems in set theory [2], limit theorems [3,45] and topology [1], and a current version is working on theorems arising from program verification [6]. It has been extended [5,27] to handle these program verification theorems; Larry Fagan and Peter Bruell at Information Sciences Institute, USC, have helped considerably in this extension.

2. IMPLY and HOA

The central routines of PROVER are IMPLY and HOA which are described below. They attempt to establish the validity of an expression of the form

$$(H \longrightarrow C) \quad ,$$

(H and C are arguments of IMPLY), by applying a set of (sound) rules (see Tables I and II). These routines are recursive, they call each other and themselves, but the initial call is to IMPLY.

These two algorithms, and their supporting subroutines, form a natural deduction type system. It is like a Gentzen system [14,25], but is more "human like" in that no attempt is made to force the formula being proved into a canonical form. In particular the implication symbol, $\longrightarrow$, is retained, and we believe that the proof proceeds in a manner that would be natural to a mathematician.

IMPLY has five arguments (TYPELIST,H,C,TL,LT) but we will deal with only two of them, H and C at this time. TL and LT are discussed later but TYPELIST is not discussed in this paper. See [27]. HOA has three arguments (B,C,HL) and we will deal with only two of them, B and C, at this time.

When we make a call IMPLY(H,C), the algorithm IMPLY tries to establish the validity of the formula $(H \longrightarrow C)$ by applying a set of (sound) rules. Similarly a call to HOA(B,C) causes the algorithm HOA to try to establish the validity of $(B \longrightarrow C)$.

Actually, neither algorithm is complete[1], but they call upon each other to perform various tasks. IMPLY performs AND-SPLITS, as when the conclusion is a conjunction (Rule 4) or the hypothesis is a disjunction (Rule 3); and HOA handles OR-SPLITS, as when the conclusion is a disjunction (Rule 4) or the hypothesis is a conjunction (Rule 6) or an implication (Rule 7, Back-Chaining). Additionally IMPLY handles various manipulations of the conclusion C, while HOA handles those for the hypothesis B.

A theorem being proved is first sent to IMPLY which calls HOA and itself as needed. Before a formula E is initially sent to IMPLY, it is first converted to quantifier free form (but without converting it to prenex form) by skolemization (see Appendix 1). This (usually) produces skolem variables in E which are replaced by terms during the proof. A substitution θ is derived which consists of these replacements.

If H and C are formulas, then IMPLY either returns NIL or a substitution θ, such that

$$(H\theta \longrightarrow C\theta)^{2}$$

[1]Even the combination of both of them working together is not complete, in that there are valid formulas which PROVER cannot prove. See Appendix 2.

[2]Sometimes when multiple substitutions are necessary the implication $(H\theta \longrightarrow C\theta)$ is not valid, even though $(H \longrightarrow C)$ is. See Appendix 3.

is valid (usually a theorem in propositional logic). θ is usually the most general such substitution. If no substitution is needed them IMPLY returns "T". It will return "NIL" if $(H \longrightarrow C)$ is not valid or if it cannot find a proof in the prescribed time limit.

Similarly HOA and many of the supporting routines such as UNIFY return substitutions θ.

The routines IMPLY and HOA are described in algorithmic form in Tables I and II. These tables give only the basic rules of IMPLY and HOA. Some additional details are mentioned in footnotes and in the later descriptions.

A formula E is initially sent to IMPLY by a call IMPLY(NIL,E).

Table I

ALGORITHM

IMPLY(H, C)

	IF	ACTION	RETURN
1.	C ≡ "T" or H ≡ "FALSE"		"T"
2.	TYPELIST*		
3.	$H \equiv (A \vee B)$[3]		IMPLY(NIL, $(A \longrightarrow C) \wedge (B \longrightarrow C)$)
4.	(AND-SPLIT) $C \equiv (A \wedge B)$	Put θ: = IMPLY(H,A)	
4.1	$\theta \equiv$ NIL		NIL
4.2	$\theta \not\equiv$ NIL	Put λ: = IMPLY(H,Bθ)[4]	
4.3	$\lambda \equiv$ NIL		NIL
4.4	$\lambda \not\equiv$ NIL		$\theta \circ \lambda$[5]
5.	(REDUCE)	Put H: = REDUCE(H) Put C: = REDUCE(C)	
5.1	C ≡ "T" or H ≡ "FALSE"	Go to 1	
5.2	$H \equiv (A \vee B)$	Go to 3	
5.3	$C \equiv (A \wedge B)$	Go to 4	
5.4	ELSE	Go to 6	

---

*See [27].

[3]By the expression "$H \equiv (A \vee B)$" we mean that H has the form "$A \vee B$". Rules 4 and 3 are called "AND-SPLIT's". See [2] and [19].

[4]If θ has two entries, a/x, b/x with $a \neq b$, or g(y)/x and f(x)/y, then two λ's, λ_1 and λ_2 are computed, one for each case, and $\lambda_1 \circ \lambda_2$[5] is returned for λ. See App. 3

[5]This is just (APPEND $\theta\lambda$). If θ has an entry a/x and λ has an entry b/x where $a \neq b$, then leave both values in $\theta \circ \lambda$. For example, if θ = (a/x b/y), λ = (c/x d/z) then $\theta \circ \lambda$ = (a/x b/y c/x d/z).

