Typing and Proof by Cases in Program Verification

by

W. W. Bledsoe and Mabry Tyson

May 1975 ATP 15

* The work reported here was supported by NSF Grant #DCR74-12886.

Typing and Proof by Cases in Program Verification

W.W. Bledsoe and Mabry Tyson

ABSTRACT

Special procedures have been added to an automatic prover to facilitate its handling of inequalities and proof by cases. A data base, called TYPELIST, is used which maintains upper and lower bounds of variables occuring in the proof of a theorem. These procedures have been coded and used to (interactively) prove several theorems arising in automatic program verification.

Introduction

We describe here procedures that have been added to an automatic theorem prover [1] to make it more effective in proving verification conditions (theorems) that arise in the field of program verification. These procedures, which handle inequalities and equalities, and proof by cases, are based upon a pointer system used by Bundy [2], SRI [3,4], and others to handle inequalities, and upon the interval types used in [5]. The present description follows somewhat the discussion in [6].

In order to follow this presentation the reader should have some understanding of the prover described in [1]. However we feel that many workers in this field are already generally familiar with our prover and can read this paper directly, referring to [1] only when the need arises. Tables I and II from [1] are included here in Appendix 1, for convenience, but the reader is referred to Section 2 of [1] for a fuller understanding.

These methods can also be used in Resolution based provers and other Gentzen type systems. Section 5 gives a brief description of this for resolution.

1. Types

Typing information can be a powerful asset in automatic theorem proving. For example, knowing that j and k are non-negative integers and that $j < k$ lets us deduce that $j * k \geq 0$, $j \leq k - 1$, etc. Often, we have other "typing" information. For example, we may know (from a given hypothesis) that j lies in some interval, $a \leq j \leq b$. In our system, we have decided to include such information as part of the type of j. Thus j has the type: "non-negative integer in the interval $a \leq j \leq b$". We express this fact by the notation $\{j: a\ b\}$.

In what follows, certain variables $i, j, k, \ldots$ occur in inequalities and can assume only non-negative integer values. These will be "typed" as indicated above. Such variables often arise as program variables in computer programs. (Actually these variables are all universally quantified in the theorem being proved and are converted to skolem constants by the skolemization process, but that need not concern us here. Refer to Appendix 1 of [1] and Section 1, of [1].)

Upper and lower bounds are computed and maintained for these typed variables. When a new inequality is encountered, as a hypothesis, the bounds for these variables are updated appropriately. This interval information is kept in a knowledge base (which we call the TYPELIST), which represents the "state of the world for these variables at that particular time, and serves as an additional hypothesis to the theorem or subgoal being considered. For example, a hypothesis

$$(a \leq j \leq b)$$

is stored in TYPELIST as

$$\{j:\ a\ b\}$$

which means that j is in the closed interval $[a,b]$[1]. If a contradiction such as $\{j:\ k\ k\text{-}1\}$ occurs in TYPELIST, this represents a false hypothesis and successfully terminates the proof. Also if an entry $\{j:\ N\ \infty\}$ is already in TYPELIST, any new hypothesis such as $(j \le N+1)$ causes the entry to be updated to $\{j:\ N\ \ N+1\}$, which means that j can take only the value N or the value $N+1$.

An entry of the form $\{j:\ N+1\ \ N+1\}$ which occurs in TYPELIST is treated as the equality $(j = N+1)$.

Initially all typed variables j are given the type $\{j:\ 0\ \ \infty\}$.

A subroutine SET-TYPE is used to convert information in the hypothesis of a theorem to TYPELIST entries. It is called at the beginning of the proof and at each point in the proof when new expressions are added to the hypothesis of the theorem being proved. For example, if the theorem being proved is

<u>Ex. 1</u>.

(1) $$(P(1) \wedge 1 \le j \wedge j \le n \wedge j \le 1 \longrightarrow P(j))$$

the original value of TYPELIST is

$$(\{j:\ 0\ \infty\}\{n:\ 0\ \infty\})\ ,$$

[1] Except in the case when b is $+\infty$; then the interval is $[a,\infty)$.

but then SET-TYPE is called on the hypothesis of (1) which changes TYPELIST to

$$(\{j:\ 1\ 1\}\{n: j\ \infty\})$$

and converts (1) to

(2) $$(j = 1 \wedge P(1) \longrightarrow P(j)) \quad .$$

Notice that the program detected that j was equal to 1 from the entry $\{j:\ 1\ 1\}$. The prover will now substitute 1 for j in (2) to obtain

$$(P(1) \longrightarrow P(1))$$

which it recognizes as true.

