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 occurring 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 I, 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 \( j \times k \geq 0 \), \( j \leq k - 1 \), 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 \([j: k \ k-1]\) 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 \((j \leq N+1)\) 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

**Ex. 1.**

\[ (1) \quad (P(1) \land 1 \leq j \land j \leq n \land j \leq 1 \rightarrow P(j)) \]

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 \text{SET-TYPE} is called on the hypothesis of (1) which changes
\text{TYPELIST} to

\[
([j: 1 \: 1][n:j \infty])
\]

and converts (1) to

(2) \hspace{1cm}
\[ j = 1 \land P(1) \rightarrow P(j) \]

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

\[
(P(1) \rightarrow P(1))
\]

which it recognizes as true.

Other examples are now given.

\text{Ex. 2.}

(3) \hspace{1cm}
\[ 1 \leq j \land P(1) \rightarrow (j \leq k \land k \leq 1 \rightarrow P(k)) \]

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

(4) \hspace{1cm}
\[ P(1) \rightarrow (j \leq k \land k \leq 1 \rightarrow P(k)) \]

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

(5) \hspace{1cm}
\[ (P(1) \land j \leq k \land k \leq 1 \rightarrow P(k)) \]

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

\[
(j = 1 \land k = 1 \land P(1) \rightarrow P(k))
\]
The prover, as before, converts this to

\[(P(1) \rightarrow P(1))\]

which it recognizes as true.

Ex. 3.

\[(2 \leq j \land j \leq 1 \rightarrow P(j))\]

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.

\[(6) \quad (j \leq 1 \land k \leq j \land P \rightarrow k \leq 3)\ ,\]

TYPELIST is given the value \([(j: k 1)[k: 0 j])\) and (6) is converted to

\[(P \rightarrow k \leq 3)\ .\]

The prover now uses \((k \leq 3)\), which is first converted to \((4 \leq k)^2\), to update TYPELIST, getting \([(j: k 1)[k: 4 j])\), which contains the contradiction

\[(4 \leq k \leq j \leq 1)\ .\]

\(^2\)Since \(k\) is an integer. See [7, p. 27].
The prover detects such contradictions by computing absolute upper and lower bounds, \( \text{sup} \) and \( \text{inf} \), for \( j \) and \( k \). For this case

\[
\begin{align*}
\text{sup } j &= 1, \quad \text{inf } j = 4 \\
\text{sup } k &= 1, \quad \text{inf } k = 4.
\end{align*}
\]

Since \( 4 > 1 \) we have a contradiction. The prover uses the routines \( \text{SUP} \) and \( \text{INF} \) to evaluate these bounds. In [7] we carefully define the algorithms \( \text{SUP} \) and \( \text{INF} \) 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].

\textbf{Ex. 5.} \( (2 \leq j \leq 4 \land k \leq j \land k \leq 7 \rightarrow C) \). Here we use the symbols 'max' and 'min' in typing \( j \) and \( k \). \textsc{Typelist} is given the value

\[
[[j: \text{max}(2,k) 4][k: 0 \text{ min}(j,7)]].
\]
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

\[ \text{IMPLY}(\text{TYPELIST, H, C, TL, LT}) \] ,

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

\[ (\text{TL}) \quad (H \Rightarrow C) \] .

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

\[ (\text{TL}) \quad ([\text{TYPELIST}] \land H \Rightarrow C) \] .

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

\[ (1) \quad ([\{j: 1 1\}[k: 1 1]] \land P(1) \Rightarrow P(k)) \] .

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

\[
\text{SET-TYPE}(k \leq 5 \land P(j))
\]

updates TYPELIST to

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

and returns the value P(j).
### IMPLY RULE CHANGES

<table>
<thead>
<tr>
<th>IF</th>
<th>ACTION</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.</td>
<td>C ≡ (A → B) is changed to</td>
<td>IMPLY(H ∧ A, B)</td>
</tr>
<tr>
<td>7.</td>
<td>C ≡ (A → B) Put A': = SET-TYPE(A)</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>TY' has a contradiction</td>
<td>&quot;T&quot;</td>
</tr>
<tr>
<td>7.2</td>
<td>ELSE</td>
<td>IMPLY(TY', H ∧ A', B)</td>
</tr>
</tbody>
</table>

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

Rule 11 and 14 are added to IMPLY

| 11. | C ≡ (a ≤ b) Put A': = SET-TYPE(¬(a ≤ b)) Let TY' be the updated TYPELIST |
| 11.1 | TY' has a contradiction | "T" |
| 11.2 | TY' = TYPELIST Go to 12 (with TYPELIST and C as they were) | |
| 11.3 | TY' ≠ TYPELIST | (T TY') |
| 14.1 | C ≡ (a = b) Put C' ≡ (a ≤ b ∧ b ≤ a) | IMPLY(H, C') |
| 14.2 | C ≡ (a ≠ b) Put C' ≡ (a < b ∨ 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.** \((Q \rightarrow (j \leq 1 \land k \leq j \land P \rightarrow k \leq 3))\)