IMPLY(H,C) Cont'd

	IF	ACTION	RETURN
6.	$C \equiv (A \vee B)$		HOA(H,C)
7.	(PROMOTE) $C \equiv (A \longrightarrow B)$		$IMPLY(H \wedge A, B)$[6]
7.1	Forward Chaining		
7.2	PEEK forward chaining		
8.	$C \equiv (A \longleftrightarrow B)$		$IMPLY(H, (A \longrightarrow B) \wedge (B \longrightarrow A))$
9.	$C \equiv (A = B)$	Put $\theta := UNIFY(A,B)$	
9.1	$\theta \not\equiv NIL$		θ
9.2	$\theta \equiv NIL$	Go To 10	
10.	$C \equiv (\sim A)$		$IMPLY(H \wedge A, NIL)$
11.	INEQUALITY*		
12.	(call HOA)	Put $\theta := HOA(H,C)$	
12.1	$\theta \not\equiv NIL$		θ
12.2	(PEEK) $\theta \equiv NIL$	Put PEEK[7] light "ON" Put $\theta := HOA(H,C)$	
12.3	$\theta \not\equiv NIL$		θ
12.4	$\theta \equiv NIL$	Go To 13	

---

[6] Actually we call $IMPLY(OR\text{-}OUT(H \wedge A), AND\text{-}OUT(B))$. See p. 17.

[7] See p.30. The PEEK Light is turned off at the entry to IMPLY.

IMPLY(H,C) Cont'd

	IF	ACTION	RETURN
13.	(Define C)	Put C': = DEFINE(C)	
13.1	C' ≡ NIL	Go To 14	
13.2	C' ≢ NIL		IMPLY(H,C')
14.	(See Section 2 of [27])		
15.	ELSE		NIL

Table II

ALGORITHM

HOA(B,C)

	IF	ACTION	RETURN
1.	Time limit Exceeded		NIL
2.	(MATCH)	Put $\theta := \text{UNIFY}(B,C)$	
2.1	$\theta \not\equiv$ NIL		θ
2.2	PEEK (See Section 4)		HOA(B,C)
3.	PAIRS (See Section 4)		
4.	(OR-SPLIT) $C \equiv (A \vee D)$	Put $C' := \text{AND-OUT}(C)$	
4.1	$C' \not\equiv C$		IMPLY(H,C')
4.2	$C' \equiv C$	Put $\theta := \text{HOA}(B \wedge \sim D, A)$[8]	
4.3	$\theta \not\equiv$ NIL		θ
4.4	$\theta \equiv$ NIL		$\text{HOA}(B \wedge \sim A, D)$[8]
5.1	$C \equiv (A \longrightarrow D)$		IMPLY(B,C)
5.2	$C \equiv (A \wedge D)$		IMPLY(B,C)
6.	$B \equiv (A \wedge D)$	Put $\theta := \text{HOA}(A,C)$	
6.1	$\theta \not\equiv$ NIL		θ
6.2	$\theta \equiv$ NIL		HOA(D,C)

[8] In Step 4.2, the "$\sim$" in $(\sim D)$ is pushed to the inside; e.g., $\sim(\sim P)$ goes to P, and $\sim(P \longrightarrow Q)$ goes to $P \wedge \sim Q$. If D contains no "$\sim$" or "$\longrightarrow$" then $(\sim D)$ is omitted and the call is made HOA(B,A). Similarly in Step 4.4.

HOA(B,C) Cont'd

	IF	ACTION	RETURN
7.	(Back-chaining) $B \equiv (A \longrightarrow D)$	Put $\theta := ANDS(D,C)$*	
7.1	$\theta \equiv NIL$	Go To 7E	
7.2	$\theta \not\equiv NIL$	Put $\lambda := IMPLY(H, A\theta)$[4]	
7.3	$\lambda \equiv NIL$	Go To 8	
7.4	$\lambda \not\equiv NIL$		$\theta \circ \lambda$
7E.	$B \equiv (A \longrightarrow a = b)$	Put $\theta := HOA(a=b, C)$	
7E.1	$\theta \equiv NIL$		NIL
7E.2	$\theta \not\equiv NIL$	Put $\lambda : \equiv IMPLY(H, A\theta)$[4]	
7E.3	$\lambda \equiv NIL$	Go To 8	
7E.4	$\lambda \not\equiv NIL$		$\theta \circ \lambda$
8.	$B \equiv (A \longleftrightarrow D)$		$HOA((A \longrightarrow D) \wedge (D \longrightarrow A), C)$
9.	$B \equiv (a = b)$	Put $Z := MINUS\text{-}ON(a,b)$	
9.1	$Z \equiv 0$		NIL
9.2	Z is a number		T
9.3	Z is not a number	Put $a' := CHOOSE(a,b)$, $b' := OTHER(a,b)$ (see p. 20) Put $H' := H(a'/b')$, $C' := C(a'/b')$	$IMPLY(H', C')$
10.	$B \equiv (A \vee D)$		$IMPLY(B,C)$
11.	$B \equiv \sim A$		$IMPLY(H, A \vee C)$[8]
12.	ELSE		NIL

*ANDS is explained on p.15.

[8] Actually we use AND-PURGE(H,~A) instead of H, which removes ~A from H.

When proving a theorem of the form

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

IMPLY uses Rule 4 to split it into the two subgoals

$$(H \longrightarrow A)$$

and

$$(H \longrightarrow B)$$

which it tries to prove separately. It is (of course) necessary that the substitution θ derived for $(H \longrightarrow A)$ be applied to B (but not to H) in proving the second subgoal, $(H \longrightarrow B\theta)$.[9]

The fourth argument, TL, of IMPLY is a "theorem label" (or more appropriately, a "subgoal label"), which is a sequence of 1's and 2's that indicate the progress that has been made in proving the theorem. For example, a theorem

$$(H \longrightarrow C_1 \wedge C_2)$$

would have theorem label (1) and its two principal subgoals

$$(H \longrightarrow C_1) \quad \text{and} \quad (H \longrightarrow C_2)$$

would have theorem labels (1 1) and (1 2). Such theorem labels are exhibited in the left margin for the examples given in this paper. In addition to 1's and 2's we also utilize other letters such as H, P, and =, to indicate other actions of the prover.

