It can be seen from these examples that the new TYPELIST TY' which is returned as

$$(\theta \ TY')$$

represents the <u>cases that have not been proved</u> by this call to IMPLY or HOA. Thus it represents cases which are still to be proved by further calls to IMPLY. As long as TY' is not NIL in the returned (θ TY'), then the theorem has not been completely proved. Hence the final return from IMPLY (for the original theorem itself) must be of the form

$$(\theta \ NIL) \ .$$

Else the theorem is considered not to be proved.

<u>Ex. 10</u>. $\forall k(k \leq 2 \longrightarrow A[k] \leq A[k+1])$

$\wedge \forall m(3 \leq m \leq 7 \longrightarrow A[m] \leq A[m+1])$

$\wedge \forall n(6 \leq n \leq j \longrightarrow A[n] \leq A[n+1])$

$\longrightarrow \forall K(K \leq j \longrightarrow A[K] \leq A[K+1])$

(1) $(k \overset{\alpha}{\leq} 2 \longrightarrow A[k] \leq A[k+1])$

$\wedge (3 \leq \overset{\beta}{m} \leq 7 \longrightarrow A[m] \leq A[m+1])$

$\wedge (6 \leq \overset{\gamma}{n} \leq j \longrightarrow A[n] \leq A[n+1])$

$\longrightarrow K \leq j \longrightarrow A[K] \leq A[K+1]$

(1) $(\{K\colon\ 0\ j\}\{j\colon\ K\ \infty\} \wedge \alpha \wedge \beta \wedge \gamma \Rightarrow A[K] \leq A[K+1])$ I 7

(1 h1) $(\alpha \longrightarrow A[K] \leq A[K+1])$ K/k H 6

(1 h1 H) $(\{K\colon\ 0\ j\}\{j\colon\ K\ \infty\} \wedge \alpha \wedge \beta \wedge \gamma \Rightarrow K \leq 2)$ H 7, 7.2

SET-TYPE $(\sim(K \leq 2))$, $3 \leq K$ I 11

$TY' = [\{K\colon\ 3\ j\}\{j\colon\ K\ \infty\}]$, has no contradiction

Returns (T TY') I 11.3

Returns (K/k TY') for (1 h1).

(1 h2) $(TY' \wedge (\beta \wedge \gamma) \Rightarrow A[K] \leq A[K+1])$ H 6.4

(1 h2 h1) $(\beta \Rightarrow A[K] \leq A[K+1])$ K/m H 6

(1 h2 h1 H) $(TY' \wedge \beta \wedge \gamma \Rightarrow 3 \leq K \wedge K \leq 7)$ H 7, 7.2

(1 h2 h1 H1) $(TY' \wedge (\beta \wedge \gamma) \Rightarrow 3 \leq K)$ I 4

SET-TYPE $(\sim(3 \leq K))$, $K \leq 2$ I 11

$TY' = [\{K\colon\ 3\ \min(2,j)\}]$, has a contradiction

Returns (T NIL) I 11.1

(1 h2 h1 H2)	(TY' $\wedge$ $(\beta \wedge \gamma) \Rightarrow K \leq 7$)	I 4.2
	SET-TYPE($\sim(K \leq 7)$), $8 \leq K$	I 11
	TY'' = [{K: 8 j}{j: K ∞}], has no contradiction	
	Returns (T TY'')	I 11.3
	Returns (T TY'') for (1 h2 h1 H)	I 4.4
	Returns (K/m TY'') for (1 h2 h1)	H 7.4
(1 h2 h2)	(TY'' $\wedge$ $\gamma \Rightarrow A[K] \leq A[K+1]$) K/n	H 6.4
(1 h2 h2 H)	(TY'' $\wedge$ $\gamma \Rightarrow 6 \leq K \wedge K \leq j$)	H 7, 7.2
(1 h2 h2 H1)	(TY'' $\wedge$ $\gamma \Rightarrow 6 \leq K$)	I 4
	SET-TYPE($\sim(6 \leq K)$), $K \leq 5$	I 11
	TY'' = [{K min(5,j)}{j: K ∞}], has a contradiction	
	Returns (T NIL)	I 11.1
(1 h2 h2 H2)	(TY'' $\wedge$ $\gamma \Rightarrow K \leq j$)	I 4.2
	SET-TYPE($\sim(K \leq j)$), $j+1 \leq K$	I 11
	TY' = [{K: max(8,j+1)j}{j: K K-1}], has a contradiction	
	Returns (T NIL)	I 11.1
	Returns (T NIL) for (1 h2 h2 H)	I 4.4
	Returns (K/n NIL) for (1 h2 h2)	H 7.4
	Returns ({K/m, K/n} NIL) for (1 h2)	H 6.6
	Returns ({K/k, K/m, K/n} NIL) for (1)	H 6.6

The theorem is proved.

Simplification.

The prover utilizes a simplification routine to manipulate algebraic expressions. Its chief function is to put such expressions in canonical form. See [7, p. 27]. Many such simplifiers have been programmed [14, 10, 3, 11, etc.].

Such a routine is crucial in our program for handling TYPELIST and proving assertions about inequalities, because it eliminates the need for adding the field axioms for the real numbers.

Algebraic Unification.

If k is a skolem variable and b a constant, an ordinary unification algorithm will fail to unify the two expressions: $k+2$, and $b+5$.

We have augmented our algorithm to handle such arithmetic expressions. In this case the expressions are subtracted and simplified, and then solved for a variable, getting successively: $k+2-(b+5) = 0$, $k-b-3 = 0$

$$k = (b+3) .$$

Thus $(b+3)/k$ is returned for UNIFY $(k+2,\ b+5)$.

Similarly, the two expressions,

$$B[k+1] = \mathrm{Amax}(B,\ j,\ k+1) \quad ,$$

$$A_o[i_o] = \mathrm{Amax}(A_o,\ 1,\ i_o) \quad ,$$

where B, j, k are variables and A_o, i_o are constants, are unified as follows: (we show this in the prefix form).