\((1) \quad \{[j: 0 \infty][k: 0 \infty]\}
\Rightarrow (Q \rightarrow (j \leq 1 \land k \leq j \land P \rightarrow k \leq 3))\)

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

\((1) \quad \{[j: 0 \infty][k: 0 \infty]\} \land Q
\Rightarrow (j \leq 1 \land k \leq j \land P \rightarrow k \leq 3))\)

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

\[\text{TYPELIST} \quad \text{H} \quad \text{C}\]
\((1) \quad \{[j: k 1][k: 0 j]\} \land (Q \land P \rightarrow k \leq 3)\)

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

Now the new Rule I-11, employs \(\text{SET-TYPE}((\neg (k \leq 3)) = \text{SET-TYPE}(4 \leq k)\) to update \(\text{TYPELIST}\) to \(\text{TY'} = \{[j: k 1][k: 4 j]\}\), 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

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

(or any other list of inequalities) by computing

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

In this case

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

and since $4 > 1$ we have a contradiction. These are computed by the algorithms $\text{SUP}$ and $\text{INF}$ (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 one interval $[j: a b]$ in TYPELIST. So if we are proving a goal of the form

$$((j \leq 1 \lor j \geq 5) \land H \Rightarrow C),$$

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

$$\mathbb{[}j: 0 1]\mathbb{]}[k: \ldots] \lor \mathbb{[}j: 5 \infty]\mathbb{]}[k: \ldots]$$

$$\land H \Rightarrow C).$$

To handle such examples we add Rule 2 to $\text{IMPLY}$ to split such goals into
two subgoals.

2. $\text{TYPELIST} \equiv \text{TY}' \lor \text{TY}''$
   Put $\Theta = \text{IMP}(\text{TY}', H, C)$

2.1 $\Theta = \text{NIL}$

2.2 $\Theta \neq \text{NIL}$
   Put $\lambda = \text{IMP}(\text{TY}'', H, C)$

2.3 $\lambda = \text{NIL}$

2.4 $\lambda \neq \text{NIL}$
   $\sigma \circ \lambda$

Ex. 7. $(k \leq 3 \rightarrow k \leq 1 \lor 2 \leq k \leq 3)$.

(1) $([k: 0 \infty] \Rightarrow (k \leq 3 \rightarrow k \leq 1 \lor 2 \leq k \leq 3))$

(1)

(1) $([k: 0 \infty] \Rightarrow (k \leq 1 \lor 2 \leq k \leq 3))$

(1) $([k: 0 \infty] \Rightarrow k \leq 1 \lor 2 \leq k \leq 3)$

(1) $([k: 0 \infty] \land \neg(2 \leq k \leq 3) \Rightarrow k \leq 1)$

(1) $([k: 0 \infty] \land (k \leq 1 \lor 4 \leq k) \Rightarrow k \leq 1)$

(1) $([k: 0 \infty] \lor \{k: 4 \infty\}) \Rightarrow k \leq 1)$

(1) $([k: 0 \infty] \Rightarrow k \leq 1)$

Rule 10' uses $\neg(k \leq 1)$ to update $\text{TYPELIST}$ to $\{k: 2 \infty\}$ and Rule 10.2

detects the contradiction.

(1 2) $([k: 4 \infty] \Rightarrow k \leq 1)$

Proved since $\{k: 4 \infty\}$ 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.** \((k \leq 3 \land (k \leq 1 \rightarrow C) \land (2 \leq k \leq 3 \rightarrow C) \rightarrow C)\).

(1) \hspace{1cm} ([k: 0 \ 3] \land (k \leq 1 \rightarrow C) \land (2 \leq k \leq 3 \rightarrow C) \Rightarrow C) \hspace{1cm} 17

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

(1 H) \hspace{1cm} ([k: 0 \ 3] \land (k \leq 1 \rightarrow C) \land (2 \leq k \leq 3 \rightarrow C) \rightarrow k \leq 1)

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

\((k \leq 1 \lor 2 \leq k \leq 3)\),
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

**Ex. 9.** \( (1 \leq n) \)
\[ \wedge \forall m \ (2 \leq n \wedge 1 \leq m \wedge m \leq 1 \rightarrow A[m] \leq A[2]) \]
\[ \wedge \forall k \ (k+1 \leq n \wedge 2 \leq k \rightarrow A[k] \leq A[k+1]) \]
\[ \implies \forall K (K+1 \leq n \wedge 1 \leq K \rightarrow A[K] \leq A[K+1]) \]