---

[9] The reader can see the necessity of this rule by considering the three examples $(P(a) \wedge Q(a) \longrightarrow P(x) \wedge Q(x))$, $(P(a) \wedge Q(b) \longrightarrow P(x) \wedge Q(x))$, and $(P(x) \longrightarrow P(a) \wedge P(b))$, where x is a skolem variable, and a and b are constants.

Some Examples

Ex. 1. $(A \longrightarrow A)$

A call is made to

$$\text{IMPLY}(\text{NIL},\ A \longrightarrow A)$$

which in turn uses Rule 7 to call

$$\text{IMPLY}(A, A)$$

which uses Rule 11 to call

$$\text{HOA}(A, A)$$

which returns "T" by HOA Rule 2.

In order to shorten the presentation of this example and those that follow, we will use the notation

(TL) $(D \Rightarrow C)$

in place of IMPLY(D,C) and HOA(D,C).

Thus Ex. 1 becomes

(1) $(\text{NIL} \Rightarrow (A \longrightarrow A))$

(1) $(A \Rightarrow A)$ I 7
Returns "T" I 11, H 2

The theorem label, which is (1) in this case, will be exhibited in the left margin, and some rule numbers from Tables I and II will be given in the right margin, with the prefix I for Table I and the Prefix H for Table II.

Ex. 2. $\forall a(\ \forall x\ \ P(x) \longrightarrow P(a))$.

(1)	$(NIL \Rightarrow (P(x) \longrightarrow P(a_o)))$	Skolemized
(1)	$(P(x) \Rightarrow P(a_o))$	I 7
	UNIFY$(P(x),\ P(a_o))$ returns a_o/x	H 2

Henceforth we will drop "NIL ⇒" and write "A" instead of "NIL ⇒ A". Thus Ex. 2 becomes

(1)	$(P(x) \longrightarrow P(a_o))$	
(1)	$(P(x) \Rightarrow P(a_o))$	I 7
	Returns a_o/x	H 2

ANDS.

In the following example we use the algorithm ANDS. It is a mini version of IMPLY which handles only theorems of the form

$$(H_1 \wedge H_2 \wedge \ldots \wedge H_n \longrightarrow C)$$

where $(H_i\theta = C\theta)$ for some θ. (In which case θ is returned).

Ex. 3. $\forall a(P(a) \wedge \forall x(P(x) \longrightarrow Q(x)) \longrightarrow Q(a))$.

(1)	$(P(a_o) \wedge (P(x) \longrightarrow Q(x)) \longrightarrow Q(a_o))$	
(1)	$(P(a_o) \wedge (P(x) \longrightarrow Q(x)) \Rightarrow Q(a_o))$	I 7
	$(P(a_o) \Rightarrow Q(a_o))$	H 6
	Returns NIL	
	$((P(x) \longrightarrow Q(x)) \Rightarrow Q(a_o))$	H 6.1
	ANDS $(Q(x), Q(a_o))$	H 7
	Returns a_o/x	Back-chaining
(1 H)	$(P(a_o) \wedge (P(x) \longrightarrow Q(x)) \Rightarrow P(a_o))$	H 7.2
	Returns "T"	H 6, H 2
	Returns a_o/x for (1)	H 7.2.2.2

Ex. 3'. $(A \vee B \longrightarrow A \vee B)$

(1)	$(A \vee B \Rightarrow A \vee B)$	I 7
(1 1)	$(A \Rightarrow A \vee B)$	I 3, 4, 7
	$(A \Rightarrow A)$	H 4.2, Footnote 5
	"T"	H 2
(1 2)	$(B \Rightarrow A \vee B)$	I 4.2
	"T"	H 4.2, H 2

<u>Ex. 3"</u>. $(A \longrightarrow B \vee C)$ (Not a theorem)

In this example if we applied HOA Step 4.2 without the footnote we would obtain an indefinite repetition as follows:

(1)	$(A \Rightarrow B \vee C)$		I 7
	$(A \wedge \sim C \Rightarrow B)$		H 4.1
	$(A \Rightarrow B)$	NIL	H 6
	$(\sim C \Rightarrow B)$		H 6.2
	$(A \Rightarrow B \vee C)$		H 11
	Repeat		

But by preventing the addition of $\sim C$ to the hypothesis, unless it is fundamentally changed, we eliminate this problem.

(1)	$(A \Rightarrow B \vee C)$		I 7
	$(A \Rightarrow B)$	NIL	H 6
	$(A \Rightarrow C)$	NIL	H 6.2
	NIL is returned for (1).		

AND-OUT is an algorithm which puts expressions in conjunctive form (but does not convert implications).

For example

AND-OUT $(A \vee (B \wedge C))$ returns $((A \vee B) \wedge (A \vee C))$,

AND-OUT $(A \vee (D \longrightarrow B \wedge C))$ returns $(A \vee (D \longrightarrow B \wedge C))$.

Similarly OR-OUT puts expressions in disjunctive form.

Ex. 3'''. $B \longrightarrow A \wedge (\sim A \vee B)$

This example shows the utility of "AND-OUT" in Rule H 4. For without it we would get

(1) $(B \Rightarrow A \wedge (\sim A \vee B))$ I 7

If we don't use AND-OUT of H 4

(1 1) $(B \Rightarrow A)$ Returns NIL

(1 2) $(B \Rightarrow \sim A \wedge B)$ Returns NIL

Returns NIL for (1)

Since we do use AND-OUT in H 4, we get

(1) $(B \Rightarrow A \vee (\sim A \wedge B))$ I 7

(1) $(B \Rightarrow (A \vee \sim A) \wedge (A \vee B))$ H 4

(1) $(B \Rightarrow A \vee B)$ I 4
REDUCE Rules 15, 17

(1 1) $(B \Rightarrow A)$ Returns NIL

(1 2) $(B \Rightarrow B)$ H 4.1

Returns "T" for (1 2) and (1) as desired H 2, H 4. 4

<u>Ex. 3''''</u>. $(A \wedge (\sim A \vee B) \longrightarrow B)$

Similarly OR-OUT is required in I 7. Because without it we would get

(1)	$(A \wedge (\sim A \vee B) \Rightarrow B)$		I 7
(1 1)	$(A \Rightarrow B)$	Returns NIL	H 6
(1 2)	$(\sim A \vee B \Rightarrow B)$		
(1 2 1)	$(\sim A \Rightarrow B)$	Returns NIL	I 3