(UNIFY(= (Array B (+ k 1))(Amax B j (k+1)))
      (= (Array A_o i_o)(Amax A_o 1 i_o)))

(UNIFY (Array B (+ k 1))
      (Array A_o i_o))

(UNIFY B A_o) , A_o/B

(UNIFY (+ k 1) i_o) It deduces that

(+ k (+ ($-i_o$) 1)) = 0, and returns the substitution

(+ i_o -1)/k

UNIFY Amax(A_o, j, i_o)
      Amax(A_o, 1, i_o) 1/j

Returns {A_o/B, (i_o - 1)/k, 1/j}.

The routine also handles such examples as

UNIFY(A[i_o] + A[j] , A[i] + A[j_o]) , Easy

UNIFY(A[i_o] + A[j] , A[j_o] + A[i_o])

In this last example, even though a canonical form is used there is no assurance that

i_o preceeds j_o

in the canonical ordering, even though i_o preceeds j. Hence the last example and those like it can present problems.

4. A Program Verification System

The interactive prover described in [1] has been augmented by the features described above in Sections 1-3, and used as part of a program verification system [9]. This system is running on the PDP-10 in London's group at the Information Sciences Institute, Marina Del Rey, California, and on the CDC 6600 and the PDP-10 in Good's group at The University of Texas at Austin.

The version at ISI has been augmented extensively by Larry Fagan and Peter Bruell, especially with features to facilitate man-machine interaction.

Both versions are coded in approximately 200 functions in LISP. Two additional sybsystems, INFPRINT and XEVAL, are used to augment the prover. INFPRINT is a routine which was coded by Don Lynn at ISI, and which takes an expression in LISP prefix notation and prints it out in (more readable) infix form, with appropriate identation. XEVAL which was developed at ISI by Don Good, is a simplification package for handling arithmetic expression, and also includes the rewrite rules of REDUCE described in [1] (Table IV). Since the combined code of these programs exceeds the allowed core space for the time-sharing system at UT, a version of UT-LISP has been developed by Mabry Tyson at UT which utilized virtual memory for LISP functions.

Appendix 3 is an example of output from the ISI program.

5. TYPELIST in RESOLUTION

The typing and proof by cases procedures described above can also be incorporated into RESOLUTION provers if an additional rule is added to resolution, and if the algorithms for simplification, set-type, sup and inf are included. Also a new algorithm INTERSECT is needed which combines two typelists (see examples below).

Before the start of resolution, after the theorem has been put into clausal form, each literal of the form

$$(a \leq b)$$

is converted to a TYPELIST by the algorithm SET-TYPE. Literals of the form

$$\sim(a \leq b)$$

are first transformed to $(b+1 \leq a)$ before being converted. Thus the new clauses will consist of ordinary literals L and typelist literals T. For example, the theorem

$$(x \leq 5 \wedge (x \leq 1 \rightarrow C) \wedge (2 \leq x \wedge x \leq 7 \rightarrow C) \rightarrow C)$$

is first converted to ordinary clausal form

1. $(x_o \leq 5)$
2. $(\sim(x_o \leq 1) \vee C)$
3. $(\sim(2 \leq x_o) \vee \sim(x_o \leq 7) \vee C)$
4. $\sim C$,

and then converted by SET-TYPE to

1. $\{x_o\colon 0 \quad 5\}$

2. $\{x_o\colon 2 \quad \infty\} \vee C$

3. $\{x_o\colon 0 \quad 1\} \vee \{x_o\colon 8 \quad \infty\} \vee C$

4. $\sim C$

Ordinary resolution is performed on non typelist literals. <u>Any</u> two typelist literals T_1 and T_2 are resolved, by calling

$$\mathrm{INTERSECT}(T_1, T_2) \ .$$

The result is another typelist which is included as a literal of the resolvent. If this resultant typelist contains a contradiction it is eliminated. For example clauses 1 and 2 above can be resolved on their first literals. Since

$$\mathrm{INTERSECT}(\{x_o\colon 0 \quad 5\}, \{x_o\colon 2 \quad \infty\} = \{x_o\colon 2 \quad 5\},$$

the resolvent of 1 and 2 is

5. $\{x_o\colon 2 \quad 5\} \vee C.$

Similarly we get

6.	$\{x_o\colon 2 \quad 5\}$	5, 4
7.	$\{x_o\colon 0 \quad 1\} \vee \{x_o\colon 8 \quad \infty\}$	3, 4
8.	$\cancel{\{x_o\colon 2 \quad 1\}} \vee \{x_o\colon 8 \quad \infty\}$	6, 7
9.	$\cancel{\{x_o\colon 8 \quad 5\}}$ or $\square$	8, 6 .

Since $\{x_o\colon 2 \quad 1\}$ and $\{x_o\colon 8 \quad 5\}$ contained contradictions they were eliminated. The algorithms SUP and INF are used for this purpose, exactly as described in Section 1. Here, for $\{x_o\colon 2 \quad 1\}$,

$$\mathrm{SUP}(x_o, \ \mathrm{NIL}) = 1$$

$$\mathrm{INF}(x_o, \ \mathrm{NIL}) = 2 \ .$$

Since [2,1] contains no integer we have a contradiction.

The algorithm INTERSECT when applied to type lists

$$(\{x_1: a_1 \quad b_1\} \ \{x_2: a_2 \quad b_2\} \cdots \{x_n: a_n \quad b_n\}) ,$$

$$(\{x_1: c_1 \quad d_1\} \ \{x_2: c_2 \quad d_2\} \cdots \{x_n: c_n \quad d_n)\} ,$$

simply intersects the corresponding entries, getting

$$(\{x_1: e_1 \quad f_1\} \ \{x_2: e_2 \quad f_2\} \cdots \{x_n: e_n \quad f_n\}) ,$$

where $e_i = \max(a_i, c_i)$ and $f_i = \min(b_i, d_i)$.

