THE UT ~~INTERACTIVE~~ Natural Deduction PROVER

by

W. W. Bledsoe

April 1983 ATP-17B

Table of Contents

List of Tables

Abstract

The theorem prover developed by our group at The University of Texas is described. Algorithms are given for its principal routine, IMPLY, and supporting routines.

The prover itself (without man-machine interaction) is a natural deduction system which uses the concepts of: subgoaling, reductions (rewrite rules), procedures, controlled definition instantiation, controlled forward chaining, conditional rewriting and conditional procedures, algebraic simplification, and induction.

It, or variations of it, have been used to prove theorems in set theory, intermediate analysis, and topology, theorems arising from program verification, and limit theorems of calculus.

This paper, ATP 17B, is a revison of the first part of ATP 17A, "The UT Interactive Prover", June 1978, and an earlier version, ATP-17, May 1975. The most significant change (in the first part of ATP 17A) is in the way in which the IMPLY table is presented -- In ATP 17A two routines, IMPLY and HOA, are described, whereas in this paper these have been combined into one routine, IMPLY. Also two new rules, 2 and 2', have been added, along with a "smart" backtracking procedure for AND-SPLITS; these have been part of our implemented program for a number of years; Rule 13-O is a recent addition. Also some of the sections and appendices have been omitted, with appropriate references to the same material in the literature. And various additional comments and examples have been added, and slight revisions have been made.

θ is derived which consists of these replacements. The skolemized formula E is then sent to the prover by a call IMPLY(NIL,E).

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

$$(H \dashrightarrow C)\theta$$

is propositionally valid. θ[1]is a most general such substitution. If no substitution is needed then IMPLY returns "T". It will return NIL if (H --> C) is not valid or if it cannot find a proof within a prescribed time limit. It uses supporting routines such as UNIFY to obtain the substitution θ.

The routine IMPLY is described in algorithmic form in Table 2-2. This table gives the basic rules of IMPLY: some additional details, which are mentioned in footnotes and in the later descriptions, have been implemented.

The skolemized formula E being proved is initially sent to IMPLY by a call, IMPLY(NIL,E). The rules are then processed in order, as listed; if none apply, NIL is returned.

(A first reading of the table should be be done ignoring the footnotes, to get a general idea; then a more careful reading can illuminate the details.)

[1]Actually the formula (APPLYθ(H -> C)) is valid, where APPLY is the function defined in Section 2.2.

Table 2-2: ALGORITHM IMPLY(H,C)

Rule No.	IF	RETURN
1.	$C = T$ or $H = FALSE$	T
2. ANCESTRY		
2.1	$\theta :=$ ANCESTOR(C) $\neq$ NIL	θ
2.2	HIGHER-GOAL-FAILURE(C) $\neq$ NIL	NIL
3. AND-SPLIT		
	$C \equiv (A \wedge B)$[2] $\theta :=$ IMPLY(H,A) $\neq$ NIL, B' := APPLY(θ,B), $\lambda :=$ IMPLY(H,B') $\neq$ NIL	COMPOSE(θ,λ)
4. CASES		
	$H \equiv (A \vee B)$, $\theta :=$ IMPLY(A,C) $\neq$ NIL, B' := APPLY(θ,B), $\lambda :=$ IMPLY(B',C) $\neq$ NIL	COMPOSE(θ,λ)
5. REDUCE (See Section 5)		
	H' := REDUCE(H) $\neq$ H, or C' := REDUCE(C) $\neq$ C,	IMPLY(H',C')
6. OR-FORK		
	$C \equiv (A \vee B)$, C' := AND-OUT(C) $\neq$ C	IMPLY(H,C')
	$\theta :=$ IMPLY(H $\wedge$ ~B, A)[3] $\neq$ NIL	θ
	ELSE	IMPLY(H $\wedge$ ~A, B)
7. PROMOTE		
	$C \equiv$ (A -> B)	IMPLY(H $\wedge$ A, B)[4]
8.	$C \equiv$ (A <--> B)	IMPLY(H, [(A->B) $\wedge$ (B->A)])
9.	$C \equiv (a = b)$, $\theta :=$ UNIFY(a,b) $\neq$ NIL	θ
10. FLIP-C		
	$C \equiv$ (~A)	IMPLY(H $\wedge$ A, NIL)

[2] By the expression "C = (A $\wedge$ B)" we mean that C has the form "A $\wedge$ B".

[3] In Rule 6 the "~" in (~B) is pushed to the inside ; e.g., ~(~P) goes to P, ~(P -> Q) goes to (P $\vee$ ~Q. If B contains no "~" or "-->" then (B) is omitted and the call is made to IMPLY(H,A). Similarly for the (~A) in IMPLY(H~A, B).

[4] Actually we call IMPLY(OR-OUT(H $\wedge$ A), AND-OUT(B)).

Table 2-2: continued

Rule No.	IF	RETURN
11. INEQUALITY[5]		
12. MATCH	H' is a conjunct of H, θ:= UNIFY(H',C) $\neq$ NIL	θ
13. BACK-CHAIN	(A -> B) is a conjunct of H, θ:= UNIFY(B,C) $\neq$ NIL, A':= APPLY(θ,A), λ:= IMPLY(H,A') $\neq$ NIL	COMPOSE(θ,λ)
	13-O. (A -> B) is a conjunct of H, B' is a disjunct of B, θ:= UNIFY(B',C) $\neq$ NIL, B":= DELETE(B',B), λ:= IMPLY(H, APPLY(θ,A $\wedge$ ~B")) $\neq$ NIL	COMPOSE(θ,λ)
	13->. [A -> (B -> D)] is a conjunct of H, H':= REPLACE([A -> (B -> D)], [A $\wedge$ B -> D], H)	IMPLY(H',C)
14. SUBSTITUTION OF EQUALS	a=b is a conjunct of H, z:= MINUS-ON(a,b) $\neq$ 0, z is a number	T
	H':= SUB=((a,b), H), C':= SUB=((a,b), C), θ:= IMPLY(H',C') $\neq$ NIL	θ
	14H. (A -> a=b) is a conjunct of H, H':= SUB=((a,b), H), C':= SUB=((a,b), C), θ:= IMPLY(H',C') $\neq$ NIL, A':= APPLY(θ,A), λ:= IMPLY(H,A') $\neq$ NIL	COMPOSE(θ,λ)
15. FLIP-H	~A is a conjunct of H, H':= DELETE(~A,H)	IMPLY(H',C $\vee$ A)
16. DEFINE-C (See Section 5)	C':= DEFINE(C) $\neq$ NIL	IMPLY(H,C')
17.	ELSE	NIL