Other examples are now given.

Ex. 2.

(3) $$(1 \le j \wedge P(1) \longrightarrow (j \le k \wedge k \le 1 \longrightarrow P(k))) \quad .$$

An initial call to SET-TYPE, on the hypothesis of (3), changes TYPELIST to $(\{j:\ 1\ \infty\}\{k:\ 0\ \infty\})$ and converts (3) to

(4) $$(P(1) \longrightarrow (j \le k \wedge k \le 1 \longrightarrow P(k))) \quad .$$

Now Rule 7 of IMPLY (see [1], Table I), converts (4) to

(5) $$(P(1) \wedge j \le k \wedge k \le 1 \longrightarrow P(k))$$

at which time SET-TYPE is again called, which uses $j \le k$ and $k \le 1$ to change TYPELIST to $(\{j:\ 1\ 1\}\{k:\ 1\ 1\})$, and converts (5) to

$$(j = 1 \wedge k = 1 \wedge P(1) \longrightarrow P(k)) \quad .$$

The prover, as before, converts this to

$$(P(1) \longrightarrow P(1))$$

which it recognizes as true.

Ex. 3.

$$(2 \leq j \wedge j \leq 1 \longrightarrow P(j))$$

SET-TYPE changes TYPELIST to ({j: 2 1}). The program detects the contradictions in TYPELIST (i.e., $2 \leq 1$) and successfully concludes the proof.

Whenever an inequality $(a \leq b)$ occurs in the conclusion of the theorem being proved, the prover updates TYPELIST with the negation of $(a \leq b)$, and looks for a contradiction. Thus, for the example

Ex. 4.

(6) $$(j \leq 1 \wedge k \leq j \wedge P \longrightarrow k \leq 3) \quad ,$$

TYPELIST is given the value ({j: k 1}{k: 0 j}) and (6) is converted to

$$(P \longrightarrow k \leq 3) \quad .$$

The prover now uses $(k \not\leq 3)$, which is first converted to $(4 \leq k)$[2], to update TYPELIST, getting ({j: k 1}{k: 4 j}), which contains the contradiction

$$(4 \leq k \leq j \leq 1) \quad .$$

[2]Since k is an integer. See [7, p. 27].

The Prover detects such contradictions by computing absolute upper and lower bounds, sup and inf, for j and k. For this case

$$\sup j = 1 , \quad \inf j = 4$$
$$\sup k = 1 , \quad \inf k = 4 \quad .$$

Since $4 > 1$ we have a contradiction. The prover uses the routines SUP and INF to evaluate these bounds. In [7] we carefully define the algorithms SUP and INF and prove that they have the required properties.

Formula (6), (without the P), is an example of a formula in Presburger Arithmetic. These often arise from computer programs and are discussed in [7] and by Cooper in [8].

Ex. 5. $(2 \leq j \leq 4 \wedge k \leq j \wedge k \leq 7 \rightarrow C)$. Here we use the symbols 'max' and 'min' in typing j and k. TYPELIST is given the value [{j: max(2,k) 4}{k: 0 min(j,7)}].

2. TYPELIST in PROVER

In Section 2 of [1] we describe IMPLY and HOA, the main algorithms of Prover, and give Tables I and II which define them, and list several examples of their use. Tables I and II are reproduced in Appendix 1 of this paper for convenience. The reader is referred to the Section 2 of [1] for a fuller understanding.

IMPLY has five arguments

(TYPELIST, H, C, TL, LT) ,

but in Section 2 of [1] we deal with only H, C, and TL, the hypothesis, conclusion, and theorem label of the theorem or subgoal being proved. For convenience to the reader we represent, in this paper, a call to IMPLY(TYPELIST, H, C, TL, LT) by the notation

(TL) $(H \Rightarrow C)$.

As mentioned earlier TYPELIST represents an additional hypothesis, so we will augment this notation as follows:

(TL) $([\text{TYPELIST}] \wedge H \Rightarrow C)$.

Thus Ex. 2., after it is partially converted, is represented by

(1) $([\{j\colon 1\ 1\}\{k\colon 1\ 1\}] \wedge P(1) \Rightarrow P(k))$.

We will now describe some changes and additions to the Rules of IMPLY and HOA (Tables I and II, of [1]) which have been made to facilitate the use of TYPELIST. Before doing so we first describe the algorithm SET-TYPE, which was mentioned earlier.