Consider now Example 10, of Section 3.

$$(\forall k(k \leq 2 \longrightarrow A[k] \leq A[k+1])$$
$$\forall m(3 \leq m \wedge m \leq 7 \longrightarrow A[m] \leq A[m+1])$$
$$\forall n(6 \leq n \wedge n \leq j \longrightarrow A[n] \leq A[n+1])$$
$$\longrightarrow \forall K(K \leq j \longrightarrow A[K] \leq A[K+1])) .$$

The ordinary clausal form is

1. $\sim(k \leq 2) \vee A[k] \leq A[k+1]$
2. $\sim(3 \leq m) \vee \sim(m \leq 7) \vee A[m] \leq A[m+1]$
3. $\sim(6 \leq n) \vee \sim(n \leq j_o) \vee A[n] \leq A[n+1]$
4. $K_o \leq j_o$
5. $\sim(A[K_o] \leq A[K_o+1])$,

where K_o and j_o are skolem constants, and k, m and n are variables.

The clauses are converted to

1. $\{k:\ 3\ \ \infty\} \vee A[k] \leq A[k+1]$

2. $\{m:\ 0\ \ 2\} \vee \{m:\ 8\ \ \infty\} \vee A[m] \leq A[m+1]$

3. $\{n:\ 0\ \ 5\} \vee [\{n:\ j_o+1\ \ \infty\}\{j_o:\ 0\ \ n-1\}] \vee A[n] \leq A[n+1]$

4. $(\{K_o:\ 0\ \ j_o\}\ \ \{j_o:\ K_o\ \ \infty\})$

5. $\sim(A[K_o] \leq A[K_o+1])$

Some of the resolvents of 1-5 are

6.	$\{K_o:\ 3\ \ \infty\}$	1, 5
7.	$\{K_o:\ 0\ \ 2\} \vee \{K_o:\ 8\ \ \infty\}$	2, 5
8.	$\{K_o:\ 0\ \ 5\} \vee [\{K_o:\ j_o+1\ \ \infty\}\{j_o:\ 0\ \ K_o-1\}]$	3, 5
9.	$\{K_o:\ 3\ \ 2\} \vee \{K_o:\ 8\ \ \infty\}$	6, 7
10.	$\{K_o:\ 8\ \ 5\} \vee [\{K_o:\ j_o+1\ \ \infty\}\{j_o:\ 0\ \ K_o-1\}]$	8, 9
11.	$(\{K_o:\ j_o+1\ \ j_o\}\ \ \{j_o:\ K_o\ \ K_o-1\})$ or $\square$	10, 4

In each of 9, 10, and 11, a typelist was removed which had a contradiction.

In the above example we did <u>not</u> convert the formula $A[k] \leq A[k+1]$ to typelist form

$$\{A[k]:\ 0\ \ A[k+1]\}\ .$$

This is controlled in the program by having a list $(j_o\ \ K_o\ \ k\ \ m\ \ n)$ of those variables and skolem constants which we allow to be typed.

One could allow <u>all</u> inequalities to be converted, but in that case a mechanism would need to be provided for unifying expressions when two type-list literals are resolved.

Appendix 1

Tables I and II listed below are lifted from Section 2 of [1]. They define IMPLY and HOA, the principal algorithms of the interactive prover described in [1]. The reader is referred to Section 2 of [1] for a full description of them and their use, and several examples.

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 Sections 1 and 2.

[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 [17] of [1].

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

[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 $\theta = (a/x\ b/y)$, $\lambda = (c/x\ d/z)$ 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 θ: = UNIFY(A,B)	
9.1	$\theta \not\equiv$ NIL		θ
9.2	$\theta \equiv$ NIL	Go To 10	
10.	$C \equiv (\sim A)$		IMPLY$(H \wedge A, \text{NIL})$
11.	INEQUALITY*		
12.	(call HOA)	Put θ: = HOA(H,C)	
12.1	$\theta \not\equiv$ NIL		θ
12.2	(PEEK) $\theta \equiv$ NIL	Put PEEK[7] light "ON" Put θ: = HOA(H,C)	
12.3	$\theta \not\equiv$ NIL		θ
12.4	$\theta \equiv$ NIL	Go To 13	

---

[6]Actually we call IMPLY(OR-OUT $(H \wedge A)$, AND-OUT(B)). See p. 13 of [1].

[7]See p. 26 of [1]. 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,)		
15.	ELSE		NIL

Table II

ALGORITHM

HOA(B,C)

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

[8] In Step 4.2, the "~" in (~D) is pushed to the inside; e.g., ~(~P) goes to P, and ~(P⟶Q) goes to $P \wedge \sim Q$. If D contains no "~" or "⟶" then (~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 $\theta := \text{ANDS}(D,C)^{*}$	
7.1	$\theta \equiv$ NIL	Go To 7E	
7.2	$\theta \not\equiv$ NIL	Put $\lambda := \text{IMPLY}(H, A\theta)$[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 $\theta := \text{HOA}(a = b, C)$	
7E.1	$\theta \equiv$ NIL		NIL
7E.2	$\theta \not\equiv$ NIL	Put $\lambda : \equiv \text{IMPLY}(H, A\theta)$[4]	
7E.3	$\lambda \equiv$ NIL	Go To 8	
7E.4	$\lambda \not\equiv$ NIL		$\theta \circ \lambda$
8.	$B \equiv (A \longleftrightarrow D)$		$\text{HOA}((A \longrightarrow D) \wedge (D \longrightarrow A), C)$
9.	$B \equiv (a = b)$	Put $Z := \text{MINUS-ON}(a,b)$	
9.1	$Z \equiv 0$		NIL
9.2	Z is a number		T
9.3	Z is not a number	Put $a' := \text{CHOOSE}(a,b)$, $b' := \text{OTHER}(a,b)$ (see p.16 of [1]) Put $H' := H(a'/b')$, $C' := C(a'/b')$	$\text{IMPLY}(H', C')$
10.	$B \equiv (A \vee D)$		$\text{IMPLY}(B, C)$
11.	$B \equiv \sim A$		$\text{IMPLY}(H, A \vee C)$[8]
12.	ELSE		NIL

---

*ANDS is explained on p.11. of [1].

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

Appendix 2

Some Soundness Results

In this appendix we establish some soundness results for the system, with particular emphasis on the role of TYPELIST.

We would like to establish the property:

If TYPELIST has the value TY and IMPLY (TY, H, C) or HOA (H,C) returns the value (θ TY'), then

(*) $(\sim TY' \wedge TY \wedge H\theta \rightarrow C\theta)$

is a valid formula.