When proving a theorem of the form

[5]See [7].

$$H \dashrightarrow A \wedge B$$

IMPLY uses Rule 3 to split it into the two subgoals

$$H \dashrightarrow A \text{ and } H \dashrightarrow B$$

which it tries to prove separately. It is (of course) necessary that the substitution θ derived for (H --> A) be applied to B , but not to all of (H --> B) in proving the second subgoal. The reader can see the necessity for the particular form of this rule by considering the three examples

- [P(a) ∧ Q(a) --> P(x) ∧ Q(x)],
- [P(a) ∧ Q(b) --> P(x) ∧ Q(x)],
- [P(x) --> P(a) ∧ P(b)].

2.2. Other algorithms.

- ANCESTRY. Suppose that C' appears as an ancestor goal of C, on the part of the proof tree between the root of the proof tree and the current goal. Then ANCESTOR(C) will return θ if C' unifies with (~ C) with mgu θ. And HIGHER-GOAL-FAILURE(C) will return T if C' is identical to C, for C is on the part of the proof tree since last something was added to H (as for example, in Rule 7: promote).
- UNIFY is the standard unification algorithm, except that UNIFY(x,y) returns "T" if x=y , and NIL if x and y are not unifiable. (However, it will not return a substitution which is contained in the list, EXCLUDE. See "Smart" backtracking, Section 4, below.)
- AND-OUT (OR-OUT) is an algorithm which puts an expression into conjunctive normal form (disjunctive normal form), but does not convert implications.

  Ex. AND-OUT(A ∨ [(P->Q) ∧ R]) = [A ∨ (P->Q)] ∧ [A ∨ R]

  Ex. OR-OUT([A ∨ (P->Q)] ∧ [A ∨ R]) = A ∨ [(P->Q) ∧ R]
- APPLY(θ,D) applies the substitution θ to the formula D. It returns D if θ= "T", and returns the ordinary instance Dθ (See P.75 of [7]) if θ has no conflict.[6].] If θ has two entries a/x, b/x, with a$\neq$b, then APPLY(θ,D) returns the conjunction of the two formulas APPLY(θ1,D) and APPLY(θ2,D), where $\theta 1$ = DELETE(a/x, θ), $\theta 2$ = DELETE(b/x, θ).

  Ex. If θ= (a/x, b/x, c/y), D = P(x,y,z), then APPLY(θ,D) = P(a,c,z) ∧P(b,c,z).
- COMPOSE(θ,λ) is defined as follows: If θ= "T" then λ; if λ = "T"then θ; else APPEND(θ,λ)

  Ex. If θ= (f(y)/x, b/x, c/z), λ = (d/y,e/y,g(u)/z), then COMPOSE(θ,λ) = (f(y)/x, b/x, c/z, d/y,e/y, g(u)/z).

  Thus if neither θ nor λ is "T", then COMPOSE(θ,λ) has the same effect, when applied to a formula, as does the ordinary composition $\theta \circ \lambda$, unless COMPOSE(θ,λ) has conflicting bindings.[7]
- DELETE(a,A) returns the result of removing the <u>first</u> occurrence of a from A.
- REPLACE(a,b,A) returns the result of replacing the first occurrence of a in A by b.

[6] A substitution θ has a conflict if it has two entries of the form a/x and b/x with a$\neq$b. Such substitutions are called "generalized substitutions", and are treated more fully in [8

[7] See footnote 6. .

- MINUS-ON(a,b) returns the result of algebraically simplifying the difference (a - b).

  Ex. MINUS-ON(x+5, x+y+2) = 3-y

- SUB= is a routine for substituting equals. Sub=((a,b),D) is defined as follows:
  Put a':= CHOOSE(a,b), b':= OTHER((a,b), a'), and
  Return D(a'/b'). (i.e, each b' in D is replaced by a'). CHOOSE is an algorithm used by SUB=. When CHOOSE(a,b) is called it selects one of {a,b} to replace the other. (In a current implementation, CHOOSE(a,b) returns a if a is a number, or b if b is a number, or b if b occurs in a, else a.)

- OTHER((a,b), a') is b if a'=a , else a.

2.3. Theorem Label

The third 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

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

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

$$(H \dashrightarrow A) \text{ and } (H \dashrightarrow B)$$

would have theorem labels (1 1) and (1 2) respectively. And two subgoals of (1 1) would have labels (1 1 1) and (1 1 2). Such theorem labels are exhibited in the left margin for the examples given in this paper. In addition to these 1's and 2's we also utilize other letters such as H, P, and =, to indicate other actions of the prover, and sometime use (1), (2), (3), etc., (i), to indicate that the current subgoal has been obtained from the last by the use of hypothesis number i.

3. Some Examples of Proofs by IMPLY

Ex. 1. (A --> A)

An initial call is made to

IMPLY(NIL, A -> A)

which in turn uses Rule 7 to call IMPLY(A,A) which returns "T" by Rule 12.

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). Thus the presentation of Ex. 1 becomes

(1)	$(NIL \Rightarrow (A \to A))$	
(1)	$(A \Rightarrow A)$	7 (indicating Rule 7)
	Returns "T".	12

Ex. 2.	$\forall a[\forall x\, P(x) \dashrightarrow P(a)\,]$.	
(1)	$(NIL \Rightarrow (P(x) \to P(a_o)))$	Skolemized
	(x is a skolem variable, and a_0 is a skolem constant.)	
	$(P(x) \Rightarrow P(a_0))$	7
	UNIFY(P(x),P(a_0)) returns a_0/x	12

Henceforth we will drop "NIL$\Rightarrow$" and write "A" instead of "NIL$\Rightarrow$A". Thus Ex. 2 becomes $\forall$a[$\forall$x P(x) --> P(a)]

(1)	$(P(x) \Rightarrow P(a_0))$	7
	Returns a_0/x.	12

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

(1)	$P(a) \wedge (\ P(x) \rightarrow Q(x)\ ) \Rightarrow Q(a)$	7
(1 (2))	$P(a) \wedge (\ P(x) \rightarrow Q(x)\ ) \Rightarrow P(a)$	13

(It has matched $Q(a)$ with $Q(x)$ getting a/x, and applied a/x to $P(x)$ to get the new goal $P(a)$. The (2) in the theorem label indicates that Hypothesis (2) was backchained upon, by Rule 13.)