SET-TYPE (A)

This algorithm updates TYPELIST by using inequalities and equalities in conjunctive positions of A, and returns a value A', which is the remainder of A not used in updating TYPELIST.

For example, if TYPELIST = [{j: 0 k}{k: j 7}] then a call

$$\text{SET-TYPE}(k \leq 5 \wedge P(j))$$

updates TYPELIST to

[{j: 0 k}{k: j 5}]

and returns the value P(j).

IMPLY RULE CHANGES

	IF	ACTION	RETURN
7.	$C \equiv (A \twoheadrightarrow B)$		IMPLY$(H \wedge A, B)$
	is changed to		
7.	$C \equiv (A \twoheadrightarrow B)$	Put A': = SET-TYPE(A)	
7.1	TY' has a contradiction		"T"
7.2	ELSE		IMPLY(TY', $H \wedge A'$, B)

Where TY' is the updated value of TYPELIST after the action of SET-TYPE(A).

Rule 11 and 14 are added to IMPLY

	IF	ACTION	RETURN
11.	$C \equiv (a \leq b)$	Put A': = SET-TYPE($\sim(a \leq b)$) Let TY' be the updated TYPELIST	
11.1	TY' has a contradiction		"T"
11.2	TY' $\equiv$ TYPELIST	Go to 12 (with TYPELIST and C as they were)	
11.3	TY' $\not\equiv$ TYPELIST		(T TY')
14.1	$C \equiv (a = b)$	Put $C' \equiv (a \leq b \wedge b \leq a)$	IMPLY(H, C')
14.2	$C \equiv (a \neq b)$	Put $C' \equiv (a < b \vee b < a)$	IMPLY(H, C')

Later in this description we will further change these tables, but the reader need not be concerned with that at this time. We will summarize all of these changes in Tables I-T, II-T, of Section 3.

Ex. 5. $(Q \longrightarrow (j \leq 1 \wedge k \leq j \wedge P \longrightarrow k \leq 3))$

(1) $([\{j: 0\ \infty\}\{k: 0\ \infty\}]$

$\Rightarrow (Q \longrightarrow (j \leq 1 \wedge k \leq j \wedge P \longrightarrow k \leq 3)))$

Note that each of j and k is given the original type $[0\ \infty)$, when the theorem is given to Prover.

(1) $([\{j: 0\ \infty\}\{k: 0\ \infty\}] \wedge Q$

$\Rightarrow (j \leq 1 \wedge k \leq j \wedge P \longrightarrow k \leq 3))$ I 7

In this case SET-TYPE(Q) left TYPELIST unchanged and returned the value Q.

(1) $(\underbrace{[\{j: k\ 1\}\{k: 0\ j\}]}_{\text{TYPELIST}} \wedge \underbrace{(Q \wedge P)}_{H} \longrightarrow \underbrace{k \leq 3}_{C})$ I 7

Here SET-TYPE $(j \leq 1 \wedge k \leq j \wedge P)$ has updated TYPELIST to the new value shown, and returned P, which was conjoined to Q.

Now the new Rule I-11, employes SET-TYPE$(\sim(k \leq 3))$ = SET-TYPE$(4 \leq k)$ to update TYPELIST to TY' = $[\{j: k\ 1\}\{k: 4\ j\}]$, and Rule 11.1 detects the contradiction

$$4 \leq j \leq 1$$

in TY' and terminates the proof successfully.

As mentioned in Section 1, we detect the contradiction in

$$TY' = [\{j: k\ 1\}\{k: 4\ j\}]$$

(or any other list of inequalities) by computing

$$\sup_{TY'}(j) \quad \text{and} \quad \inf_{TY'}(j) \quad .$$

In this case

$$\sup_{TY'}(j) = 1, \qquad \inf_{TY'}(j) = 4 \quad ,$$

and since $4 > 1$ we have a contradiction. These are computed by the algorithms SUP and INF (See [7], especially Section 3). In this example the values of sup and inf are rather obvious; for more involved examples see Section 5 of [7].

We have decided to give each variable j just <u>one</u> interval {j: a b} in TYPELIST. So if we are proving a goal of the form

$$((j \leq 1 \vee j \geq 5) \wedge H \Longrightarrow C) \quad ,$$

where there is a <u>disjunction</u> of inequalities in the hypothesis, then we use <u>two</u> TYPELIST's expressed in the form

$$(([\{j:\ 0\ 1\}\{k: \quad \}\cdots] \vee [\{j:\ 5\ \infty\}\{k: \quad \}\cdots])$$
$$\wedge\ H \Longrightarrow C).$$

To handle such examples we add Rule 2 to IMPLY to split such goals into two subgoals.