This is equivalent to the informal statement that $(TY \wedge H\theta \rightarrow C\theta)$ is valid "except for the case when TY' is false". (Recall that TYPELIST does not contain skolem variables so substitutions are not applied to it).

To establish this property we will use recursive induction (see [12,13], or [7] p.28). Thus we need only prove that each rule of IMPLY and HOA preserves the above property, assuming that it is preserved by each subcall to IMPLY and HOA within the Rule. This last assumption is called the "induction hypothesis". These induction thypotheses appear as hypotheses in the various theorems below. In every case we will use the abbreviation "TY" for "TYPELIST".

The property (*) is clearly preserved in all cases when a result of the form (θ NIL) is returned for then TY' $\equiv$ NIL, and (*) becomes

$$(TY \wedge H\theta \rightarrow C\theta).$$

It also holds in case NIL is Returned. Since also IMPLY Rules 3, 5, 6, 7, 8, 10, 11, 12, and HOA Rules 2.2, 2.3, 3, 5, 8, 9, 10, 11, returns a single call to IMPLY or HOA, we are left with only IMPLY Rules 2.4, 4.4, 11, and HOA Rules

4.5, 4.6, 6.5, 6.6, 7.4, 7E.4, and 7LE.6, to handle. These appear in Tables I-T, and II-T, pp. 16-19.

For each of these, we state below: the goal being attempted when the rule is applied; the rule itself; and the theorem validating that rule. The proofs are given by Resolution.

In these proofs we assume that no contradictory substitution θ is ever substituted (i.e., a case where a/x and b/x are both in θ, where $a \neq b$). The results given here can easily be generalized to handle substitutions, which consist of disjunctions of ordinary substitution (see Appendix 3 of [1]), where such contradictory entries are allowed.

<u>GOAL</u> $(TY \wedge H \rightarrow A \wedge B)$

<u>Rule I-T 4.4</u>. If $(TY \wedge H \Rightarrow A)$ returns $(\theta\ TY1)$ and $(TY \wedge H \Rightarrow B\theta)$ returns $(\theta 2\ TY2)$ then return $(\theta \circ \theta 2\ \ (TY1 \vee TY2))$ for $(TY \wedge H \Rightarrow A \wedge B)$.

<u>Theorem</u>. $(\sim TY1 \wedge TY \wedge H\theta \rightarrow A\theta)$

$(\sim TY2 \wedge TY \wedge H\theta 2 \rightarrow (B\theta)\theta 2)$

$\rightarrow (\sim(TY1 \vee TY2) \wedge TY \wedge H \rightarrow A \wedge B)$

<u>Proof</u>. By Resolution

1. $TY1 \vee \sim TY \vee \sim H\theta \vee A\theta$
2. $TY2 \vee \sim TY \vee \sim H\theta 2 \vee B\theta\theta 2$
3. $\sim TY1$
4. $\sim TY2$
5. TY
6. H
7. <u>$\sim A \vee \sim B$</u>
8. $A\theta$ 1,3,5,6
9. $(B\theta)\theta 2$ 2,4,5,6
10. $\sim B\theta$ 7,8
11. $\square$ 9,10

GOAL. $((TY' \vee TY'') \wedge H \rightarrow C)$

Rule I-T 2.4. If $(TY' \wedge H \Rightarrow C)$ returns $(\theta \quad TY1)$ and $(TY'' \wedge H \Rightarrow C)$ returns $(\lambda \quad TY2)$ then return $(\theta \circ \lambda \quad (TY1 \vee TY2))$ for $((TY' \vee TY'') \wedge H \Rightarrow C)$.

Theorem. $(\sim TY1 \wedge TY' \wedge H\theta \rightarrow C\theta)$

$(\sim TY2 \wedge TY'' \wedge H\lambda \rightarrow C\lambda)$

$\rightarrow (\sim(TY1 \vee TY2) \wedge (TY' \vee TY'') \wedge H \rightarrow C)$

Proof. By Resolution.

1. $TY1 \vee \sim TY' \vee \sim H\theta \vee C\theta$
2. $TY2 \vee TY'' \vee H\lambda \vee C\lambda$
3. $\sim TY1$
4. $\sim TY2$
5. $TY' \vee TY''$
6. H
7. $\sim C$
8. $\sim TY'$ 1,3,6,7
9. $\sim TY''$ 2,4,6,7
10. $\Box$ 5,8,9

GOAL. $(TY \wedge H \rightarrow a \leq b)$

RULE I11. Return $(NIL \quad \sim(a \leq b) \wedge TY)$

Theorem. $\sim[\sim(a \leq b) \wedge TY] \rightarrow (TY \wedge H \rightarrow a \leq b)$

Proof. $\sim[\sim(a \leq b) \wedge TY] \longleftrightarrow [a \leq b \vee \sim TY]$

$\longleftrightarrow (TY \rightarrow a \leq b)$

$\rightarrow (TY \wedge H \rightarrow a \leq b)$

GOAL. $(TY \wedge B \rightarrow A \vee D)$

Rule H-T 4.5. If $(TY \wedge B \wedge \sim D \Rightarrow A)$ returns $(\theta \quad TY1)$ and $(TY1 \wedge B \wedge \sim A \Rightarrow D)$ returns NIL, then return $(\theta \quad TY1)$ for $(TY \wedge B \Rightarrow A \vee D)$.

Theorem. $(\sim TYL \wedge TY \wedge B\theta \wedge \sim D\theta \rightarrow A\theta)$

$\longrightarrow (\sim TY1 \wedge TY \wedge B\theta \rightarrow A\theta \vee D\theta)$

Proof. These are equivalent.

Rule H-T 4.6. If $(TY \wedge B \wedge \sim D \Rightarrow A)$ returns $(\theta \quad TY1)$ and $(TY1 \wedge B \wedge \sim A \Rightarrow D)$ returns $(\lambda \quad TY2)$ then return $(\theta \circ \lambda \quad TY2)$ for $(TY \wedge B \Rightarrow A \vee D)$

Theorem. $(\sim TY1 \wedge TY \wedge B\theta \wedge \sim D\theta \rightarrow A\theta)$

$(\sim TY2 \wedge TY1 \wedge B\lambda \wedge \sim A\lambda \rightarrow D\lambda)$

$\longrightarrow (\ TY2 \wedge TY \wedge B \rightarrow A \vee D)$

Proof. By Resolution.