Now it returns "T" for goal (1 (2)) and a/x for goal (1), because COMPOSE(a/x, "T") = a/x.

Ex. 4. $(A \vee B \longrightarrow A \vee B)$

(1)	$(A \vee B \Rightarrow A \vee B)$	7
(1 1)	$(A \Rightarrow A \vee B)$	4, CASES
	$(A \Rightarrow A)$	6, Footnote 4
	TRUE	12
(1 2)	$(B \Rightarrow A \vee B)$	
	$(B \Rightarrow A)$ Fails	6
	$(B \Rightarrow B)$ TRUE	12

Ex. 5. $(A \longrightarrow B \vee C)$ (Not a theorem).

(1)	$(A \Rightarrow B \vee C)$	7
	$(A \Rightarrow B)$ NIL	6
	$(A \Rightarrow C)$ NIL	
	Returns NIL (Failure).	

Ex. 6. $(A \wedge (\sim A \vee B) \longrightarrow B)$

(1)	$A \wedge (\sim A \vee B) \Rightarrow B$	7
	This becomes, by Footnote 4,	
	$(A \wedge \sim A) \vee (A \wedge B) \Rightarrow B$	
(1 1)	$(A \wedge \sim A) \Rightarrow B$	4
	$A \Rightarrow A \vee B$	15
	$A \Rightarrow A$ TRUE	6, 12
(1 2)	$A \wedge B \Rightarrow B$ TRUE	12

Ex. 7. $(\sim A \wedge B \longrightarrow \sim A)$

(1)	$(\sim A \wedge B \Rightarrow \sim A)$	7
	$(A \wedge \sim A \wedge B \Rightarrow \text{NIL})$	10
	$(A \wedge B \Rightarrow A)$ TRUE	15, 12

	IF	RETURN
Ex. 7'.	$(B->A) \wedge (C->B) \wedge (\sim A->C) \Rightarrow A$	
(1)	$(B->A) \wedge (C->B) \wedge (\sim A->C) \Rightarrow A$	7
	" $\Rightarrow B$	13
	" $\Rightarrow C$	13
	" $\Rightarrow \sim A$	13
	It returns T by Ancestry.	2.1
Ex. 8.	$(a=b \wedge P(a) ---> P(b)\ )$	
(1)	$a=b \wedge P(a) \Rightarrow P(b)$	7
	$CHOOSE(a,b) = a$	14
(1 (1))	$a=a \wedge P(a) \Rightarrow P(a)$	14
	$P(a) \Rightarrow P(a)$ TRUE	5, 12
EX. 9.	$\forall x\, P(x) ---> P(a) \wedge P(b)$	
(1)	$P(x) \Rightarrow P(a) \wedge P(b)$	7
(1 1)	$P(x) \Rightarrow P(a)$ a/x	3, 12
(1 2)	$P(x) \Rightarrow P(b)$ b/x	12
	So $(a/x, b/x)$ is returned for goal (1).	
Ex. 9'	$Q(a) \wedge Q(b) --> \exists x[(P(x) --> P(a) \wedge P(b)) \wedge Q(x)]$	
(1)	$Q(a) \wedge Q(b) \Rightarrow [(P(x) --> P(a) \wedge P(b)) \wedge Q(x)]$	7
(1 1)	$Q(a) \wedge Q(b) \Rightarrow (P(x) --> P(a) \wedge P(b))$	3
	$Q(a) \wedge Q(b) \wedge P(x) \Rightarrow P(a) \wedge P(b)$	7
(1 1 1)	$Q(a) \wedge Q(b) \wedge P(x) \Rightarrow P(a)$ a/x	3,12
(1 1 2)	$Q(a) \wedge Q(b) \wedge P(x) \Rightarrow P(b)$ b/x	12
	$\theta = (a/x\ b/x)$ is returned for (1 1).	
	Apply $(\theta, Q(x)) = Q(a) \wedge Q(b)$.	
(1 2)	$Q(a) \wedge Q(b) \Rightarrow Q(a) \wedge Q(b)$	
(1 2 1)	$Q(a) \wedge Q(b) \Rightarrow Q(a)$ TRUE	3,12
(1 2 2)	$Q(a) \wedge Q(b) \Rightarrow Q(b)$ TRUE	12
	$(a/x\ b/x$ is returned for (1).	
Ex. 10.	$(A \wedge B \wedge (A --> (B -> C)) ---> C)$	
(1)	$A \wedge B \wedge (A --> (B -> C)) \Rightarrow C$	7
	$A \wedge B \wedge (A \wedge B --> C) \Rightarrow C$	13->
(1 (3))	$A \wedge B \wedge (A \wedge B --> C) \Rightarrow A \wedge B$	13
	TRUE	3,12,12

Some more examples are given in Sections 4-7, and other more more substantial examples can be found in [3,1,10,9,7].

4. AND-SPLITS and "Smart" Backtracking

Instead of the AND-SPLIT rule shown in Rule 3 of Table 2.2, we have implemented a generalization of it shown below, which prevents certain kinds of trapping: (A similar procedure is used for Rule 4).

	IF	RETURN
3'. AND-SPLIT	$C \equiv (A \wedge B)$,	AND-C(C,NIL)

AND-C is a recursive function of two arguments, C and EXCLUDE. C is a conjunction, A $\wedge$ B, and EXCLUDE is a list of "illegal" substitutions, which the unification algorithm is forbidden to return. AND-C uses a function CONFLICT; CONFLICT(θ, λ) returns the set of conflicting substitutions from $\theta$and λ, which are causing the conflict. For example, if $\theta = \{ a/x \}$, $\lambda = \{ b/x \}$, with $a \neq b$, then CONFLICT(θ, λ) would return $\{ a/x \}$.