2.	TYPELIST $\equiv$ TY' $\vee$ TY"	Put θ: = IMPLY(TY',H,C)	
2.1	$\theta \equiv$ NIL		NIL
2.2	$\theta \not\equiv$ NIL	Put λ: = IMPLY(TY",H,C)	
2.3	$\lambda \equiv$ NIL		NIL
2.4	$\lambda \not\equiv$ NIL		$\sigma \circ \lambda$

<u>Ex. 7</u>. $(k \leq 3 \longrightarrow k \leq 1 \vee 2 \leq k \leq 3)$.

(1) $(\{k:\ 0\ \infty\} \Rightarrow (k \leq 3 \longrightarrow k \leq 1 \vee 2 \leq k \leq 3))$

(1) $(\{k:\ 0\ 3\} \Rightarrow k \leq 1 \vee 2 \leq k \leq 3)$ I 7

$(\{k:\ 0\ 3\} \wedge \sim(2 \leq k \leq 3) \Rightarrow k \leq 1)$ H 4.2

$(\{k:\ 0\ 3\} \wedge (k \leq 1 \vee 4 \leq k) \Rightarrow k \leq 1)$

$((\{k:\ 0\ 1\} \vee \{k:\ 4\ 3\}) \Rightarrow k \leq 1)$

(1 1) $(\{k:\ 0\ 1\} \Rightarrow k \leq 1)$ I 2

Rule 10' uses $\sim(k \leq 1)$ to update TYPELIST to $\{k:\ 2\ 1\}$ and Rule 10.2 detects the contradiction. "T"

(1 2) $(\{k:\ 4\ 3\} \Rightarrow k \leq 1)$

Proved since $\{k:\ 4\ \ 3\}$ is a contradiction.

3. Cases

Many of the theorems (verification conditions) from program validation require a proof by cases, in that the theorem must be proved separately for two different ranges of values for some variable. Ex. 7 is such a case, but there the proof was straightforward because the two cases,

$$k \leq 1 \quad \text{and} \quad 2 \leq k \leq 3$$

were stated explicitly in the theorem.

On the other hand, consider the following equivalent form of Ex. 7.

Ex. 8. $((k \leq 3 \wedge (k \leq 1 \longrightarrow C) \wedge (2 \leq k \leq 3 \longrightarrow C) \longrightarrow C)$.

(1) $(\{k: 0\ 3\} \wedge (k \leq 1 \longrightarrow C) \wedge (2 \leq k \leq 3 \longrightarrow C) \Rightarrow C)$ I 7

Backchaining (Rule H 7) off of the hypothesis $(k \leq 1 \longrightarrow C)$ we obtain the subgoal

(1 H) $(\{k: 0\ 3\} \wedge (k \leq 1 \longrightarrow C) \wedge (2 \leq k \leq 3 \longrightarrow C) \longrightarrow k \leq 1)$

which is false. Similarly if we backchain off of the hypothesis $(2 \leq k \leq 3 \longrightarrow C)$ we fail again.

If the prover could somehow be made to know that it should consider the two cases

$$k \leq 1 \quad \text{and} \quad 2 \leq k \leq 3$$

as it did in Ex. 7 the proof would proceed routinely.

We could, of course, require that prover backchain off of both of these hypotheses and thereby set up the provable subgoal

$$(k \leq 1 \vee 2 \leq k \leq 3) \quad ,$$

but such a rule is not only unnatural, it is combinatorially explosive. What's more, a similar problem arises in many other theorems, such as

<u>Ex. 9</u>. $(1 \leq n)$

$$\wedge \forall m\ (2 \leq n \wedge 1 \leq m \wedge m \leq 1 \longrightarrow A[m] \leq A[2])$$
$$\wedge \forall k\ (k+1 \leq n \wedge 2 \leq k \longrightarrow A[k] \leq A[k+1])$$
$$\longrightarrow \forall\ K(K+1 \leq n \wedge 1 \leq K \longrightarrow A[K] \leq A[K+1])$$

and Example 10 below, which are more complicated than Exercise 8 and which will not submit to such an attack.

The procedure we employ to prove Ex. 8 and all others like it, forces the prover into a proof by cases in a natural way. This is effected by further changes and additions to Tables 1 and 2. These are shown (for the most part) in Tables I-T and II-T below. These changes are justified by the results in Appendix 2.