1. $TY1 \vee TY \vee \sim B\theta \vee D\theta \vee A\theta$
2. $TY2 \vee \sim TY1 \vee \sim B\lambda \vee A\lambda \vee D\lambda$
3. $\sim TY2$
4. TY
5. B
6. $\sim A$
7. $\sim D$
8. $TY1$ 1,4,5,7,6
9. $\sim TY1$ 2,3,5,6,7
10. $\Box$ 8,9

GOAL. $(TY \wedge H \wedge (A \rightarrow D) \rightarrow C)$

Rule H-T 7.4. If ANDS (D,C) returns θ and $(TY \wedge H \wedge (A \rightarrow D) \rightarrow A\theta)$ returns $(\lambda \quad TY2)$ then return $(\theta \circ \lambda \quad TY2)$ for $(TY \wedge H \wedge (A \rightarrow D) \Rightarrow C)$.

Theorem. $(D\theta \rightarrow C\theta)$

$\wedge (\sim TY2 \wedge TY \wedge H \wedge (A \rightarrow D)\lambda \rightarrow A\theta\lambda)$

$\longrightarrow (\sim TY2 \wedge TY \wedge H \wedge (A \rightarrow D) \rightarrow C)$

Proof. By Resolution.

1. $\sim D\theta \vee C\theta$
2. $TY2 \vee \sim TY \vee \sim H \vee A\lambda \vee A\theta\lambda$

3. $TY2 \vee \sim TY \vee \sim H \vee \sim D\lambda \vee A\theta\lambda$

4. $\sim TY2$

5. TY

6. H

7. $\sim A \vee D$

<u>8. $\sim C$</u>

9. $\sim D\theta$ 1,8

10. $A\lambda \vee A\theta\lambda$ 2,4,5,6

11. $\sim D\lambda \vee A\theta\lambda$ 3,4,5,6

12. $D\lambda \vee D\theta\lambda$ 10,7

13. $D\lambda$ 9,12

14. $A\theta\lambda$ 13,11

15. $D\theta\lambda$ 7,14

16. $\Box$ 9,15

<u>GOAL</u>. $(TY \wedge A \wedge D \rightarrow C)$

<u>Rule H-T 6.5</u>. If $(TY \wedge A \Rightarrow C)$ returns $(\theta \quad TY1)$ and $(TY1 \wedge D \Rightarrow C)$ returns NIL then return $(\theta \quad TY1)$ for $(TY \wedge A \wedge D \Rightarrow C)$.

<u>Theorem</u>. $(\sim TY1 \wedge TY \wedge A\theta \rightarrow C\theta)$

$\longrightarrow (TY1 \wedge TY \wedge A\theta \wedge D\theta \Rightarrow C\theta)$

<u>Proof</u>. Obvious

<u>Rule H-T 6.6</u>. If $(TY \wedge A \Rightarrow C)$ returns $(\theta \quad TY1)$ and $(TY1 \wedge D \Rightarrow C)$ returns $(\lambda \quad TY2)$ then return $(\theta \circ \lambda \quad TY2)$ for $(TY \wedge A \wedge D \Rightarrow C)$.

<u>Theorem</u>. $(\sim TY1 \wedge TY \wedge A\theta \rightarrow C\theta)$

$(\sim TY2 \wedge TY1 \wedge D\lambda \rightarrow C\lambda)$

$\longrightarrow (\sim TY2 \wedge TY \wedge A \wedge D \rightarrow C)$

<u>Proof</u>. By Resolution

1. $TY1 \vee \sim TY \vee \sim A\theta \vee C\theta$

2. $TY2 \vee \sim TY1 \vee \sim D\lambda \vee C\lambda$

3. $\sim$TY2

4. TY

5. A

6. D

7. $\sim$C

8. TY1 1,4,5,7

9. TY1 2,3,6,7

10. $\square$ 8,9

GOAL. $(TY \wedge H \wedge (A \rightarrow A = b) \rightarrow C)$

Rule H-T 7E.4. If $(TY \wedge H \wedge a = b \Rightarrow C)$ returns $(\theta \quad TY1)$ and

$(TY \wedge H \wedge (A \rightarrow a = b) \Rightarrow A\theta)$ returns $(\lambda \quad TY2)$ then

returns $(\theta \circ \lambda \quad (TY1 \vee TY2))$ for $(TY \wedge H \wedge (A \rightarrow a = b) \Rightarrow C)$

GOAL. $(TY \wedge H \wedge (A \rightarrow a \leq b) \rightarrow C)$

Rule H-T 7LE.6. If $(TY \wedge H \wedge a \leq b \Rightarrow C)$ returns $(\theta \quad TY1)$ and

$(TY \wedge H \wedge (A \rightarrow a \leq b) \Rightarrow A\theta)$ returns $(\lambda \quad TY2)$ then

return $(\theta \circ \lambda \quad (TY1 \vee TY2)$ for $(TY \wedge H \wedge (A \rightarrow a \leq b) \Rightarrow C)$.

Theorem. (For both). (D for a = b or a b.

$(\sim TY1 \wedge TY \wedge H\theta \wedge D \rightarrow C\theta)$

$(\sim TY2 \wedge TY \wedge H\lambda \wedge (A\lambda \rightarrow D) \rightarrow A\theta\lambda)$

$\rightarrow (\sim (TY1 \vee TY2) \wedge TY \wedge H \wedge (A \rightarrow D) \rightarrow C)$

Proof. By Resolution.