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

\[(\emptyset \ TY')\]

where \( \emptyset \) 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 outputted value \( TY' \) represents the part of the theorem that has not been proved. Thus if \((\emptyset \ TY')\) is returned from a call IMPLY (TYPELIST, H, C), it means that (TYPELIST \( \wedge \ H \rightarrow C\)) is valid except for the case \( TY' \), or that

\[ (~ TY' \wedge \text{TYPELIST} \wedge H \rightarrow C) \]

is valid. See Appendix 2.
<table>
<thead>
<tr>
<th>IF</th>
<th>ACTION</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. TYPELIST \equiv (TY' \lor TY'')</td>
<td>Put Z := IMPLY(TY', H, C)</td>
<td>NIL</td>
</tr>
<tr>
<td>2.1 Z \equiv NIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2 Z \equiv (\Theta TY1)</td>
<td>Put Z2 := IMPLY(TY'', H, C)</td>
<td>NIL</td>
</tr>
<tr>
<td>2.3 Z2 \equiv NIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4 Z2 \equiv (\Theta2 TY2)</td>
<td></td>
<td>(\Theta \circ \Theta2 (TY1 \lor TY2))</td>
</tr>
<tr>
<td>3. H \equiv (A \lor B)</td>
<td>Put Z := IMPLY(TYPELIST, A, C)</td>
<td>NIL</td>
</tr>
<tr>
<td>3.1 Z \equiv NIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2 Z \equiv (\Theta TY1)</td>
<td>Put Z2 := IMPLY(TYPELIST, B\Theta, C)</td>
<td>NIL</td>
</tr>
<tr>
<td>3.3 Z2 \equiv NIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4 Z2 \equiv (\Theta2 TY2)</td>
<td></td>
<td>(\Theta \circ \Theta2 (TY1 \lor TY2))</td>
</tr>
<tr>
<td>4. C \equiv (A \land B)</td>
<td>Put Z := IMPLY(TYPELIST, H, A)</td>
<td>NIL</td>
</tr>
<tr>
<td>4.1 Z \equiv NIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2 Z \equiv (\Theta TY1)</td>
<td>Put Z2 := IMPLY(TYPELIST, H, B\Theta)</td>
<td>NIL</td>
</tr>
<tr>
<td>4.3 Z2 \equiv NIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4 Z2 \equiv (\Theta2 TY2)</td>
<td></td>
<td>(\Theta \circ \Theta2 (TY1 \lor TY2))</td>
</tr>
<tr>
<td>7. C \equiv (A \rightarrow B)</td>
<td>Put A' := SET-TYPE(A). Ty' is the updated TYPELIST</td>
<td>(T NIL)</td>
</tr>
<tr>
<td>7.1 TY' has a contradiction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.2 ELSE</td>
<td>IMPLY(TY', H \land A', B)</td>
<td></td>
</tr>
</tbody>
</table>

*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. $C = (a \leq b)$

Put $A' = \text{SET-TYPE}(\neg (a \leq b))$
$TY'$ is the updated TYPELIST

11.1 $TY'$ has a contradiction (T NIL)