These changes require that IMPLY and HOA now return a pair

$$(\theta \quad TY') \quad ,$$

where θ is the same substitution we got before, and TY' is a new value of TYPELIST which can be used in subsequent calls to IMPLY. This outputed value TY' represents the part of the theorem that has not been proved. Thus if $(\theta \quad TY')$ is returned from a call IMPLY (TYPELIST, H, C), it means that $(TYPELIST \wedge H \longrightarrow C)$ is valid except for the case TY', or that

$$(\sim TY' \wedge TYPELIST \wedge H \longrightarrow C)$$

is valid. See Appendix 2.

Table I-T

TYPELIST VERSION

IMPLY RULE CHANGES*

	IF	ACTION	RETURN
2.	TYPELIST ≡ (TY' ∨ TY")	Put Z: = IMPLY(TY', H, C)	
2.1	Z ≡ NIL		NIL
2.2	Z ≡ (θ TY1)	Put Z2: = IMPLY(TY", H, C)	
2.3	Z2 ≡ NIL		NIL
2.4	Z2 ≡ (θ2 TY2)		(θ ∘ θ2 (TY1 ∨ TY2))
3.	H ≡ (A ∨ B)	Put Z: = IMPLY(TYPELIST, A, C)	
3.1	Z ≡ NIL		NIL
3.2	Z ≡ (θ TY1)	Put Z2: = IMPLY(TYPELIST, Bθ, C)	
3.3	Z2 ≡ NIL		NIL
3.4	Z2 ≡ (θ2 TY2)		(θ ∘ θ2 (TY1 ∨ TY2))
4.	C ≡ (A ∧ B)	Put Z: = IMPLY(TYPELIST, H, A)	
4.1	Z ≡ NIL		NIL
4.2	Z ≡ (θ TY1)	Put Z2: = IMPLY(TYPELIST, H, Bθ)	
4.3	Z2 ≡ NIL		NIL
4.4	Z2 ≡ (θ2 TY2)		(θ ∘ θ2 (TY1 ∨ TY2))
7.	C ≡ (A ⟶ B)	Put A': = SET-TYPE(A). TY' is the updated TYPELIST	
7.1	TY' has a contradiction		(T NIL)
7.2	ELSE		IMPLY(TY', H∧A', B)

---

*IMPLY has arguments (TYPELIST, H, C, TL, LT). H is the hypothesis and C the conclusion. We are ignoring TL and LT here.

Table I-T (Continued)

11. $C \equiv (a \leq b)$	Put A': = SET-TYPE(~ $(a \leq b)$) TY' is the updated TYPELIST	
11.1 TY' has a contradiction		(T NIL)
11.2 TY' ≡ TYPELIST	Go to 12	
11.3 TY' ≢ TYPELIST[3]		(T TY')

[3] If TY' has an equality entry of the form {k: t t} then k is replaced by t in H, C, and TY'.

Table II-T

TYPELIST VERSION

HOA RULE CHANGES*

	IF	ACTION	RETURN
4.	C ≡ A ∨ D	Put Z: = HOA(B ∧ ~D, A)	
4.1	Z ≡ NIL		HOA(B ∧ ~ A, D)
4.2	Z ≡ (θ TY1)	Go to 4.3.	
4.3	TY1 ≡ NIL		(θ NIL)
4.4	TY1 ≢ NIL	Put Z2: = IMPLY(TY1, B ∧ ~ A, D)	
4.5	Z2 ≡ NIL		(θ TY1)
4.6	Z2 ≡ (θ2 TY2)		(θ ∘ θ2 TY2)
6.	B ≡ A ∧ D	Put Z: = HOA(A, C)	
6.1	Z ≡ NIL		HOA(D, C)
6.2	Z ≡ (θ TY1)	Go to 6.3.	
6.3	TY1 ≡ NIL		(θ NIL)
6.4	TY1 ≢ NIL	Put Z2: = IMPLY(TY1, D, C)	
6.5	Z2 ≡ NIL		(θ TY1)[4]
6.6	Z2 ≡ (θ2 TY2)		(θ ∘ θ2 TY2)
7.	B ≡ (A ⟶ D)	Put θ: = ANDS(D, C)	
7.1	θ ≡ NIL	GO TO 7E	
7.2	θ ≢ NIL	Put Z2: = IMPLY(TYPELIST, H, Aθ)	
7.3	Z2 ≡ NIL		NIL
7.4	Z2 ≡ (θ2 TY2)		(θ ∘ θ2 TY2)

[4] In case Z2 ≡ NIL it repeats Rule 6 (once) with D ∧ A instead of A ∧ D. If on this second time Z2 = NIL then (θ TY1) is returned.