1. $TY1 \vee \sim TY \vee \sim H\theta \vee \sim D \vee C\theta$

2. $TY2 \vee \sim TY \vee \sim H\lambda \vee A\lambda \vee A\theta\lambda$

3. $TY2 \vee \sim TY \vee \sim H\lambda \vee \sim D \vee A\theta\lambda$

4. $\sim$TY1

5. $\sim$TY2

6. TY

7. H

8. $\sim A \vee D$

<u>9. ∿C</u>

10.	∿D	1,4,6,7,9
11.	Aλ ∨ Aθλ	2,5,6,7
12.	∿D ∨ Aθλ	3,5,6,7
13.	∿A	10,8
14.	□	13,11

Appendix 3

Output from the ISI Program Verification System
(The prover is called on page 5)

```
TELNET typescript file started at FRI 25 APR 75 0954:04v
@XVERIFIER/3-2-1.SAV;1

VERIFIER  3.2  UCILSP BASED    18-APR-75
HI LARRY

>SCANTR:=NIL;

NIL


>TY;

FILE TO BE TYPED: BSRCH.PAS;1 [Old version]


%This program does binary search on the array A[1 .. P-1] trying
%to locate the element X.  If successful, then LOOKUP is set
%such that A[LOOKUP]=X and ERROR is set FALSE.  If unsuccessful,
%ERROR is set TRUE.  More on this problem may be found in
%Section 5 of Igarashi, London, and Luckham.
ENTRY 1 < P & SORTED(A) & A[1] LE X & X < A[P];
EXIT (A[LOOKUP]=X) AND (ERROR=FALSE) OR NOTFOUND(X,1,P) AND ( ER
**ROR =TRUE);
BEGIN M:=1;N:=P;ERROR:=FALSE;
ASSERT M < N & A[M] LE X & X < A[N] & SORTED (A) &ERROR=FALSE;
WHILE M+1<N DO BEGIN
        I:=(M+N)DIV 2;
        IF X < A[I] THEN N:=I ELSE IF A[I] < X THEN M:=I
                ELSE BEGIN LOOKUP:=I;GOTO 1 END
        END;
IF A[M] NE X THEN GO TO 2 ELSE BEGIN LOOKUP:=M;GO TO 1 END;
2: ASSERT NOTFOUND(X,1,P);ERROR:=TRUE;
1: ASSERT (A[LOOKUP]=X) AND (ERROR=FALSE ) OR NOTFOUND(X,1,P) AN
**D (ERROR=TRUE);
END.

NIL


>PROVE BSRCH;
 RESTORE:  NO($), DMP, PRE, VC, VCS
>$
PARSE: $=BSRCH.PAS,RESET (FILENAME)
>$PROCEEDING

Parsing . . .

VCGEN: P(ROCEED)($), UNIT
>$
MAIN#1
```

```
MAIN#2

MAIN#3

MAIN#4

MAIN#5

MAIN#6

MAIN#7

MAIN#8

 TRYING TO SIMPLIFY MAIN#1
CHOICE:   P(ROCEED)($),+/-N,VCGEN,ASSUME,
          END,DEFER,SWITCH,STATUS,RED(UCE)
>$PROCEEDING

VERIFICATION CONDITION  MAIN#1

SIMPLIFICATION
>>> ENTERING RPV WITH

            1<P
        AND SORTED(A)
        AND A[1] LE X
        AND X < A[P]
  IMP       1<P
        AND A[1] LE X
        AND X < A[P]
        AND SORTED(A)
        AND FALSE=FALSE

>>> ENTERING RPROVER WITH

TRUE


<<< LEAVING RPROVER WITH
TRUE

VC WAS MAIN#1
 TRYING TO SIMPLIFY MAIN#2
CHOICE:   P(ROCEED)($),+/-N,VCGEN,ASSUME,
          END,DEFER,SWITCH,STATUS,RED(UCE)
>$PROCEEDING

VERIFICATION CONDITION  MAIN#2

SIMPLIFICATION
>>> ENTERING RPV WITH

            M<N
```

```
      AND A[M] LE X
      AND X < A[N]
      AND SORTED(A)
      AND ERROR=FALSE
      AND M+1 < N
 IMP        X < A[(M+N) DIV 2]
     IMP        (M < (M+N) DIV 2) AND (A[M] LE X)
           AND X < A[(M+N) DIV 2]
           AND SORTED(A)
           AND ERROR=FALSE


SUBING ERROR:=FALSE
>>> ENTERING RPROVER WITH

            SORTED(A)
      AND M+2 LE N
      AND M<N
      AND X < A[N]
      AND X < A[(N+M) DIV 2]
      AND A[M] LE X
 IMP        SORTED(A)
      AND M < (N+M) DIV 2
      AND X < A[(N+M) DIV 2]
      AND A[M] LE X

HCMATCH MATCHED SORTED(A)
MATCHED X < A[(N+M) DIV 2]
MATCHED A[M] LE X

HCMATCH GIVES

            SORTED(A)
      AND M+2 LE N
      AND M<N
      AND X < A[N]
      AND X < A[(N+M) DIV 2]
      AND A[M] LE X
 IMP M < (N+M) DIV 2
INSUB LEPRV IMPPRV LOGSUB SAVESTATE MPHYP EXPQ CHECKSTATE
<<< LEAVING RPROVER WITH
            SORTED(A)
      AND M+2 LE N
      AND M<N
      AND X < A[N]
      AND X < A[(N+M) DIV 2]
      AND A[M] LE X
 IMP M < (N+M) DIV 2

VC WAS MAIN#2     SAVE AS?
>$MAIN#S2

 TRYING TO PROVE MAIN#S2
CHOICE:  P(ROCEED)($),+/-N,VCGEN,ASSUME,
```

```
          END,DEFER,SWITCH,STATUS,RED(UCE)
>DEFER

 TRYING TO SIMPLIFY MAIN#3
CHOICE:   P(ROCEED)($),+/-N,VCGEN,ASSUME,
          END,DEFER,SWITCH,STATUS,RED(UCE)
>2

VERIFICATION CONDITION  MAIN#5

SIMPLIFICATION
>>> ENTERING RPV WITH

          M<N
      AND A[M] LE X
      AND X < A[N]
      AND SORTED(A)
      AND ERROR=FALSE
      AND NOT (M+1 < N)
 IMP A[M] NE X IMP NOTFOUND(X, 1, P)


SUBING ERROR:=FALSE
>>> ENTERING RPROVER WITH

          SORTED(A)
      AND M<N
      AND X < A[N]
      AND N LE M+1
      AND A[M] LE X
      AND NOT (X = A[M])
 IMP NOTFOUND(X, 1, P)

HCMATCH INSUB LEPRV IMPPRV LOGSUB SAVESTATE MPHYP EXPQ
NEW EQUALITY  M+1 = N
FROM: M<N
AND:  N LE M+1
EXPQ GIVES

          SORTED(A)
      AND M<N
      AND X < A[N]
      AND N LE M+1
      AND A[M] LE X
      AND NOT (X = A[M])
      AND M+1 = N
 IMP NOTFOUND(X, 1, P)
CHECKSTATE INSUB
SUB: TYPE Y(ES), N(O), ? FOR MNEMONICS, HELP FOR COMMAND SUMMARY
M:=N-1
WARNING!!! LEFT SIDE OF PROPOSED SUBST DOES NOT APPEAR IN ANY CONCS.
>SS
1) M:=N-1
2) N:=M+1
```

```
TYPE NUMBER BETWEEN 1 AND 2
>2

SUB: TYPE Y(ES), N(O), ? FOR MNEMONICS, HELP FOR COMMAND SUMMARY
N:=M+1
WARNING!!! LEFT SIDE OF PROPOSED SUBST DOES NOT APPEAR IN ANY CONCS.
>Y
 SUB USED: N:=M+1

INSUB GIVES

          SORTED(A)
      AND X < A[M+1]
      AND A[M] LE X
      AND NOT (X = A[M])
 IMP NOTFOUND(X, 1, P)
LEPRV IMPPRV
<<< LEAVING RPROVER WITH
          SORTED(A)
      AND X < A[M+1]
      AND A[M] LE X
      AND NOT (X = A[M])
 IMP NOTFOUND(X, 1, P)

VC WAS MAIN#5      SAVE AS?
>$MAIN#S5

 TRYING TO PROVE MAIN#S5
CHOICE:  P(ROCEED)($),+/-N,VCGEN,ASSUME,
         END,DEFER,SWITCH,STATUS,RED(UCE)
>STATUS
   MAIN#1 ***PROVED***
   MAIN#2 HAS BEEN SIMPLIFIED TO
      MAIN#S2 (DEFFERED) TO BE PROVED
   MAIN#3 HAS BEEN SIMPLIFIED TO
      MAIN#S3 (DEFFERED) TO BE PROVED
   MAIN#4 ***PROVED***
   MAIN#5 HAS BEEN SIMPLIFIED TO
      MAIN#S5 TO BE PROVED
   MAIN#6 HAS BEEN GENERATED
   MAIN#7 HAS BEEN GENERATED
   MAIN#8 HAS BEEN GENERATED

 TRYING TO PROVE MAIN#S5
CHOICE:  P(ROCEED)($),+/-N,VCGEN,ASSUME,
         END,DEFER,SWITCH,STATUS,RED(UCE)
>END

PROVE: NO($),UN(DEFERRED),OR DEF(ERRED) (VC'S)
>$
DUMP:  DMP($), PRE, VC, VCS, NO, CLEAR (STRUCTURE)
>NO

NIL
```