Returns NIL for
(1 2) and (1)

But since we use OR-OUT in I 7 we get

(1)	$(A \wedge (\sim A \vee B) \longrightarrow B)$	Original
(1)	$(\text{OR-OUT}(A \wedge (\sim A \vee B)) \Rightarrow B)$	I 7
	$((A \wedge \sim A) \vee (A \wedge B) \Rightarrow B)$	
(1 1)	$(A \wedge \sim A) \Rightarrow B)$	I 4
	$(\text{FALSE} \Rightarrow B)$	I 5
	"T"	I 1
(1 2)	$(A \wedge B \Rightarrow B)$	I 4.2
	"T"	H 6.2, H 2
	Returns "T" for (1) as desired	I 4.4

Substituting Equals

HOA Rule 9 gives the prover an ability to substitute equals. When an equality unit $(a = b)$ is in the hypothesis, the program uses the algorithm CHOOSE(a,b) to select either a or b, and replaces it by the other in H and C. CHOOSE selects neither if neither a or b occurs in H or C. It selects a if b is a number, and vice versa. It will not choose a if b occurs in a, and vice versa. In the interactive mode the user can enter this decision process (see Section 7).

3. Definitions and Reduction

Definitions.

Rule 12 of IMPLY calls DEFINE(C) which expands definitions from a stored list. Table III gives some such definitions.

When the defining form introduces quantifiers (e.g., Rule 2 of Table III) it is necessary to eliminate these quantifiers by skolemization. We have done this by pre-skolemizing the formula in the table, but it is necessary to store two such skolemizations because the correct one will depend on whether the formula occupies a positive[10] or negative position in the theorem being proved. For example, $(A \subseteq B)$ is replaced by $(x_o \in A \rightarrow x_o \in B)$ in

$$(H \longrightarrow A \subseteq B)$$

whereas it would be replaced by $(x \in A \rightarrow x \in B)$ in

$$(A \subseteq B \longrightarrow C) \ .$$

[10] See [23, 3] and Appendix 1.

Table III

SOME DEFINITIONS

	Formula Being Defined	Defining Form
1.	$(A = B)$[11]	$(A \subseteq B \wedge B \subseteq A)$
2.	$(A \subseteq B)$	$\forall x(x \in A \rightarrow x \in B)$
		Skolem form[12]
		$(x_o \in A \rightarrow x_o \in B)$ in "Conclusion"
		$(x \in A \rightarrow x \in B)$ in "Hypothesis"
3.	$(A \cup B)$	$\{x: x \in A \vee x \in B\}$
4.	$(A \cap B)$	$\{x: x \in A \wedge x \in B\}$
5.	$\bigcup_{t \in S} A(t)$	$\{x: \exists t(t \in S \wedge x \in A(t))$[12]
6.	$\bigcap_{t \in S} A(t)$	$\{x: \forall t(t \in S \rightarrow t \in A(t))$[12]
7.	subsets(A)	$\{x: x \subseteq A\}$
7'.	sb(A)	subsets(A)
8.	range f	$\{y: \exists x(y = f(x))\}$
9.	Oc F	(Open F $\wedge$ Cover F)

[11] A different symbol is used for set equality to distinguish it from the arithmetic equality. Here in Entry 1 we mean set equality.

[12] When the defining form introduces quantifiers, two versions of its skolemization are needed. See page 21.

REDUCE

Rule 5 of IMPLY calls REDUCE(H) and REDUCE(C). If E is a formula then a call to REDUCE(E) causes the algorithm REDUCE to apply a set of rewrite rules to convert parts of the formula E. See [2,29-36]. Table IV gives some examples of rewrite rules in use.

REDUCE helps convert expressions into forms which are more easily proved by IMPLY. Also the rewrite table is a convenient place to store facts that can be conveniently used by the machine as they are needed. For example, REDUCE returns "T"(TRUE), when applied to the formulas (Closed(Clsr A)), (Open $\emptyset$), (Open(interior A)), ($\emptyset \subseteq$ A).

Table IV

REDUCE Rewrite Rules

	INPUT	OUTPUT
1.	$(t \in A \cap B)$	$(t \in A \wedge t \in B)$
2.	$(t \in A \cup B)$	$(t \in A \vee t \in B)$
3.	$(t \in \{x: P(x)\})$	$P(t)$
4.	$(t \in A)$ If A has Definition $\{x: P(x)\}$	$P(t)$
5.	$t \in$ subsets(A)	$t \subseteq A$
6.	$t \subseteq A \cap B$	$(t \subseteq A \wedge t \subseteq B)$
7.	$(A \cap A)$	A
8.	$(A \cup A)$	A
9.	$(A \cap \emptyset)$	$\emptyset$
10.	$(A \cup \emptyset)$	A
11.	$(\emptyset \subseteq A)$	"T"
12.	$A \in \{B\}$	$A = B$
13.	(range $\lambda x f(x)$)	$\{y: \exists x(y = f(x)\}$
14.	(Choice $A \in A$)	$A \neq \emptyset$
15.	$(A \vee \sim A)$	"T"
16.	$(A \wedge \sim A)$	"FALSE"
17.	("T" $\wedge$ A)	A
18.	(A $\wedge$ "T")	A

Table IV (Con't)

INPUT	OUTPUT
19. $(A \vee \text{"T"})$	"T"
20. $(\text{"T"} \vee A)$	"T"
21. $(G \subseteq\subseteq G)$[13]	"T"
22. $(G \subseteq\subseteq \overline{G})$[13]	"T"
23. $(A \subseteq A)$	"T"
24. $(A \subseteq \overline{A})$	"T"
25. A ∧ FALSE	FALSE
26. FALSE ∧ A	FALSE
27. A ∨ FALSE	A
28. FALSE ∨ A	A

etc.