*HOA has arguments (B,C,HL). B is the hypothesis and C the conclusion. We are ignoring HL here.

Table II-T (Continued)

7E.	B ≡ (A ⟶ a = b)	Put Z: = HOA(a = b, C)	
7E.1	Z ≡ NIL	Go to 7LE	
7E.2	Z ≡ (θ TY1)	Put Z2: = IMPLY(TYPELIST, H, Aθ)	
7E.3	Z2 ≡ NIL		NIL
7E.4	Z2 ≡ (θ2 TY2)		(θ ∘ θ2 (TY1 ∨ TY2))
7LE.	B ≡ (A a ≤ b)	Put A': = SET-TYPE(a ≤ b) Let TY' be the updated TYPELIST	
7LE.1	TY' ≡ TYPELIST	Go to 8	
7LE.2	TY' ≢ TYPELIST	Put Z: = IMPLY(TY',H,C)	
7LE.3	Z ≡ NIL		NIL
7LE.4	Z ≡ (θ TY1)	Put Z2: = IMPLY(TYPELIST,H, Aθ)	
7LE.5	Z2 ≡ NIL		NIL
7LE.6	Z2 ≡ (θ2 TY2)		(θ ∘ θ2 (TY1 ∨ TY2))

The other rules of IMPLY and HOA should be changed similarly, always changing an output

$$\theta$$

to

$$(\theta \text{ NIL}) \quad .$$

These changes are best explained by the use of examples.

In the following proofs, the theorem label (X h1) is used to indicate that the first hypothesis is being used to try to prove the subgoal (X). Similarly for (X h2), etc. Also the label (X h2 H) is used to indicate that, after backchaining on the second hypothesis (see Rule H 7), it is now trying to prove the hypothesis of the second hypothesis, etc.

<u>Ex. 8</u>. $(k \leq 3 \wedge \underset{\alpha}{(k \leq 1 \longrightarrow C)} \wedge \underset{\beta}{(2 \leq k \leq 3 \longrightarrow C)} \longrightarrow C)$

(1)	$(\{k:\ 0\ 3\} \wedge (k \leq 1 \longrightarrow C) \wedge (2 \leq k \wedge k \leq 3 \longrightarrow C) \Rightarrow C)$	I 7
(1 h1)	$(\{k:\ 0\ 3\} \wedge (k \leq 1 \longrightarrow C) \Rightarrow C)$	H 6
(1 h1 H)	$(\{k:\ 0\ 3\} \wedge \alpha \wedge \beta \Rightarrow k \leq 1)$	H 7, 7.2
	SET-TYPE$(\sim(k \leq 1))$, $2 \leq k$	I 11
	TY' = {k: 2 3}, has no contradiction.	
	Returns (T {k: 2 3}) for (1 h1 H)	I 11.3
	and for (1 h1)	H 7.4
(1 h2)	$(\{k:\ 2\ 3\} \wedge \beta \Rightarrow C)$	H 6.4
(1 h2 H)	$(\{k:\ 2\ 3\} \wedge \alpha \wedge \beta \Rightarrow 2 \leq k \wedge k \leq 3)$	H 7, 7.2
(1 h2 H1)	$(\{k:\ 2\ 3\} \wedge \alpha \wedge \beta \Rightarrow 2 \leq k)$	I 4
	SET-TYPE$(\sim(2 \leq k))$, $k \leq 1$	I 11
	TY' = {k: 2 1}, has a contradiction	
	Returns (T NIL)	I 11.1
(1 h2 H2)	$(\{k:\ 2\ 3\} \wedge \alpha \wedge \beta \Rightarrow k \leq 3)$	I 4.2
	SET-TYPE$(\sim(k \leq 3))$, $4 \leq k$	
	TY' = {k: 4 3}, has a contradiction.	
	Returns (T NIL)	I 11.1
	Returns (T NIL) for (1 h2 H)	I 4.2
	Returns (T NIL) for (1 h2)	H 7.4
	Returns (T NIL) for (1)	H 6.6

Thus the theorem is true.