The Prover is called here

```
>PROVEIT VCM5;

VERIFICATION CONDITION VCM5
(THEOREM TO BE PROVED)
NIL

          SORTED(M, MIN(N+1, 2), N)
      AND 2 LE N
      AND A(M, 2, MIN(N, 1))
      AND      IP1LARGEST(MIN(N, 1), M)
           OR 0 = MIN(-N + 1, 0)
 IMP SORTED(M, 1, N)
(BACKUP POINT)
W>$PROCEEDING
(BACKUP POINT)
(P->)
W>TP

          N IN [2..INFINITY]
      AND SORTED(M, MIN(N+1, 2), N)
      AND A(M, 2, MIN(N, 1))
      AND      IP1LARGEST(MIN(N, 1), M)
           OR 0 = MIN(-N + 1, 0)
 IMP SORTED(M, 1, N)
W>$PROCEEDING
......(P-> ORH)
(P-> ORH 1)
(BACKUP POINT)
(P-> ORH 1 P->)
W>TP

          N IN [2..INFINITY]
      AND SORTED(M, MIN(N+1, 2), N)
      AND A(M, 2, MIN(N, 1))
      AND IP1LARGEST(MIN(N, 1), M)
 IMP SORTED(M, 1, N)
W>$PROCEEDING
......RAN OUT OF TRICKS
W>USE
LEMMA:
>SORTED(M,I+1,N) AND (M[I] LE M[I+1]) IMP SORTED(M,I,N);
==> (1)

          SORTED(M, I+1, N)
      AND M[I] LE M[I+1]
 IMP SORTED(M, I, N)
<== (1)

          SORTED(M, I+1, N)
      AND M[I] LE M[I+1]
 IMP SORTED(M, I, N)
(LEMMA USED SAVED IN L240)
```

```
          SORTED(M, I+1, N)
      AND M[I] LE M[I+1]
 IMP SORTED(M, I, N)
OK???
>YES
(USE)==============================
(P-> ORH 1 P-> U)
W>$PROCEEDING
.(P-> ORH 1 P-> U H)
(P-> ORH 1 P-> U H 1)
......RAN OUT OF TRICKS
W>TP

          N IN [2..INFINITY]
      AND SORTED(M, MIN(N+1, 2), N)
      AND A(M, 2, MIN(N, 1))
      AND IP1LARGEST(MIN(N, 1), M)
 IMP SORTED(M, 2, N)
W>R H

     N IN [2..INFINITY]
 AND SORTED(M, 2, N)
 AND A(M, 2, 1)
 AND IP1LARGEST(1, M)
OK???
>YES
W>TP

          N IN [2..INFINITY]
      AND SORTED(M, 2, N)
      AND A(M, 2, 1)
      AND IP1LARGEST(1, M)
 IMP SORTED(M, 2, N)
W>$PROCEEDING
...(P-> ORH 1 P-> U H 1)

SORTED(M, 2, N)
PROVED
W>$PROCEEDING
 (P-> ORH 1 P-> U H 2)
MORE TIME ? (TYPE NUMBER OR NO)
>NO

M[1] LE M[2]
FAILED TIME LIMIT
W>TP

          SORTED(M, 2, N)
      AND N IN [2..INFINITY]
      AND SORTED(M, MIN(N+1, 2), N)
      AND A(M, 2, MIN(N, 1))
      AND IP1LARGEST(MIN(N, 1), M)
  IMP M[1] LE M[2]
```

```
W>R H

     SORTED(M, 2, N)
 AND N IN [2..INFINITY]
 AND SORTED(M, 2, N)
 AND A(M, 2, 1)
 AND IP1LARGEST(1, M)
OK???
>OK
W>$PROCEEDING
........RAN OUT OF TRICKS
W>TP

          SORTED(M, 2, N)
      AND N IN [2..INFINITY]
      AND SORTED(M, 2, N)
      AND A(M, 2, 1)
      AND IP1LARGEST(1, M)
 IMP M[1] LE M[2]
W>USE
LEMMA:
>IP1LARGEST(1,M) IMP (M[1] LE M[2]);
==> (1)

     IP1LARGEST(1, M)
 IMP M[1] LE M[2]
<== (1)

     IP1LARGEST(1, M)
 IMP M[1] LE M[2]
(LEMMA USED SAVED IN L241)

     IP1LARGEST(1, M)
 IMP M[1] LE M[2]
OK???
>YES
(USE)==============================
(P-> ORH 1 P-> U H 2 U)
W>$PROCEEDING
.(P-> ORH 1 P-> U H 2 U H)
......(P-> ORH 1 P-> U H 2 U H)

IP1LARGEST(1, M)
PROVED
W>$PROCEEDING
(P-> ORH 1 P-> U H 2)

M[1] LE M[2]
PROVED
W>$PROCEEDING
(P-> ORH 1 P-> U H)

     SORTED(M, 2, N)
 AND M[1] LE M[2]
```

```
PROVED
W>$PROCEEDING
(P-> ORH 1)

          N IN [2..INFINITY]
      AND SORTED(M, MIN(N+1, 2), N)
      AND A(M, 2, MIN(N, 1))
      AND IP1LARGEST(MIN(N, 1), M)
 IMP SORTED(M, 1, N)
PROVED
W>$PROCEEDING
(P-> ORH 2)
(BACKUP POINT)
(P-> ORH 2 P->)
W>TP

          N IN [2..INFINITY]
      AND SORTED(M, MIN(N+1, 2), N)
      AND A(M, 2, MIN(N, 1))
      AND 0 = MIN(-N + 1, 0)
 IMP SORTED(M, 1, N)
W>A
ASSUMED

(P-> ORH 2)

          N IN [2..INFINITY]
      AND SORTED(M, MIN(N+1, 2), N)
      AND A(M, 2, MIN(N, 1))
      AND 0 = MIN(-N + 1, 0)
 IMP SORTED(M, 1, N)
PROVED
W>$PROCEEDING
(P-> ORH)

                  N IN [2..INFINITY]
              AND SORTED(M, MIN(N+1, 2), N)
              AND A(M, 2, MIN(N, 1))
              AND IP1LARGEST(MIN(N, 1), M)
         IMP SORTED(M, 1, N)
 AND              N IN [2..INFINITY]
              AND SORTED(M, MIN(N+1, 2), N)
              AND A(M, 2, MIN(N, 1))
              AND 0 = MIN(-N + 1, 0)
         IMP SORTED(M, 1, N)
PROVED
W>$PROCEEDING
(P->)

SORTED(M, 1, N)
PROVED
W>$PROCEEDING
NIL
```