11.2 $TY' = \text{TYPELIST}$
Go to 12

11.3 $TY' \neq \text{TYPELIST}^3$ (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>
<thead>
<tr>
<th>IF</th>
<th>ACTION</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C \equiv A \lor D$</td>
<td>Put $Z := \text{HOA}(B \land \neg D, A)$</td>
<td>$\text{HOA}(B \land \neg A, D)$</td>
</tr>
<tr>
<td>$Z \equiv \text{NIL}$</td>
<td></td>
<td>$\text{HOA}(B \land \neg A, D)$</td>
</tr>
<tr>
<td>$Z \equiv (\emptyset \text{TY1})$</td>
<td>Go to 4.3.</td>
<td>($\emptyset \text{NIL}$)</td>
</tr>
<tr>
<td>$\text{TY1} \equiv \text{NIL}$</td>
<td></td>
<td>($\emptyset \text{TY1}$)</td>
</tr>
<tr>
<td>$\text{TY1} \neq \text{NIL}$</td>
<td>Put $Z_2 := \text{IMPLY(\text{TY1}, B \land \neg A, D)}$</td>
<td>($\emptyset \text{TY1}$)</td>
</tr>
<tr>
<td>$Z_2 \equiv \text{NIL}$</td>
<td></td>
<td>($\emptyset \text{TY1}$)</td>
</tr>
<tr>
<td>$Z_2 \equiv (\emptyset_2 \text{TY2})$</td>
<td></td>
<td>($\emptyset \circ \emptyset_2 \text{TY2}$)</td>
</tr>
<tr>
<td>$B \equiv A \land D$</td>
<td>Put $Z := \text{HOA}(A, C)$</td>
<td>$\text{HOA}(D, C)$</td>
</tr>
<tr>
<td>$Z \equiv \text{NIL}$</td>
<td></td>
<td>$\text{HOA}(D, C)$</td>
</tr>
<tr>
<td>$Z \equiv (\emptyset \text{TY1})$</td>
<td>Go to 6.3.</td>
<td>($\emptyset \text{NIL}$)</td>
</tr>
<tr>
<td>$\text{TY1} \equiv \text{NIL}$</td>
<td></td>
<td>($\emptyset \text{NIL}$)</td>
</tr>
<tr>
<td>$\text{TY1} \neq \text{NIL}$</td>
<td>Put $Z_2 := \text{IMPLY(\text{TY1}, D, C)}$</td>
<td>($\emptyset \text{TY1})^4$</td>
</tr>
<tr>
<td>$Z_2 \equiv \text{NIL}$</td>
<td></td>
<td>($\emptyset \text{TY1})^4$</td>
</tr>
<tr>
<td>$Z_2 \equiv (\emptyset_2 \text{TY2})$</td>
<td></td>
<td>($\emptyset \circ \emptyset_2 \text{TY2}$)</td>
</tr>
<tr>
<td>$B \equiv (A \rightarrow D)$</td>
<td>Put $\Theta := \text{ANDS(D, C)}$</td>
<td>$\text{NIL}$</td>
</tr>
<tr>
<td>$\Theta \equiv \text{NIL}$</td>
<td>Go to 7E</td>
<td>$\text{NIL}$</td>
</tr>
<tr>
<td>$\Theta \neq \text{NIL}$</td>
<td>Put $Z_2 := \text{IMPLY(\text{TYPELIST, H, A\Theta})}$</td>
<td>($\emptyset \circ \emptyset_2 \text{TY2}$)</td>
</tr>
<tr>
<td>$Z_2 \equiv \text{NIL}$</td>
<td></td>
<td>($\emptyset \circ \emptyset_2 \text{TY2}$)</td>
</tr>
<tr>
<td>$Z_2 \equiv (\emptyset_2 \text{TY2})$</td>
<td></td>
<td>($\emptyset \circ \emptyset_2 \text{TY2}$)</td>
</tr>
</tbody>
</table>

\(^4\text{In case } Z_2 = \text{NIL it repeats Rule 6 (once) with } D \land A \text{ instead of } A \land D.\)

\(^*\text{HOA has arguments } (B, C, \text{HL}). \text{ B is the hypothesis and C the conclusion. We are ignoring HL here.}\)
Table II-T (Continued)

7E. \( B \equiv (A \rightarrow a = b) \)  
Put \( Z_1 = \text{HOA}(a = b, C) \)

7E.1 \( Z \equiv \text{NIL} \)  
Go to 7LE

7E.2 \( Z \equiv (\emptyset \ \text{TY1}) \)  
Put \( Z_2 \equiv \text{IMPLY}(\text{TYPELIST}, H, A\emptyset) \)

7E.3 \( Z_2 \equiv \text{NIL} \)  

7E.4 \( Z_2 \equiv (\emptyset_2 \ \text{TY2}) \)  
NIL

(\( \emptyset \circ \emptyset_2 (\text{TY1} \lor \text{TY2}) \))

7E. \( B \equiv (A \ a \leq b) \)  
Put \( A' \equiv \text{SET-TYPE}(a \leq b) \)  
Let \( \text{TY}' \) be the updated \( \text{TYPELIST} \)

7LE.1 \( \text{TY}' \equiv \text{TYPELIST} \)  
Go to 8

7LE.2 \( \text{TY}' \neq \text{TYPELIST} \)  
Put \( Z_1 \equiv \text{IMPLY}(\text{TY}', H, C) \)

7LE.3 \( Z \equiv \text{NIL} \)  
NIL

7LE.4 \( Z \equiv (\emptyset \ \text{TY1}) \)  
Put \( Z_2 \equiv \text{IMPLY}(\text{TYPELIST}, H, A\emptyset) \)

7LE.5 \( Z_2 \equiv \text{NIL} \)  
NIL

7LE.6 \( Z_2 \equiv (\emptyset_2 \ \text{TY2}) \)  
NIL

(\( \emptyset \circ \emptyset_2 (\text{TY1} \lor \text{TY2}) \))
The other rules of IMPLY and HOA should be changed similarly, always changing an output

\[ \emptyset \]

to

\[ (\emptyset \text{ NIL}) \]