Ex. 9. $(1 \leq n)$

$$\wedge \forall m(2 \leq n \wedge 1 \leq m \wedge m \leq 1 \longrightarrow A[m] \leq A[2])$$

$$\wedge \forall k(k \leq n \wedge 2 \leq k \longrightarrow A[k] \leq A[k+1])$$

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

(1) $$(1 \leq n \wedge \overbrace{(2 \leq n \wedge 1 \leq m \wedge m \leq 1 \longrightarrow A[m] \leq A[2])}^{\alpha}$$
$$\wedge \overbrace{(k \leq n \wedge 2 \leq k \longrightarrow A[k] \leq A[k+1])}^{\beta}$$
$$\longrightarrow (K \leq n \wedge 1 \leq K \longrightarrow \overbrace{A[K] \leq A[K+1])}^{\gamma})$$

n and K are skolem constants

(1) $(\overbrace{[\{K:\ 1\ n\}\ \ \{n:\ K\ \ \infty\}]}^{TY} \wedge \alpha \wedge \beta \Rightarrow A[K] \leq A[K+1])$ I 7

(1 h1) $(\alpha \Rightarrow \gamma)$ Returns NIL H 6

(1 h2) $(\beta \Rightarrow \gamma)$ H 6.1

$(A[k] \leq A[k+1] \longrightarrow A[K] \leq A[K+1])$, $\{K/k\}$ H 7

(1 h2 H) $(TY \wedge \alpha \wedge \beta \Rightarrow K \leq n \wedge 2 \leq K)$ H 7.2

(1 h2 H1) $(TY \wedge \alpha \wedge \beta \Rightarrow K \leq n)$ I 4

SET-TYPE($\sim(K \leq n)$), $n \leq K - 1$ I 11

TY' = $[\{K{:}n+1\ n\}\ \{n:\ K\ K-1\}]$,

has a contradiction, so returns (T NIL) I 11.1

(1 h2 H2) $(TY \wedge \alpha \wedge \beta \Rightarrow 2 \leq K)$ I 4.2

SET-TYPE ($\sim(2 \leq K)$), $K \leq 1$ I 11

TY" = $[\{K:\ 1\ \min(1,n)\}\{n:\ K\ \infty\}]$

TY" = $[\{K:\ 1\ 1\}\{n:\ K\ \infty\}]$

Here min(1,n) is converted automatically to 1, because it deduces that

$$n \geq K \geq 1 \quad .$$

TY'' has no contradiction but the program detects {K: 1 1} in TY'' and therefore replaces K by 1 in H, C, and TY'', (and in γ for (1 h1) below). Thus $(A[K] \leq A[K+1])$ becomes $(A[1] \leq A[2])$ and TY'' becomes

$$TY''' = [\{K: 1\ 1\}\{n: 1\ \infty\}] \quad .$$

It then returns (T TY''') for (1 h2 H2). I 11.3
It then returns (T TY''') for (1 h2 H) . H 7.4
It then returns (K/k TY''') for (1 h2) . I 4.4

(1 h1) $(TY''' \wedge \alpha \Rightarrow A[1] \leq A[2])$ H 6.4 and Footnote 4

$(A[m] \leq A[2] \Rightarrow A[1] \leq A[2])$, 1/m H 7

(1 h1 H) $(TY''' \wedge \alpha \wedge \beta \Rightarrow 2 \leq n \wedge 1 \leq 1 \wedge 1 \leq 1)$ H 7.2

(1 h1 H1) $(TY''' \wedge \alpha \wedge \beta \Rightarrow 2 \leq n)$ I 4
SET-TYPE$(\sim(2 \leq n))$, $n \leq 1$ I 11
TY'' = [{K: 11}{n: 1 1}]

Replaces n by 1 throughout and I 11.1
Returns (T NIL) for (1 h1 H1)

(1 h1 H2) $(TY''' \wedge \alpha \wedge \beta \Rightarrow 1 \leq 1 \wedge 1 \leq 1)$
Returns (T NIL) by REDUCE
Returns (T NIL) for (1 h1 H) H 7.4
Returns (1/m NIL) for (1 h1) H 4.4.3
Returns ((K/k 1/m)NIL) H 6.6

Thus the theorem is true.

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.