AND-C(C, EXCLUDE) is defined as follows:

IF	RETURN
$\theta :=$ IMPLY(H,A) $\neq$ NIL, B' := APPLY(θ,B), $\lambda :=$ IMPLY(H,B') $\neq$ NIL	COMPOSE(θ, λ)
$\theta \neq$ NIL, $\lambda =$ NIL, $\sigma :=$ IMPLY(H,B) $\neq$ NIL, L:=CONFLICT(θ, σ), EXCLUDE' := EXCLUDE $\cup$ L	AND-C(C, EXCLUDE')
ELSE	NIL

Ex. 11. P(a) $\wedge$ P(b) $\wedge$ Q(b) ---> $\exists$x [P(x) $\wedge$ Q(x)]

(We will abbreviate the presentation by just showing C instead of H$\Rightarrow$ C)

(1)	P(x) $\wedge$ Q(x)	7
	AND-C(P(x) $\wedge$ Q(x), NIL)	3'
(1 1)	P(x) $\theta = a/x$	12
(1 2)	Q(a) $\lambda =$ NIL	
(1 2')	Q(x) $\sigma = b/x$	12
	L = CONFLICT((a/x), (b/x)) = (a/x), EXCLUDE' = ((a/x)).	
	AND-C(P(x) $\wedge$ Q(x), ((a/x)))	
(1 1)	P(x) $\theta = b/x$	
	(a/x is rejected by UNIFY because (a/x) is in EXCLUDE)	
(1 2)	Q(b) TRUE	
	So b/x is returned for goal (1).	

Ex. 12. $P(a,c) \wedge P(b,d) \wedge P(a,e) \wedge Q(f,c) \wedge Q(a,e)$ ---> $\exists xy[P(x,y) \wedge Q(x,y)]$

```
(1)        P(x,y) ∧ Q(x,y)
           AND-C( P(x,y) ∧ Q(x,y), NIL)
(1 1)      P(x,y)    θ= a/x, c/y
(1 2)      Q(a,c)    λ = NIL
(1 2')     Q(x,y)    σ = f/x,c/y
           AND-C( P(x,y) ∧ Q(x,y), ((a/x,c/y)) )
(1 1)      P(x,y)    θ= b/x,d/y    ( (a/x,c/y) is rejected by UNIFY)
(1 2)      Q(b,d)    λ = NIL
(1 2')     Q(x,y)    σ = f/x,c/y
           AND-C( P(x,y) ∧ Q(x,y), ((a/x,c/y), (b/x,d/y)) )
(1 1)      P(x,y)    θ= a/x,e/y
(1 2)      Q(a,e)    λ = T
           So (a/x,e/y) is returned for goal (1).  QED.
```

Other examples proved by this implementation are:

Let $H = (P(a) \wedge P(b) \wedge P(c) \wedge R(c) \wedge R(d) \wedge R(e) \wedge Q(f,c) \wedge Q(b,e))$.

Ex. 13. H ---> $\exists xy\ [P(x) \wedge R(y) \wedge Q(x,y)]$

Ex. 14. H ---> $\exists xy\ [Q(x,y) \wedge P(x) \wedge R(y)]$

Ex. 15. H ---> $\exists xy\ [P(x) \wedge Q(x,y) \wedge R(y)]$

Ex. 16. $\exists x[\ P(x) \wedge Q(a) \wedge Q(b)$ ---> $P(a) \wedge P(b) \wedge Q(x)]$

Ex. 17. $\exists x[\ \{\ (Q(a) \wedge R(b) \text{ -> } P(x)) \wedge (Q'(c) \wedge R'(d) \text{ -> } P(x))$
$\wedge\ Q(x) \wedge R(x) \wedge Q'(x) \wedge R'(x) \wedge S(c) \wedge S(d)\ \}$ ----> $P(x) \wedge S(x)]$

The following is a theorem which is not proved by IMPLY. However, it too can be proved (see below) if the theorem is first "preprocessed" in a manner shown in Ex. 19' below.

Ex. 19. $\{\forall z[Q(z) \text{ -> } P(z)]$ ---> $\exists x[\ (P(x) \text{ -> } P(a)) \wedge (Q(x) \text{ -> } P(b))]\}$

```
(1)        [Q(z) -> P(z)]⇒ (P(x) -> P(a)) ∧ (Q(x) -> P(b))              7
           AND-C( (P(x) -> P(a)) ∧ (Q(x) -> P(b)), NIL)
(1 1)      (P(x) --> P(a))    θ= a/x                                    7,12
(1 2)      (Q(a) --> P(b))    λ = NIL
(1 2')     (Q(x) --> P(b))
           [Q(z) -> P(z)] ∧ Q(x)⇒ P(b)                                  7
           Succeeds with  σ = (b/z,b/x)
           AND-C( (P(x) -> P(a)) ∧ (Q(x) -> P(b)), ((a/x)) )
(1 1)      (P(x) --> P(a))    NIL
           NIL is returned for goal (1).
```

However, if Ex. 19 was first "preprocessed" to the form

Ex. 19'.

```
(1)      ([Q(z) -> P(z)] ∧ [P(x) ∨ Q(x)] ∧ [ P(a) -> Q(x)] ∧ [P(b) -> P(x)]
                  -----> P(a) ∧ P(b) ).
```

then it is easily proved by IMPLY as follows:

```
(1 1)      [Q(z) -> P(z)] ∧ [P(x) ∨ Q(x)]   ... ⇒  P(a)                       3
(1 1 1)          "          P(x)           ... ⇒  P(a)    a/x              4,12
(1 1 2)          "          Q(a)           ... ⇒  P(a)
(1 1 2 (1))            "            Q(a)           ... ⇒  Q(a)    a/z       13
           Returns (a/x,a/z) for (1 1).
(1 2)            "          [P(x) ∨ Q(x)]   ... ⇒  P(b)
(1 2 1)          "          P(x)           ... ⇒  P(b)    b/x              4,12
(1 2 2)          "          Q(b)           ... ⇒  P(b)
(1 2 2 (1))            "            Q(b)           ... ⇒  Q(b)    b/z       13
           Returns (b/x,b/z) for (1 2),  and (a/z,b/x,a/z,b/z) for goal (1).
```

It is true that many theorems do occur in a form like that of Ex. 19', (See [14, Chap 6]) but we much prefer not to force each of our theorems into that mode. We would rather have the prover fail to give a proof on a few weird examples like Ex 19, rather than to force them into a form that looks so different than the original. (Since we here are doing "Natural" deduction.)