[13] It need not concern the reader here but $\overline{G}$ is the set of closures of members of G. That is if $\overline{A}$ is the closure of the set A, then $\overline{G} = \{\overline{A}: A \in G\}$. And $(H \subseteq\subseteq J)$ means that H is a refinement of J, that is, each member of H is a subset of a member of J.

Ex. 4. $\forall A \ \forall B \ (A \subseteq A \cup B)$

(1) $(A_o \subseteq A_o \cup B_o)$

(1) $(x_o \in A_o \Longrightarrow x_o \in (A_o \cup B_o))$ I 12

(1) $(x_o \in A_o \Longrightarrow x_o \in A_o \vee x_o \in B_o)$ I 5

REDUCE Rule 2

(1) $(x_o \in A_o \Rightarrow x_o \in A_o \vee x_o \in B_o)$ I 7

(1 1) $(x_o \in A_o \Rightarrow x_o \in A_o)$ H 4.1

(1 1) "T" H 2

Return "T" for (1).

Notice how closely this parallels the usual mathematician's proof, i.e.,

$A \subseteq A \cup B$

$(x \in A \longrightarrow x \in (A \cup B))$

$(x \in A \longrightarrow x \in A \vee x \in B)$

TRUE.

Ex. 5. $\forall A \ \forall B$ (subsets $(A \cap B)$ = subsets $(A) \cap$ subsets (B))

(1) $\text{subsets}(A_o \cap B_o) = \text{subsets}(A_o) \cap \text{subsets}(B_o)$

We will here contract "subsets" to "sb" and drop the subscripts.

(1) $sb(A \cap B) = sb(A) \cap sb(B)$

(1) $[sb(A \cap B) \subseteq sb(A) \cap sb(B)] \wedge [sb(A) \cap sb(B) \subseteq sb(A \cap B)]$ I 12
Definition 1

(1 1) $[sb(A \cap B) \subseteq sb(A) \cap sb(B)]$ I 4
This is an AND-SPLIT

(1 1) $[t_o \in sb(A \cap B) \Longrightarrow t_o \in (sb(A) \cap sb(B))]$ I 12
Definition 2

(1 1) $[t_o \subseteq A \cap B \Longrightarrow t_o \in sb(A) \wedge t_o \in sb(B)]$ I 5
REDUCE Rules 5, 1

(1 1) $[t_o \subseteq A \wedge t_o \subseteq B \Rightarrow t_o \subseteq A \wedge t_o \subseteq B]$ I 5, I 7
REDUCE Rules 6, 5

Return "T" for (1 1) I 4, H 6, H 2

(1 2) $[sb(A) \cap sb(B) \subseteq sb(A \cap B)]$
Return "T" for (1 2) (Similarly)
Return "T" for (1) .

It should be noted that the use of Definitions and REDUCE on this example has eliminated the need for additional hypotheses (or axioms). The required hypotheses must be given by the user but they are given once and for all in REDUCE and definition tables and never used except when needed in the proof. An ordinary resolution proof or Gentzen type proof which did not use such mechanisms would require four additional axioms and a lengthy proof.

1. $(\alpha = \beta \longleftrightarrow \forall t(t \in \alpha \leftrightarrow t \in \beta))$
2. $(t \in A \cap B \leftrightarrow t \in A \wedge t \in B)$
3. $(t \in \text{subsets } A \leftrightarrow t \subseteq A)$
4. $(t \subseteq A \cap B \leftrightarrow t \subseteq A \wedge t \subseteq B)$.

Rule 4 of Table IV is a conditional rule. When attempting to convert a formula of the form $t \in A$, the algorithm REDUCE first checks to see if A has a definition of the form $\{x: P(x)\}$, in which case it (in effect) instantiates that definition and applies Rule 3. For example the expression

$$x_o \in \bigcup_{t \in Q} A(t)$$

is reduced by Rule 4 of Table IV and Rule 5 of Table III, to

$$\exists t(t \in Q \wedge x_o \in A(t))$$

(or actually to the skolemized form $(t \in Q \wedge x_o \in A(t))$).

Ex. 6. . $(A \in G \longrightarrow A \subseteq \bigcup_{B \in G} B)$

(1)	$(A_o \in G \Rightarrow A_o \subseteq \bigcup_{B \in G} B)$	I 7
(1)	$(A_o \in G \Rightarrow (t_o \in A_o \longrightarrow t_o \in \bigcup_{B \in G} B))$	I 12 Definition 2
(1)	$(A_o \in G \Rightarrow (t_o \in A_o \longrightarrow B \in G \wedge t_o \in B)$	I 5 REDUCE Rule 4, Definition 5
(1)	$(A_o \in G \wedge t_o \in A_o \Rightarrow B \in G \wedge t_o \in B)$	I 7
(1 1)	$(A_o \in G \wedge t_o \in A_o \Rightarrow B \in G)$	I 4
	Returns A_o/B for (1 1)	H 6.1, H 2
(1 2)	$(A_o \in G \wedge t_o \in A_o \longrightarrow t_o \in A_o)$	I 4.2
	Returns "T" for (1 2)	H 6.2, H 2
	Returns A_o/B for (1)	I 4.4

4. PEEKing and Forward Chaining

PEEK.