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

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

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

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

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

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

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

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

(1)	({K: 0 j}{j: K ∞} $\wedge\ \alpha \wedge \beta \wedge \gamma \Rightarrow A[K] \leq A[K+1]$)	I 7
(1 h1)	($\alpha \longrightarrow A[K] \leq A[K+1]$) K/k	H 6
(1 h1 H)	({K: 0 j}{j: K ∞} $\wedge\ \alpha \wedge \beta \wedge \gamma \Rightarrow K \leq 2$)	H 7, 7.2
	SET-TYPE(~($K \leq 2$)), $3 \leq K$	I 11
	TY' = [{K: 3 j}{j: K ∞}], 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(~($3 \leq K$)), $K \leq 2$	I 11
	TY' = [{K: 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(~(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'' $\wedge$ γ $\Rightarrow$ A[K] $\leq$ A[K+1]) K/n	H 6.4
(1 h2 h2 H)	(TY'' $\wedge$ γ $\Rightarrow$ 6 $\leq$ K $\wedge$ K $\leq$ j)	H 7, 7.2
(1 h2 h2 H1)	(TY'' $\wedge$ γ $\Rightarrow$ 6 $\leq$ K)	I 4
	SET-TYPE(~(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'' $\wedge$ γ $\Rightarrow$ K $\leq$ j)	I 4.2
	SET-TYPE(~(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,

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

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

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. $\{x_o: 0 \quad 5\}$
2. $\{x_o: 2 \quad \infty\} \vee C$
3. $\{x_o: 0 \quad 1\} \vee \{x_o: 8 \quad \infty\} \vee C$
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

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

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

$$\text{INTERSECT}(\{x_o: 0 \quad 5\}, \{x_o: 2 \quad \infty\} = \{x_o: 2 \quad 5\},$$

the resolvent of 1 and 2 is

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

Similarly we get

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

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

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

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

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.	$\{k:\ 3\quad \infty\} \vee A[k] \leq A[k+1]$	
2.	$\{m:\ 0\quad 2\} \vee \{m:\ 8\quad \infty\} \vee A[m] \leq A[m+1]$	
3.	$\{n:\ 0\quad 5\} \vee [\{n:\ j_o+1\quad \infty\}\{j_o:\ 0\ n-1\}] \vee A[n] \leq A[n+1]$	
4.	$(\{K_o:\ 0\quad j_o\}\quad \{j_o:\ K_o\quad \infty\})$	
5.	$\sim(A[K_o] \leq A[K_o+1])$	

Some of the resolvents of 1-5 are

6.	$\{K_o:\ 3\quad \infty\}$	1, 5
7.	$\{K_o:\ 0\quad 2\} \vee \{K_o:\ 8\quad \infty\}$	2, 5
8.	$\{K_o:\ 0\quad 5\} \vee [\{K_o:\ j_o+1\quad \infty\}\{j_o:\ 0\ K_o-1\}]$	3, 5
9.	$\{K_o:\ 3\quad 2\} \vee \{K_o:\ 8\quad \infty\}$	6, 7
10.	$\{K_o:\ 8\quad 5\} \vee [\{K_o:\ j_o+1\quad \infty\}\{j_o:\ 0\ K_o-1\}]$	8, 9
11.	$(\{K_o:\ j_o+1\quad j_o\}\quad \{j_o:\ K_o\quad K_o-1\})$ 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 not convert the formula $A[k] \leq A[k+1]$ to typelist form

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

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

One could allow all inequalities to be converted, but in that case a mechanism would need to be provided for unifying expressions when two typelist 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 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 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($H \wedge A$,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 $\sim(P \rightarrow Q)$ goes to $P \wedge \sim Q$. If D contains no "~" or "$\longrightarrow$" 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' ≡ 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. $\square$ 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. $\sim[\sim(a \leq b) \wedge TY] \leftrightarrow [a \leq b \vee \sim TY]$

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

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

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. $\square$ 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. $\sim A \vee D$

8. $\sim C$

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. □ 9,15

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

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

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

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

Proof. Obvious

Rule H-T 6.6. 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 $(TY \wedge A \wedge D \Rightarrow C)$.

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

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

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

Proof. 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 $a \quad b$.

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

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

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

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. $\sim A \vee D$

9. $\sim C$

10.	$\sim D$	1,4,6,7,9
11.	$A\lambda \vee A\theta\lambda$	2,5,6,7
12.	$\sim D \vee A\theta\lambda$	3,5,6,7
13.	$\sim A$	10,8
14.	$\square$	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."

References

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. Third Int. Joint Conf. Artif. Intell., 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. Artif. Intell., vol. 5, no. 3, pp. 235-316, Fall 1974; also in Conf. Rec. Ass. Comput. Mach. Symp. Principles of Programming Languages, 1973, pp. 169-182.

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

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. Mach. Intell. 6. 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 IEEE Trans. on Software Engineering 1(1975), pp. 59-67.

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

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. Computer J. 12, 1(Feb. 1969), pp. 41-48.

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