5. Definitions and Reductions

5.1. DEFINE

Rule 16 of IMPLY calls DEFINE(C) which expands definitions from a stored list. Table 5-1 gives some such definitions.

When the defining form introduces quantifiers (e.g., Rule 2 of Table 5-1) it is necessary to eliminate these quantifiers by skolemization. We skolemize when the definition is expanded using variables occurring in the unexpanded form in the present theorem as free variables in the skolemization. The skolemization also depends on whether the formula occupies a positive[8] or negative position in the theorem being proved. For example $(A \subseteq B)$ is replaced by $(x_0 \in A \rightarrow x_0 \in B)$ in

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

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

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

[8]See Appendix 1.

Table 5-1: Some Definitions

	Formula Being Defined	Defining Form
1.	$(A = B)$[9]	$(A \subseteq B \wedge B \subseteq A)$
2.	$(A \subseteq B)$	$\forall x(x \in A \dashrightarrow x \in B)$
		Skolem form[10]
		$(x_0 \in A \rightarrow x_0 \in B)$ in "Conclusion"
		$(x \in A \rightarrow x \in B)$ in "Hypothesis"
3.	$(A \cup B)$	$\{x\colon x \in A \vee x \in B\}$
4.	$(A \cap B)$	$\{x\colon x \in A \wedge x \in B\}$
5.	$\bigcup_{t \in S} A(t)$	$\{x\colon \exists t(t \in S \wedge x \in A(t))\}$[11]
6.	$\bigcap_{t \in S} A(t)$	$\{x\colon \forall t(t \in S \rightarrow x \in A(t))\}$[12]
7.	subsets(A)	$\{x\colon x \subseteq A\}$
7'.	sb(A)	subsets(A)
8.	range f	$\{y\colon \exists x(y = f(x))\}$
9.	Oc F	(Open F $\wedge$ Cover F)

5.2. 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 5-2 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)), $(\emptyset \subseteq A)$.

---

[9]In the prover a different symbol is used for set equality to distinguish it from the arithmetic equality. Here in we mean set equality.

[10]When the defining form introduces quantifiers, two versions of its skolemization may result, depending on the position of the formula in the theorem. See page 10.

[11]See previous footnote.

[12]See previous footnote.

Table 5-2: 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.	$t\in \mathrm{subsets}(A)$	$t\subseteq A$
6.	$t\subseteq A \cap B$	$(t\subseteq A \wedge t\subseteq B)$
7.	$(A \cap A)$	A
8.	$(A \cup A)$	A
9.	$(A \cap \emptyset)$	0
10.	$(A \cup \emptyset)$	A
11.	$(\emptyset \subseteq A)$	"T"
12.	$A\in\{B\}$	$A = B$
13.	$(\text{range } \lambda x f(x))$	$\{y: \exists x(y = f(x)\}$
14.	$(\text{Choice } A \in A)$	$A \neq \emptyset$
15.	$(A \vee \sim A)$	"T"
16.	$(A \wedge \sim A)$	"FALSE"
17.	$(\text{"T"} \wedge A)$	A
18.	$(A \wedge \text{"T"})$	A
19.	$(A \vee \text{"T"})$	"T"
20.	$(\text{"T"} \vee A)$	"T"
21.	$(G \subseteq\subseteq G)$[13]	"T"
22.	$(G \subseteq\subseteq G)$[14]	"T"

[13] It need not concern the reader here but G is the set of closures of members of G. That is if A is the closure of the set A, then $G = \{A: A \in G\}$. 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.

[14] See footnote 13.

table 5-2: continued

INPUT	OUTPUT
23. $(A \subseteq A)$	"T"
24. $(A \subseteq A)$	"T"
25. A $\wedge$ FALSE	FALSE
26. FALSE $\wedge$ A	FALSE
27. A $\vee$ FALSE	A
28. FALSE $\vee$ A	A

etc.

Ex 21. $\forall A\ \forall B\ (A \subseteq A \cup B)$

(1)	$(A_o \subseteq A_o \cup B_o)$	
(1)	$(x_o \in A_o \text{-->} x_o \in (A_o \cup B_o))$	16
(1)	$(x_o \in A_o \text{-->} x_o \in A \vee x_o \in B_o)$	5
		REDUCE Rule 2
(1)	$(x_o \in A_o \Rightarrow x_o \in A_o \vee x_o \in B_o)$	7
(1 1)	$(x_o \in A_o \Rightarrow x_o \in A_o)$	6
(1 1)	"T"	12
	Return "T" for (1).	

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

$$A \subseteq A \cup B$$
$$(x \in A \text{-->} x \in (A \cup B))$$
$$(x \in A \text{-->} x \in A \vee x \in B)$$
TRUE.

Ex. 22. $\forall A\ \forall B$ (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)	$[sb(A \cap B) \subseteq sb(A) \cap sb(B)] \wedge [sb(A) \cap sb(B) \subseteq sb(A \cap B)]$	16
		Definition 1
(1 1)	$[sb(A \cap B) \subseteq sb(A) \cap sb(B)]$	3
	This is an AND-SPLIT	
(1 1)	$[t_o \in sb(A \cap B) \Rightarrow t_o \in (sb(A) \cap sb(B))]$	16
		Definition 2
(1 1)	$[t_o \subseteq A \cap B \Rightarrow t_o \in sb(A) \wedge t_o \in sb(B)]$	5
		REDUCE Rules 5,1
(1 1)	$[t_o \subseteq A \wedge t_o \subseteq B \Rightarrow t_o \subseteq A \wedge t_o \subseteq B]$	5
		REDUCE Rules 6,5
	Return "T" for (1 1)	3, 12
(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. $(\alpha = \beta <\text{--}> \forall t(t \in \alpha <\text{--}> t \in \beta))$
2. $(t \in A \cap B <\text{--}> t \in A \wedge t \in B)$
3. $(t \in \text{subsets } A <\text{--}> t \subseteq A)$
4. $(t \subseteq A \cap B <\text{--}> t \subseteq A \wedge t \subseteq B)$.