We saw on page 21 that when all else fails, we expand the definition of the conclusion C. Such is not the case for the hypothesis H. However, when proving $(B \longrightarrow C)$, the algorithm HOA sometimes "peeks" at the definition of B to see if it has the potential of helping with the proof of C, and if so it then (temporarily) expands that definition. This is done after a regular call to HOA has failed and the "peek light" has been turned on.

To facilitate this, the program has a PEEK property list for each of the main predicates. Table V gives some of its entries. This enables the program to quickly check whether an expansion of the definition of B would have a chance of helping with the proof.

Table V
PEEK Property Lists

1. (Oc [Open Cover])

2. (Reg [Subset Open Clsr])

etc.

Ex. 7. $(\text{Reg} \wedge \text{Oc } F \longrightarrow \exists G(\text{Cover } G))$

(1) $(\text{Reg} \wedge \text{Oc } F_o \Rightarrow \text{Cover } G)$ I 7

HOA is called at Step 12 of IMPLY and fails; then the PEEK light is turn ON.

(1) $(\text{Reg} \wedge \text{Oc } F_o \Rightarrow \text{Cover } G)$ I 11.2

(1 1) $(\text{Reg} \Rightarrow \text{Cover } G)$ NIL H 6

(1 2) $(\text{Oc } F_o \Rightarrow \text{Cover } G)$ H 6.2

$((\text{Open } F_o \wedge \text{Cover } F_o) \Rightarrow \text{Cover } G)$ H 2.2 (PEEK)
Table V, Entry 1.

F_o/G is returned for (1 2) and (1).

Notice that it did not expand the definition of Reg in (1 1), i.e.,

(1 1) $(\text{Reg} \Rightarrow \text{Cover } G)$

because in Rule 2 of Table V, "Reg" did not have "Cover" on its PEEK property list.

After such a use of PEEK, the expanded definition is not retained the original form Oc F_o is retained for any further proofs that may be required. This permits the proofs to proceed at a high level where possible, and resorting to expanded definitions only when necessary. It also facilitates human understanding when operated in a man-machine mode.

Forward Chaining.

In IMPLY Rule 7, when a new hypothesis is added to H we try to "forward chain" with it. Forward chaining is another name for modus ponens: If $P'\theta = P\theta$, then a hypothesis

$$P' \wedge (P \longrightarrow Q)$$

is converted into

$$P' \wedge (P \longrightarrow Q) \wedge Q\theta \quad .$$

Ex. 8. $\forall a(P(a) \wedge \forall x(P(x) \longrightarrow Q(x)) \longrightarrow Q(a))$

(1) $(\text{NIL} \Rightarrow (P(a_o) \wedge (P(x) \longrightarrow Q(x)) \longrightarrow Q(a_o)))$

$(P(a_o) \wedge (P(x) \longrightarrow Q(x)) \wedge Q(a_o) \Rightarrow Q(a_o))$ I 7, 7.1 forward chaining

Returns "T".

It should be noted that this is Example 3 which was proved earlier using Rule H 7 (Back-Chaining). Forward chaining is an option which is available to the user. In some instances he may want to control its use. For example, forward chain with $P(x_o)$ only when $P(x_o)$ is a <u>ground</u> formula, or forward chain with an atom $P(x)$ only when P is a member of a predescribed list. Limited forward chaining has been used in a powerful way by Bundy [37], Ballantyne and Bennett [38,39], Nevins [17], Reiter [18], Siklossy et al [36], and others.

<u>PEEK forward chaining</u>.

If $P'\theta = P\theta$, A has the definition $(P \longrightarrow Q)$ then a hypothesis

$$P' \wedge A$$

is converted into

$$P' \wedge A \wedge Q\theta$$

<u>Ex. 9</u>. $(A \subseteq B \wedge B \subseteq C \longrightarrow A \subseteq C)$

(1) $(A \subseteq B \wedge B \subseteq C \Rightarrow A \subseteq C)$ I 7

We have dropped the subscripts of A_o, B_o and C_o in this example.

$(A \subseteq B \wedge B \subseteq C \Rightarrow (t_o \in A \longrightarrow t_o \in C))$ I 12 Definition 2

$(A \subseteq B \wedge B \subseteq C \wedge t_o \in A \Rightarrow t_o \in C)$ I 7

$(A \subseteq B \wedge B \subseteq C \wedge t_o \in A \wedge t_o \in B \wedge t_o \in C \Rightarrow t_o \in C)$ I 7.2 PEEK forward chaining

Returns "T" .

In the above, $(t_o \in A)$ was PEEK forward chained into $(A \subseteq B)$ by expanding the definition of $(A \subseteq B)$ to

$$(t \in A \longrightarrow t \in B)$$

and matching $(t \in A)$ to $(t_o \in A)$ with t_o/t, getting $(t_o \in B)$ as a result. Then $(t_o \in B)$ was PEEK forward chained into $(B \subseteq C)$ getting $(t_o \in C)$. The program has a checking mechanism to prevent an infinite continuation in adverse cases.

<u>Ex. 9.</u> $(A \subseteq B \wedge \overline{B} \subseteq C \wedge \forall D \forall E (D \subseteq E \longrightarrow \overline{D} \subseteq \overline{E}) \longrightarrow \overline{A} \subseteq C)$

(1) $(A_o \subseteq B_o \wedge \overline{B}_o \subseteq C_o \wedge \overbrace{(D \subseteq E \longrightarrow \overline{D} \subseteq \overline{E})}^{\alpha} \longrightarrow \overline{A}_o \subseteq C_o)$

When Rule I 7 is applied it forward chains $(A_o \subseteq B_o)$ into α to get $(\overline{A}_o \subseteq \overline{B}_o)$. A control is used to prevent repeated use of α to get, $\overline{\overline{A}}_o \subseteq \overline{\overline{B}}_o$, etc.