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 H7), it is now trying to prove the hypothesis of the second hypothesis, etc.
Ex. 8. \((k \leq 3 \land (k \leq 1 \rightarrow C) \land (2 \leq k \leq 3 \rightarrow C) \rightarrow C)\)

(1) \((\{k: 0 3\} \land (k \leq 1 \rightarrow C) \land (2 \leq k \land k \leq 3 \rightarrow C) \Rightarrow C)\)  

(1 h1) \((\{k: 0 3\} \land (k \leq 1 \rightarrow C) \Rightarrow C)\)  

(1 h1 H) \((\{k: 0 3\} \land \alpha \land \beta \Rightarrow k \leq 1)\)

SET-TYPE(\((\neg(k \leq 1))\), \(2 \leq k\))  

TY' = \(\{k: 2 3\}\), has no contradiction.  

Returns \((T \{k: 2 3\})\) for (1 h1 H)  

and for (1 h1)  

(1 h2) \((\{k: 2 3\} \land \beta \Rightarrow C)\)  

(1 h2 H) \((\{k: 2 3\} \land \alpha \land \beta \Rightarrow 2 \leq k \land k \leq 3)\)  

(1 h2 H1) \((\{k: 2 3\} \land \alpha \land \beta \Rightarrow 2 \leq k)\)

SET-TYPE(\((\neg(2 \leq k))\), \(k \leq 1\))  

TY' = \(\{k: 2 1\}\), has a contradiction  

Returns \((T \text{NIL})\)  

(1 h2 H2) \((\{k: 2 3\} \land \alpha \land \beta \Rightarrow k \leq 3)\)

SET-TYPE(\((\neg(k \leq 3))\), \(4 \leq k\))  

TY' = \(\{k: 4 3\}\), has a contradiction.  

Returns \((T \text{NIL})\)  

Returns \((T \text{NIL})\) for (1 h2 H)  

Returns \((T \text{NIL})\) for (1 h2)  

Returns \((T \text{NIL})\) for (1)  

Thus the theorem is true.
Ex. 9. (1 ≤ n)

\[ \land \forall m(2 \leq n \land 1 \leq m \land m \leq 1 \rightarrow A[m] \leq A[2]) \]
\[ \land \forall k(k \leq n \land 2 \leq k \rightarrow A[k] \leq A[k+1]) \]
\[ \rightarrow \forall K (K \leq n \land 1 \leq K \rightarrow A[K] \leq A[K+1]) \]

\[ \text{(1)} \]
\[ (1 \leq n \land (2 \leq n \land 1 \leq m \land m \leq 1 \rightarrow A[m] \leq A[2]) \land (k \leq n \land 2 \leq k \rightarrow A[k] \leq A[k+1]) \rightarrow (K \leq n \land 1 \leq K \rightarrow A[K] \leq A[K+1])) \]

\text{n and K are skolem constants}

\[ \text{TY} \]
\[ \text{(1)} \]
\[ ([\{K: 1 \ n\} \ {n: K \infty}] \land \alpha \land \beta \Rightarrow A[K] \leq A[K+1]) \]

\[ \text{I 7} \]

\[ \text{(1 h1)} \]
\[ (\alpha \Rightarrow \gamma) \]
\[ \text{Returns NIL} \]

\[ \text{H 6} \]

\[ \text{(1 h2)} \]
\[ (\beta \Rightarrow \gamma) \]
\[ (A[k] \leq A[k+1] \rightarrow A[K] \leq A[K+1]), \{K/k\} \]

\[ \text{H 7} \]

\[ \text{(1 h2 H)} \]
\[ (\text{TY} \land \alpha \land \beta \Rightarrow K \leq n \land 2 \leq K) \]

\[ \text{H 7.2} \]

\[ \text{(1 h2 H1)} \]
\[ (\text{TY} \land \alpha \land \beta \Rightarrow K \leq n) \]
\[ \text{SET-TYPE}(\sim(K \leq n)), \ n \leq K - 1 \]
\[ \text{TY'} = [\{K:n+1 \ n\} \ {n: K \ K - 1}] , \]
\[ \text{has a contradiction, so returns (T NIL)} \]

\[ \text{I 11,1} \]

\[ \text{(1 h2 H2)} \]
\[ (\text{TY} \land \alpha \land \beta \Rightarrow 2 \leq K) \]
\[ \text{SET-TYPE} \ (\sim(2 \leq K)), \ K \leq 1 \]
\[ \text{TY''} = [\{K: 1 \ \text{min}(1,n)\}\{n: K \infty\}] \]
\[ \text{TY''} = [\{K: 1 \ 1\}\{n: K \infty\}] \]

\[ \text{I 4.2} \]

\[ \text{I 11} \]
Here \( \min(1,n) \) is converted automatically to 1, because it deduces that

\[
n \geq K \geq 1.
\]