Rule 4 of Table 5-2 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 $\{x\colon\ P(x)\}$, in which case it (in effect) instantiates that definition and applies Rule 3. For example the expression

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

is reduced by Rule 4 of Table 5-2 and Rule 5 of Table 5-1, to

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

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

Ex. 23. $(A \in G \text{--}> A \subseteq \cup_{B \in G} B)$

(1)	$(A_o \in G \Rightarrow A_o \subseteq \cup_{B \in G} B$	7
(1)	$(A_o \in G \Rightarrow (t_o \in A_o \text{--}> t_o \in \cup_{B \in G} B))$	16
		Definition 2
(1)	$(A_o \in G \Rightarrow (t_o \in A_o \text{--}> B \in G \wedge t_o \in B)$	5
		REDUCE Rule 4, Definition 5
(1)	$(A_o \in G \wedge t_o \in A_o \Rightarrow B \in G \wedge t_o \in B)$	7
(1 1)	$(A_o \in G \wedge t_o \in A_o \Rightarrow B \in G)$	3
	Returns A_o/B for (1 1)	12
(1 2)	$(A_o \in G \wedge t_o \in A_o \text{--}> t_o \in A_o)$	
	Returns "T" for (1 2)	12
	Returns A_o/B for (1)	

6. PEEKing and Forward Chaining

6.1. PEEK

Note that by Rule 16 of IMPLY when all else fails, we expand the definition of the conclusion C. Such is not the case for the hypothesis H. However, when proving (H --> C), the algorithm IMPLY sometimes "peeks" at the definition of H 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 IMPLY 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 6-1 gives some of its entries. This enables the program to quickly check whether an expansion of the definition would have a chance of helping with the proof.

Table 6-1: PEEK Property Lists

1. (Oc [Open Cover])

2. (Reg [Subset Open Clsr])

etc.

Ex. 24. (Reg $\wedge$ Oc F --> $\exists$ G (Cover G))

(1) (Reg $\wedge$ Oc $F_0 \Rightarrow$ Cover G) 7
When IMPLY fails; the PEEK light is turned ON.
(1) (Reg $\wedge$ Oc $F_0 \Rightarrow$ Cover G)
(1 1) (Reg$\Rightarrow$ Cover G) NIL
(1 2) (Oc $F_0 \Rightarrow$ Cover G)
((Open F_0 $\wedge$ Cover F_0)$\Rightarrow$ Cover G) 12(PEEK)
Table 6-1, Entry 1.

F_0/G is returned for (1 2) and (1).
Notice that it did <u>not</u> expand the definition of Reg in (1 1), i.e.,
(1 1) (Reg$\Rightarrow$ Cover G)
because in Rule 2 of Table 6-1, "Reg" did <u>n</u>ot have "Cover"
on its PEEK property list.

After such a use of PEEK, the expanded definition is not retained. The original form Oc F_0 is retained for any further proofs that may be required. This permits the proofs to proceed at a high level where possible, resorting to expanded definitions only when necessary. It also facilitates human understanding when operated in man-machine mode.

6.2. Forward Chaining

In IMPLY Rule 7, when a new hypothesis is added to H we try to "forward chain" with it. (If we are using the forward chaining option.) Forward chaining is another name for <u>modus ponens</u>: If $P'\theta = P\theta$, then a hypothesis

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

is converted into

$$P' \wedge (P \to Q) \wedge Q\theta.$$

Ex. 25. $\forall$a(P(a) $\wedge$ $\forall$x(P(x) -> Q(x)) -> Q(a))

(1) (NIL$\Rightarrow$ (P(a_0) $\wedge$ (P(x) -> Q(x)) -> Q(a_0)))
(P(a_0) $\wedge$ (P(x) -> Q(x)) $\wedge$ Q(a_0) $\Rightarrow$ Q(a_0)) 7
forward chaining
Returns a_0/x.

It should be noted that this is Example 3 which was proved earlier using Rule 13 (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 others; see [6, Section 2.6] for references.

6.3. PEEK forward chaining

If $P'\theta = P\theta$ and A has the definition (P -> Q) then a hypothesis

$$P' \wedge A$$

is converted into

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

Ex. 26. $(A \subseteq B \wedge B \subseteq C \dashrightarrow A \subseteq C)$

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

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

$(A \subseteq B \wedge B \subseteq C \Rightarrow (t_o \in A \dashrightarrow t_o \in C))$ Definition 2

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

$(A \subseteq B \wedge B \subseteq C \wedge t_o \in A \wedge t_o \in B \wedge t_o \in C \Rightarrow t_o \in C)$ 12

PEEK forward chaining

Returns t_0/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

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

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.

Ex. 27. $(A \subseteq B \wedge B \subseteq C \wedge \underset{\alpha}{\forall D\, \forall E(D \subseteq E \dashrightarrow D \subseteq E)} \dashrightarrow A \subseteq C)$

(1) $(A_o \subseteq B_o \wedge B_o \subseteq C_o \wedge (D \subseteq E \dashrightarrow D \subseteq E) \Rightarrow A_o \subseteq C_o)$

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

Forward chaining returns A_o/D, B_o/E

(1) $(A_o \subseteq B_o \wedge B_o \subseteq C_o \wedge \alpha \wedge A_o \subseteq B_o \Rightarrow A_o \subseteq C_o)$ 7

$(\qquad " \qquad) \Rightarrow (t_o \in A_o \dashrightarrow t_o \in C_o)$ 16

Definition 2

$(A_o \subseteq B_o \wedge B_o \subseteq C_o \wedge \alpha \wedge A_o \subseteq B_o \wedge t_o \in A_o \wedge t_o \in B_o \wedge t_o \in C_o$
$\Rightarrow t_o \in C_o)$