(1) $(A_o \subseteq B_o \wedge \overline{B}_o \subseteq C_o \wedge \alpha \wedge \overline{A}_o \subseteq \overline{B}_o \Rightarrow \overline{A}_o \subseteq C_o)$ I 7

(" $\Rightarrow (t_o \in \overline{A}_o \longrightarrow t_o \in C_o))$ I 12, Definition 2

$(A_o \subseteq B_o \wedge \overline{B}_o \subseteq C_o \wedge \alpha \wedge \overline{A}_o \subseteq \overline{B}_o \wedge t_o \in \overline{A}_o \wedge t_o \in \overline{B}_o \wedge t_o \in C_o$

$\longrightarrow t_o \in C_o)$

In the above application of Rule I 7, $(t_o \in \overline{A}_o)$ was forward chained into $(\overline{A}_o \subseteq \overline{B}_o)$ to obtain $(t_o \in \overline{B}_o)$, which is turn was forward chained into $(\overline{B}_o \subseteq C_o)$ to obtain $(t_o \in C_o)$

(" $\wedge\ t_o \in C_o \longrightarrow t_o \in C_o)$ "T"

Ex. 9A. $(\mathrm{Oc}\ F \wedge \forall F\ \exists G(\mathrm{Oc}\ F \longrightarrow \mathrm{Cover}\ G \wedge \overline{G} \subseteq\subseteq F)$

$\longrightarrow \exists H(H \subseteq\subseteq F))$[13]

(1) $(\mathrm{Oc}\ F_o \wedge (\mathrm{Oc}\ F \longrightarrow \mathrm{Cover}\ G(F) \wedge \overline{G(F)} \subseteq\subseteq F) \longrightarrow H \subseteq\subseteq F_o)$

$(\mathrm{Oc}\ F_o \wedge (\mathrm{Oc}\ F \longrightarrow \mathrm{Cover}\ G(F) \wedge \overline{G(F)} \subseteq\subseteq F) \wedge \mathrm{Cover}\ G(F_o) \wedge \overline{G(F_o)} \subseteq\subseteq F_o$

$\Rightarrow H \subseteq\subseteq F_o)$ I 7
Forward chaining

Returns $\overline{G(F_o)}/H$.

Ex. 9B. $(\mathrm{Oc}\ F \wedge \mathrm{Reg} \longrightarrow \exists H(H \subseteq\subseteq F))$

(1) $(\mathrm{Oc}\ F_o \wedge \mathrm{Reg} \wedge \mathrm{Cover}\ G(F_o) \wedge \overline{G(F_o)} \subseteq\subseteq F_o \Rightarrow H \subseteq\subseteq F_o)$ I 7

Here $\mathrm{Oc}\ F_o$ has been PEEK Forward Chained into Reg which has the definition

$$\forall F\ \exists G(\mathrm{Oc}\ F \longrightarrow \mathrm{Cover}\ G \wedge \overline{G} \subseteq\subseteq F)$$

which has skolem form (in this case)

$$(\mathrm{Oc}\ F \longrightarrow \mathrm{Cover}\ G(F) \wedge \overline{G(F)} \subseteq\subseteq F).$$

As in the previous example $\overline{G(F_o)}/H$ is returned.

5. Conditional Rewriting and Conditional Procedures

Conditional Rewrite Rules.

In Section 3 we described the REDUCE feature which causes various formulas (or subformulas) to be rewritten. For example, the expression

$$t \in A \cap B$$

is rewritten as

$$(t \in A \wedge t \in B) .$$

Sometimes we wish such a conversion to be made only if a certain condition is satisfied. Such rules, are called "conditional rewrite rules", and are added to the REDUCE table in the form

(* P A B) .

The program upon detecting the *, checks the validity of P before rewriting B for A (with proper instantiation). If P is not true then A is not rewritten. The * is placed there to distinguish conditional rules from ordinary REDUCE rules. For example, the entry

(* A ≠ NULL NODES(A) NODES(LEFT(A)) + NODES(RIGHT(A)))

means that NODES(A)[†] can be "reduced" to NODES(LEFT(A)) + NODES(RIGHT(A)) if A ≠ NULL. The rewrite rule is not valid if A = NULL because LEFT(NULL) and RIGHT(NULL) are not defined, thus the rewrite rule is applicable only

[†]NODES(T) is one plus the number of nodes in a binary tree T. NODES(NULL) = 1
LEFT(T) is the left-hand son of T.

only if $A \neq$ NULL is known. Notice also that the result of the rewrite rule contains forms to which the rewrite rule could be applied. This would result in an infinite expansion normally but the condition on the rewrite rule precludes this. Generally this rule would be used once and then it would not be known if LEFT(A) $\neq$ NULL or if RIGHT(A) $\neq$ NULL so the rule would not be applied again.

Rewrite rules are expected to be applied quickly or not at all. Their power lies in the quickness with which they can be applied. Accordingly we avoid long drawn-out procedures for checking the validity of P. For example we do not call IMPLY itself to check P. Rather we have a "mini" version of IMPLY, for this purpose, which includes ANDS (See p. 15), which we call QK-IMPLY.

A similar remark can be made for conditional procedures described below.

Conditional Procedures.

Some procedures are conditional in that they are initiated only when certain conditions are satisfied. Examples of these are PAIRS described below, INDUCTION described on page 58 below and in [2], and the limit heuristic described in [3]. See also [40,29].

PAIRS.

Sometimes in HOA the expressions C and B will not unify even though the main predicates of C and B are the same. For example,

$$(G_o \subseteq\subseteq F_o \Rightarrow H_o \subseteq\subseteq J_o)^{13}.$$

In this case, at Step 3 of HOA, the algorithm consults the PAIRS property list of "$\subseteq \subseteq$" for advice. That property list may (or may not) list one or more subgoals that can be proved to establish the given goal. Table VI gives some such entries.