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

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

It then returns \((T \text{TY'''} )\) for \((1 \ h2 \ H2)\).
It then returns \((T \text{TY'''} )\) for \((1 \ h2 \ H)\).
It then returns \((K/k \text{TY'''} )\) for \((1 \ h2)\).

\[(1 \ h1) \quad (\text{TY'''} \land \alpha \Rightarrow A[1] \leq A[2])\]

\[
\]

\[(1 \ h1 \ H) \quad (\text{TY'''} \land \alpha \land \beta \Rightarrow 2 \leq n \land 1 \leq 1 \land 1 \leq 1)\]

\[(1 \ h1 \ H1) \quad (\text{TY'''} \land \alpha \land \beta \Rightarrow 2 \leq n)\]

\[
\text{SET-TYPE}(\sim(2 \leq n)), \quad n \leq 1
\]

\[
\text{TY''} = \{[K: 1 1][n: 1 1]\}
\]

Replaces \( n \) by 1 throughout and

Returns \((T \text{NIL})\) for \((1 \ h1 \ H1)\).

\[(1 \ h1 \ H2) \quad (\text{TY'''} \land \alpha \land \beta \Rightarrow 1 \leq 1 \land 1 \leq 1)\]

Returns \((T \text{NIL})\) by \texttt{REDUCE}

Returns \((T \text{NIL})\) for \((1 \ h1 \ H)\)

Returns \((1/m \text{NIL})\) for \((1 \ h1)\)

Returns \((K/k \ 1/m)\text{NIL})\)

Thus the theorem is true.
It can be seen from these examples that the new TYPELIST $TY'$ which is returned as

$$(\emptyset \, TY')$$

represents the cases that have not been proved 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 $(\emptyset \, TY')$, then the theorem has not been completely proved. Hence the final return from IMPLY (for the original theorem itself) must be of the form

$$(\emptyset \, NIL)$$

Else the theorem is considered not to be proved.
Ex. 10. \( \forall k (k \leq 2 \rightarrow A[k] \leq A[k+1]) \)
\( \land \forall m (3 \leq m \leq 7 \rightarrow A[m] \leq A[m+1]) \)
\( \land \forall n (6 \leq n \leq j \rightarrow A[n] \leq A[n+1]) \)

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

\( \alpha \)

\( (1) \quad (k \leq 2 \rightarrow A[k] \leq A[k+1]) \)

\( \beta \)

\( (3 \leq m \leq 7 \rightarrow A[m] \leq A[m+1]) \)

\( \gamma \)

\( (6 \leq n \leq j \rightarrow A[n] \leq A[n+1]) \)

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

\( (1) \quad ([K: 0 j][j: K \infty] \land \alpha \land \beta \land \gamma \Rightarrow A[K] \leq A[K+1]) \)

\( (1 \ h1) \quad (\alpha \rightarrow A[K] \leq A[K+1]) \quad K/k \)

\( (1 \ h1 \ H) \quad ([K: 0 j][j: K \infty] \land \alpha \land \beta \land \gamma \Rightarrow K \leq 2) \)

\[ \text{SET-TYPE}(\neg(K \leq 2)), \ 3 \leq K \]

\( \text{TY}' = [[K: 3 j][j: K \infty]], \text{ has no contradiction} \)

\( \text{Returns } (T \text{ TY}') \)

\[ \text{Returns } (K/k \text{ TY}') \text{ for } (1 \ h1). \]

\( (1 \ h2) \quad (\text{TY}' \land (\beta \land \gamma) \Rightarrow A[K] \leq A[K+1]) \)

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

\( (1 \ h2 \ h1 \ H) \quad (\text{TY}' \land \beta \land \gamma \Rightarrow 3 \leq K \land K \leq 7) \)

\( (1 \ h2 \ h1 \ H1) \quad (\text{TY}' \land (\beta \land \gamma) \Rightarrow 3 \leq K) \)

\[ \text{SET-TYPE}(\neg(3 \leq K)), \ K \leq 2 \]

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

\( \text{Returns } (T \text{ NIL}) \)
(1 h2 h1 H2) \quad (TY' \land (\beta \land \gamma) \Rightarrow K \leq 7) \quad I 4.2