In the above application of Rule 7, $(t_o \in A_o)$ was forward chained into $(A_o \subseteq B_o)$ to obtain $(t_o \in B_o)$, which in turn was forward chained into $(B_o \subseteq C_o)$ to obtain $(t_o \in C_o)$

$(\quad " \quad \wedge t_o \in C_o \Rightarrow t_o \in C_o)$ "T"

Ex. 28. $(\text{Oc } F \wedge \forall F\, \exists G\, (\text{Oc } F \dashrightarrow \text{Cover } G \wedge G \subseteq\subseteq F)$
$\dashrightarrow \exists H (H \subseteq\subseteq F))$[15]

(1) $(\text{Oc } F_o \wedge (\text{Oc } F \dashrightarrow \text{Cover } G(F) \wedge G(F) \subseteq\subseteq F) \dashrightarrow H \subseteq\subseteq F_o)$
$(\text{Oc } F_o \wedge (\text{Oc } F \dashrightarrow \text{Cover } G(F) \wedge G(F) \subseteq\subseteq F) \wedge \text{Cover } G(F_o) \wedge G(F_o) \subseteq\subseteq F_o$
$\Rightarrow H \subseteq\subseteq F_o)$ 7
Forward chaining
Returns $G(F_o)/H$, F_0/F.

Ex. 29. $(\text{Oc } F \wedge \text{Reg} \dashrightarrow \exists H (H \subseteq\subseteq F))$

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

Here Oc F_o has been PEEK forward chained into Reg which has the definition

$$(\forall F \exists G (\text{Oc } F \dashrightarrow \text{Cover } G \wedge G \subseteq\subseteq F))$$

which has skolem form (in this case)

$$(\text{Oc } F \dashrightarrow \text{Cover } G(F) \wedge G(F) \subseteq\subseteq F).$$

As in the previous example $G(F_o)/H$, F_0/F is returned.

7. Conditional Rewriting and Conditional Procedures

7.1. Conditional Rewrite Rules

In section 5 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 $\neq$ NULL NODES(A) NODES(LEFT(A)) + NODES(RIGHT(A)))

[15]See footnote 13 on page 13.

means that NODES(A)[16] can be "reduced" to NODES(LEFT(A)) + NODES(RIGHT(A)) if A $\neq$ 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 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 [0, p. 15]), which we call QK IMPLY.

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

7.2. 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 in [2], and the limit heuristic described in [3].

7.3. PAIRS

Sometimes in Rule 12 the expressions C and H' will not unify even though the main predicates of C and H' are the same. For example,

$$(G_o \subseteq\subseteq F_o \Rightarrow H_o \subseteq\subseteq J_o).^{17}$$

In this case, at Rule 12 of IMPLY 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 7-1 gives some such entries.

[16] 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.

[17] See footnote 13 on page 13.

Table 7-1: PAIR Property Lists

1. (Cover (Cover G --> Cover F)[(G $\subseteq\subseteq$ F) ( )...])

2. ($\subseteq\subseteq$[18] (G $\subseteq\subseteq$ F --> H $\subseteq\subseteq$ J)

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

3. (Lf[19] (Lf G --> Lf F)[(F = G)])

4. (countable (countable A --> countable B)

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

   (B$\subseteq$A)...]

   etc.

Ex. 30.	(G $\subseteq\subseteq$ F --> G $\subseteq\subseteq$ F)	
(1)	($G_o \subseteq\subseteq F_o \Rightarrow G_o \subseteq\subseteq F_o$)	7
	($G_o \subseteq\subseteq G_o$) $\wedge$ ($F_o \subseteq\subseteq F_o$)	12, PAIRS Entry 2
(1 1)	($G_o \subseteq\subseteq G_o$)	5, Reduce Rule 21
	"T"	
(1 2)	($F_o \subseteq\subseteq F_o$)	5,Reduce Rule 22
	"T"	

Notice that the PAIRS Rule 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 procedures which are triggered by easy-to-check conditions.

```
Ex. 31.  Th.(g is a function) ∧ countable (domain g)
            ∧ A⊆range g --> countable A
```

[18]See footnote 13 on page 13.

[19]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.

(1) (g_o is a function) $\wedge$ countable (domain g_o)
$\wedge$ $A_o \subseteq$ range $g_o \Rightarrow$ countable A_o
When an attempt is made to match countable (domain g_o) with countable A_o, a partial match results which triggers Entry 4 of the PAIRS table.

(1 P) (g_o is a function) $\wedge$ $A_o \subseteq$ range $g_o \Rightarrow$ ((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 P 2 1)(") $\Rightarrow$ (domain $g_o \subseteq$ domain g_o)
"T" REDUCE Rule 23)

(1 P 2 2)(g_o is a function) $\wedge$ $A_o \subseteq$ range $g_o \Rightarrow$ $A_o \subseteq$ range g_o.
"T"
So g_o/f is returned for (1 P) and for (1).

8. Some Advantages of Natural Deduction Provers

The prover is a "Natural" type system in that it attempts to find the proof of a theorem in a way not unlike that of a human. It is not a refutation system as is Resolution.

We do not see Natural Deduction Provers as replacements for Resolution based systems; at worst, we would expect that every good prover would have a resolution style prover as a subprogram.

Some advantages of Natural Deduction Provers are:

- The proof development is easier for the human to follow. This is especially important when used in the interactive mode.
- It is easier for the human to add to the program: effective heuristics, Reducers (rewrite rules), Algebraic simplifiers, controlled definition instantiation, induction routines, etc.
- Formulas are not "torn apart" by clausing, making processes like induction [2] and higher order instantiation [10] easier.
- This mode facilitates the building and using of an "Agenda" mechanism, whereby the program can be made to pursue a line of attack which seems the most "promising" at the moment, drop it later for a more promising one, and come back to it later if necessary.

The algorithm IMPLY, and its supporting subroutines, form a natural deduction type system. It is like a Gentzen system [22], 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, --->, is retained, and we believe that the proof proceeds in a manner that would be natural to a mathematician.