May 6, 1975

Unsolicited remarks of a user

who had just proved a theorem on the interactive system:

"I really had no idea what the theorem was saying, but armed with the relevant lemmas, I just let the machine do the work.

The conclusion of the theorem looked very much like the conclusion of one of the lemmas I had. So naturally I tried to use it, but soon realized that it was a back-chaining trap. That was no real problem, I simply backed up and tried another lemma which seemed to fit. When I back-chained and tried to prove the hypotheses of that lemma it soon became apparent that another lemma was needed. And so it went until I noticed that an equality chain could possibly be built. I wasn't sure one existed but it didn't hurt to try. You know what happened then - it actually discovered a chain and reduced my problem to proving the hypotheses of that chain. I still didn't know what I was proving, but the only remaining problem was to find values for the two variables A and B in C, which it did quickly."

<u>References</u>

1. W.W. Bledsoe and Mabry Tyson. The U.T. Interactive Prover. Univ. of Texas at Austin, Math. Dept. Memo. ATP-17, May 1975.

2. Alan Bundy. Doing Arithmetic with diagrams. <u>Third Int. Joint Conf. Artif. Intell.</u>, 1973, pp. 130-138.

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

4. R.J. Waldinger and K.N. Levitt. Reasoning about programs. <u>Artif. Intell.</u>, vol. 5, no. 3, pp. 235-316, Fall 1974; also in <u>Conf. Rec. Ass. Comput. Mach. Symp. Principles of Programming Languages</u>, 1973, pp. 169-182.

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

6. W.W. Bledsoe, Program Correctness. Univ. of Texas at Austin, Math. Dept. Memo ATP-14, January 1974. (out of print)

7. W.W. Bledsoe. The Sup-Inf method in Presburger Arithmetic. Univ. of Texas at Austin Math. Dept. 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.

8. D.C. Cooper. Programs for mechanical program verification. <u>Mach. Intell. 6</u>. American Elsevier, New York, 1971. 43-59.

9. 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 <u>IEEE Trans. on Software Engineering</u> 1(1975), pp. 59-67.

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

11. Mabry Tyson. An algebraic simplifier. Univ. of Texas at Austin, Math. Dept. Memo. ATP-26, 1975.

12. Zohar Manna, S. Ness and J. Vuillemin. Inductive method for proving properties of programs. Comm. ACM, Aug. 1973.

13. R.M. Burstall. Proving properties of programs by structural induction. <u>Computer J.</u> 12, 1(Feb. 1969), pp. 41-48.

14. Joel Moses. Symbolic-Integration. MIT-AI Memo 97, June 1966.