\quad \text{SET-TYPE}(\neg(K \leq 7)), \quad 8 \leq K \quad I 11

\quad TY'' = \{[K: 8 j][j: K \infty]\}, \text{has no contradiction} \quad I 11.3

\quad \text{Returns} \quad (T \ TY'') \quad I 4.4

\quad \text{Returns} \quad (T \ TY'') \text{ for } (1 \ h2 \ h1 \ H) \quad H 7.4

\quad \text{Returns} \quad (K/m \ TY'') \text{ for } (1 \ h2 \ h1) \quad H 6.4

(1 h2 h2) \quad (TY'' \land \gamma \Rightarrow A[K] \leq A[K + 1]) \quad K/n \quad H 6.4

(1 h2 h2 H) \quad (TY'' \land \gamma \Rightarrow 6 \leq K \land K \leq j) \quad H 7, 7.2

(1 h2 h2 H1) \quad (TY'' \land \gamma \Rightarrow 6 \leq K) \quad I 4

\quad \text{SET-TYPE}(\neg(6 \leq K)), \quad K \leq 5 \quad I 11

\quad TY'' = \{[K \ \min(5,j)][j: K \infty]\}, \text{has a contradiction} \quad I 11.1

\quad \text{Returns} \quad (T \ NIL) \quad I 11.1

(1 h2 h2 H2) \quad (TY'' \land \gamma \Rightarrow K \leq j) \quad I 4.2

\quad \text{SET-TYPE}(\neg(K \leq j)), \quad j + 1 \leq K \quad I 11

\quad TY'' = \{[K: \ \max(8,j+1)][j: K \ K-1]\}, \text{has a contradiction} \quad I 11.1

\quad \text{Returns} \quad (T \ NIL) \quad I 11.1

\quad \text{Returns} \quad (T \ NIL) \text{ for } (1 \ h2 \ h2 \ H) \quad I 4.4

\quad \text{Returns} \quad (K/n \ NIL) \text{ for } (1 \ h2 \ h2) \quad H 7.4

\quad \text{Returns} \quad ([K/m, K/n] \ NIL) \text{ for } (1 \ h2) \quad H 6.6

\quad \text{Returns} \quad ([K/k, K/m, K/n] \ NIL) \text{ for } (1) \quad 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,

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

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 deducts 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


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 the PDP-10 and on the CDC 6600 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 subsystems, 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 indentation. 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

\[\neg(a \leq b)\]

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

\[(x \leq 5 \land (x \leq 1 \rightarrow C) \land (2 \leq x \land x \leq 7 \rightarrow C) \rightarrow C)\]

is first converted to ordinary clausal form

1. \((x_0 \leq 5)\)
2. \((\neg(x_0 \leq 1) \lor C)\)
3. \((\neg(2 \leq x_0) \lor \neg(x_0 \leq 7) \lor C)\)
4. \(\neg C\)

and then converted by SET-TYPE to
1. \( \{x_0 : 0 \ 5\} \)

2. \( \{x_0 : 2 \ \infty\} \lor C \)

3. \( \{x_0 : 0 \ 1\} \lor \{x_0 : 8 \ \infty\} \lor C \)

4. \( \sim C \)

Ordinary resolution is performed on non typelist literals. Any 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

\[ \text{INTERSECT}(\{x_0 : 0 \ 5\}, \{x_0 : 2 \ \infty\} = \{x_0 : 2 \ 5\}, \]

the resolvent of 1 and 2 is

5. \( \{x_0 : 2 \ 5\} \lor C \).

Similarly we get

6. \( \{x_0 : 2 \ 5\} \)

7. \( \{x_0 : 0 \ 1\} \lor \{x_0 : 8 \ \infty\} \)

8. \( \{x_0 : 2 \ 1\} \lor \{x_0 : 8 \ \infty\} \)

9. \( \{x_0 : 8 \ 5\} \) or \( \square \)

Since \( \{x_0 : 2 \ 1\} \) and \( \{x_0 : 8 \ 5\} \) contained contradictions they were eliminated. The algorithms \( \text{SUP} \) and \( \text{INF} \) are used for this purpose, exactly as described in Section 1. Here, for \( \{x_0 : 2 \ 1\}, \)

\[ \text{SUP}(x_0, \text{NIL}) = 1 \]

\[ \text{INF}(x_0, \text{NIL}) = 2 \]
Since \([2,1]\) contains no integer we have a contradiction.