Table VI

PAIR Property Lists

1. $(\text{Cover } (\text{Cover } G \longrightarrow \text{Cover } F)[(G \subseteq\subseteq F) \ (\quad)\cdots])$

2. $(\subseteq\subseteq$[13] $(G \subseteq\subseteq F \longrightarrow H \subseteq\subseteq J)$

$[(H \subseteq\subseteq G \wedge F \subseteq\subseteq J)(\quad)\cdots])$

3. $(\text{Lf}$[14] $(\text{Lf } G \longrightarrow \text{Lf } F)[(F = \overline{G}) \qquad])$

4. $(\text{countable } (\text{countable } A \longrightarrow \text{countable } B)$

$[\exists f(f \text{ is a function} \wedge \text{domain } f \subseteq A \wedge B \subseteq \text{range } f)$

$(B \subseteq A)\cdots]$

etc.

---

[14] Lf G means that G is locally finite. That is, at any point x, there is an open set A which intersects only a finite number of members of G.

Ex. 10. $(G \subseteq\subseteq F \longrightarrow G \subseteq\subseteq \overline{F})$

(1) $(G_o \subseteq\subseteq F_o \Rightarrow G_o \subseteq\subseteq \overline{F}_o)$ I 7

$(G_o \subseteq\subseteq G_o) \wedge (F_o \subseteq\subseteq \overline{F}_o)$ H 2.3

PAIRS Entry 2

(1 1) $(G_o \subseteq\subseteq G_o)$

"T" I 5

Reduce Rule 21

(1 2) $(F_o \subseteq\subseteq \overline{F}_o)$

"T" I 5

Reduce Rule 22

Notice that the PAIRS Rule H 3 has converted the goal (1) into a subgoal that is easily proved by the REDUCE rules 21 and 22.

REDUCE and PAIRS act a lot alike in that they change one goal into another, the difference being that REDUCE acts on a "single entry" (i.e., a given formula is rewritten as another), while PAIRS acts on a double entry. However, that double entry requires that the two input formulas be partially matched (their main predicates are identical).

Such a pairs concept can be extended to include pairs of predicates that are not identical, but that has not been done for the present algorithms.

In general we favor procedure which are triggered by easy to check conditions.

<u>Ex. 11</u>. <u>Th</u>. (g is a function) $\wedge$ countable (domain g)

$\wedge$ $A \subseteq$ range g $\longrightarrow$ countable A

(1) $(g_o$ is a function) $\wedge$ countable (domain g_o)

$\wedge$ $A_o \subseteq$ range $g_o \Rightarrow$ countable A_o — I 7

countable (domain g_o) $\Rightarrow$ countable A_o — H 6.2

(1 P) $(g_o$ is a function) $\wedge$ $A_o \subseteq$ range g $\Rightarrow$ ((f is a function)

$\wedge$ (domain f $\subseteq$ domain g_o) $\wedge$ ($A_o \subseteq$ range f)) — PAIRS Entry 4

(1 P 1) $(g_o$ is a function) $\wedge$ $A_o \subseteq$ range $g_o \Rightarrow$ (f is a function)

g_o/f

(1 P 2) $(g_o$ is a function) $\wedge$ $A_o \subseteq$ range g_o

$\Rightarrow$ (domain $g_o \subseteq$ domain g_o) $\wedge$ ($A_o \subseteq$ range g_o)

(1 P2 1) (") $\Rightarrow$ (domain $g_o \subseteq$ domain g_o)

"T" by REDUCE Rule 23

(1 P2 2) $(g_o$ is a function) $\wedge$ $A_o \subseteq$ range $g_o \Longrightarrow A_o \subseteq$ range g_o.

"T"

So g_o/f is returned for (1 P) and for (1).

6. Complete Sets of Reductions

The use of rewrite rules as in our REDUCE procedure is a very powerful device. It is extremely more efficient than ordinary substitution of equals as is used in Paramodulation or in HOA Rules 9 and 7E, because the latter allows substitution both ways. Thus it is highly desirable to get as many entries as possible in the REDUCE table and to remove the corresponding equality units from the hypotheses.

The questions that naturally arise are: How far can you go with rewrite rules? Can such a system be made complete in some sense? How do we choose the entries for the REDUCE table? Can we generate all needed REDUCE table entries from a few key ones?

Very general, although incomplete, answers to these questions are given by a beautiful paper of Lankford [30] which is based on pioneering work of Knuth and Bendix [31] and some earlier work of Slagle [32].

The reader is referred to [30] for details but the general idea is that some theories, such as group theory, allow a "complete set of reductions." For example, there exists a set of entries for a REDUCE table which handles all equality substitutions for the equational axioms of group theory. A very powerful algorithm is given which often generates a complete set of reductions from the axioms of a given equational theory. One problem with the concept of the rewrite rule currently in vogue is that it does not allow commutative axioms to be included in a REDUCE table since, for example, the rewrite rule $x \cdot y \longrightarrow y \cdot x$ when applied to $a \cdot b$ produces the infinite sequence of rewrites $a \cdot b$, $b \cdot a$, $a \cdot b$, $b \cdot a, \ldots$. However, Lankford [30] has shown how commutative theories, such as commutative, groups, rings, Boolean algebras, and modules over rings, which allow no complete sets of reductions, can nevertheless be treated efficiently and in a complete way with most of the equality units in a REDUCE table. Earlier, Bledsoe, et al [3] used such a decision procedure for ring theory as the basis of a heuristic approximation of an unavailable decision procedure for field theory with encouraging results.

Table IV shows only a few of the REDUCE rules used by our prover, and many others can be easily added (see for example, ADD-REDUCE in Section 6). The largeness of the table does not impede the speed of its use because hash code techniques can be employed.

As pointed out earlier, the REDUCE table is a convenient place to store facts that may be needed at some point in a proof but which will never be accessed until actually needed. If these same facts were made part of the hypothesis they would greatly clutter up and slow down the operation of the prover.