9. Relation to the Work of Others

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 books by Chang and Lee [13] and Loveland [14] for information and references on resolution type systems, and to the work of

Allen and Luckham [15], Guard, et al [16], and Huet [17], on interactive provers. Our prover is in the spirit of Newell, Simon, Shaw [19], Gelernter , and has much in common with the work of Gentzen [22], Nevins, Reiter, Ernst [19], Bibel [20], Brown, Hewitt, McDermott and Sussman, Wang, Maslov, Rulifson, et al, Loveland [14, Ch. 6], and Plaisted. See [0] for these references.

I. Appendix I

<u>Skolemization -- Elimination of Quantifiers</u>

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. Similarly,

$$Q \dashrightarrow \exists x\ P(x) \text{ and}$$
$$\exists x\ (Q \dashrightarrow P(x)),$$

are skolemized as

$$Q \dashrightarrow P(x) \text{ and}$$
$$(\forall x\ P(x) \dashrightarrow C)$$

is skolemized as

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

where x is a skolem variable.

On the other hand, the formulas

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

are skolemized as

$$P(x_o)\ ,$$
$$(Q \dashrightarrow P(x_o), \text{ and}$$
$$(P(x_o) \dashrightarrow C)\ ,$$

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

Finally

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

is skolemized as

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

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, leaving each logical symbol except ∀ and ∃ in its original position.

We now give the general rules.

Given a formula E, we recursively define[20] as "positive" or "negative" the subformulas of E, as follows:

1. E is positive,

[20] See Wang [21].

2. If (A ∧ B) is positive (negative) then so are A and B,
3. If (A ∨ B) is positive (negative) then so are A and B,
4. If ~A is positive (negative) then A is negative (positive),
5. If (A --> B) is positive (negative) then
   A is negative (positive), and
   B is a positive (negative),
6. If (∀x A) is positive (negative) then
   A is positive (negative), and
   ∀ is a positive (negative) quantifier,
7. If (∃x A) is positive (negative) then
   A is positive (negative), and
   ∃ is a negative (positive) quantifier.

For example if E is the formula

$$([H \to (C \to \sim D)] \to [\sim A \lor (B \to F)])$$

then E, [~A ∨ (B --> F)], ~A, (B --> F), F, H, C and D are positive, while [H --> (C --> ~D)], (C --> ~D), ~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 varible v bound by a positive quantifier with the skolem expression $g(x_1, ..., x_n)$ where g is a new function symbol (a "skolem function" symbol) and x_1, ..., 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 ∃ is the negative quantifier, ∀ is a positive quantifier, and

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

is the skolem form of E, whereas the formulas

$$\forall x(P(x) \to \exists y\ Q(x,y)) \text{ and}$$
$$\exists x(\forall y\ \exists z\ P(x,y,z) \to \forall w\ Q(x,w))$$

have the skolem forms

$$(P(x_o) \to Q(x_o,y)) \text{ and}$$
$$(P(x,y,g(x,y)) \to Q(x,h(x)))$$

respectively, and the formula

$$\forall x(\forall y[\exists z\ H(x,y,z) --$$

∃u(C(x,u) --> ~∀v D(u,v))]>

--

∀w[~A(x,w) ∨ ∃s(∃t B(s,t) --> ∀r F(x,r,s,w))]> has skolem form

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

It should be noted (by those familiar with Resolution proofs) that the formula E is not first negated before the skolem form is derived. This difference reverses the roles of ∀ and ∃ in the skolemization process.

References

0. Bledsoe, W. W., and Tyson, Mabry, The UT Interactive Prover. University of Texas at Austin, Math Department Memo ATP 17A, June 1978.

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

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

4. Bruell, P., A description of the functions of the man-machine topology theorem prover. The University of Texas at Austin, Math Department Memo ATP-8, 1973.

5. Bledsoe, W. W., The sup-inf method in Presburger arithmetic. The University of Texas at Austin, Math Department Memo ATP-18. December 1974. Essentially the same as: A new method for proving certain Presburger formulas. Fourth IJCAI, Tblisi, USSR, September 3-8, 1975.

6. Bledsoe, W. W., Non-Resolution Theorem Proving. AI Journal (9) pp. 1-35, 1977.

7. Bledsoe, W. W., Bruell, P., and Shostak, Robert, A prover for general inequalities. The University of Texas at Austin, Math Department Memo ATP-40A, June 1978.

8. Tyson, Mabry, and Bledsoe, W. W., Conflicting and Generalized Substitution. Fourth Work Shop on Automatic Deduction, Austin, Texas, February 1-3, 1979, 103, pp. 14-18.

9. Ballantyne, A. M., and Bledsoe, W. W., Automatic proofs of theorems in analysis using non-standard techniques. J. ACM 24, p. 353-374, 1977.

10. Bledsoe, W. W., A Maximal Method for Set Variables in Automatic Theorem Proving. Machine Intelligence 9, p. 53-100, 1979.

11. Bledsoe, W. W., Some Automatic Proofs in Analysis. The University of Texas at Austin, Math Department Memo ATP-71, December 1982. (Presented at the Special Session on ATP, AMS meeting, Denver, Colorado, January 1983. To be published in the Proceedings of that Special Session.)

12. Good, D. I., London, R. L., and Bledsoe, W. W., 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.

13. Chang, C., and Lee, R. C., Symbolic logic and mechanical theorem proving. Academic Press, 1973.

14. Loveland, Donald, Automated theorem proving; a logical basis. North Holland, 1978.

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

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

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

18. Newell, A., Shaw, J. C., and Simon, H. A., Empirical explorations of the logic theory machine: a case study in heuristics. RAND Corp. Memo P-951, February 28, 1957. Proc. Western Joint Computer Conf. 1956, 218-239. Computers and Thought, Feigenbaum and Feldman (Eds) 134-152.

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

20. Bibel, Wolfgang, Automated Theorem Proving. Vieweg, 1982.

21. Wang, Hao, Towards Mechanical Mathematics, IBM J. Res. Dev., 4, 2-22.

22. Gentzen, G., Untersuchungen wher das logische Schliessen I. Mathematic. Zeitschrift 39, 176-210 (1935).