The algorithm \textsc{intersect} when applied to type lists

\[
\{ [x_1: a_1 b_1], [x_2: a_2 b_2], \ldots, [x_n: a_n b_n] \},
\]

\[
\{ [x_1: c_1 d_1], [x_2: c_2 d_2], \ldots, [x_n: c_n d_n] \},
\]

simply intersects the corresponding entries, getting

\[
\{ [x_1: e_1 f_1], [x_2: e_2 f_2], \ldots, [x_n: e_n f_n] \},
\]

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

Consider now Example 10, of Section 3.

\[
\forall k \ (k \leq 2 \rightarrow A[k] \leq A[k+1])
\]

\[
\forall m \ (3 \leq m \land m \leq 7 \rightarrow A[m] \leq A[m+1])
\]

\[
\forall n \ (6 \leq n \land n \leq j \rightarrow A[n] \leq A[n+1])
\]

\[
\rightarrow \forall K (K \leq j \rightarrow A[K] \leq A[K+1])).
\]

The ordinary clausal form is

1. \(~(k \leq 2) \lor A[k] \leq A[k+1]\)
2. \(~(3 \leq m) \lor \neg(m \leq 7) \lor A[m] \leq A[m+1]\)
3. \(~(6 \leq n) \lor \neg(n \leq j) \lor A[n] \leq A[n+1]\)
4. \(K_o \leq j_o\)
5. \(~(A[K_o] \leq A[K_o+1])\),

where \(K_o\) and \(j_o\) are skolem constants, and \(k, m\) and \(n\) are variables.
The clauses are converted to

1.  \( \{k: 3 \infty\} \lor A[k] \leq A[k+1] \)

2.  \( \{m: 0 2\} \lor \{m: 8 \infty\} \lor A[m] \leq A[m+1] \)

3.  \( \{n: 0 5\} \lor \{n: j_0 +1 \infty\} \lor \{j_0: 0 n-1\} \lor A[n] \leq A[n+1] \)

4.  \( \{K_0: 0 j_0\} \lor \{j_0: K_0 \infty\} \)

5.  \( \sim(A[K_0] \leq A[K_0 +1]) \)

Some of the resolvents of 1-5 are

6.  \( \{K_0: 3 \infty\} \)

7.  \( \{K_0: 0 2\} \lor \{K_0: 8 \infty\} \)

8.  \( \{K_0: 0 5\} \lor \{K_0: j_0 +1 \infty\} \lor \{j_0: 0 K_0 -1\} \)

9.  \( \{K_0: 3 -2\} \lor \{K_0: 8 \infty\} \)

10. \( \{K_0: 8 -5\} \lor \{K_0: j_0 +1 \infty\} \lor \{j_0: 0 K_0 -1\} \)

11. \( \{K_0: j_0 +1 j_0\} \lor \{j_0: K_0 K_0 -1\} \) or \( \sim \)

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

\[
\{A[k]: 0 \, A[k+1]\}
\]

This is controlled in the program by having a list \( (j_0, K_0, k, m, n) \) 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>
<thead>
<tr>
<th>Table I</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALGORITHM</strong></td>
</tr>
<tr>
<td><strong>IMPLY (H, C)</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IF</th>
<th>ACTION</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>$C = \text{&quot;T&quot; or } H = \text{&quot;FALSE&quot;}$</td>
<td>$\text{&quot;T&quot;}$</td>
</tr>
<tr>
<td>2.</td>
<td><strong>TYPELIST</strong></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>$H = (A \lor B)^3$</td>
<td><strong>IMPLY (NIL, $(A \rightarrow C) \land (B \rightarrow C)$)</strong></td>
</tr>
<tr>
<td>4.</td>
<td><strong>(AND-SPLIT) C = (A \land B)</strong></td>
<td>Put $\theta = \text{IMPLY (H, A)}$</td>
</tr>
<tr>
<td>4.1</td>
<td>$\theta = \text{NIL}$</td>
<td><strong>NIL</strong></td>
</tr>
<tr>
<td>4.2</td>
<td>$\theta \neq \text{NIL}$</td>
<td>Put $\lambda = \text{IMPLY (H, B, \theta)}^4$</td>
</tr>
<tr>
<td>4.3</td>
<td>$\lambda = \text{NIL}$</td>
<td><strong>NIL</strong></td>
</tr>
<tr>
<td>4.4</td>
<td>$\lambda \neq \text{NIL}$</td>
<td>$\theta \circ \lambda^5$</td>
</tr>
</tbody>
</table>
| 5.  | **(REDUCE)** | Put $H = \text{REDUCE (H)}$
| 5.1 | $C = \text{"T" or } H = \text{"FALSE"}$ | Go to 1 |
| 5.2 | $H = (A \lor B)$ | Go to 3 |
| 5.3 | $C = (A \land B)$ | Go to 4 |
| 5.4 | ELSE | Go to 6 |

*See Sections 1 and 2.

3. By the expression "$H = (A \lor B)$" we mean that $H$ has the form "$A \lor B$". Rules 4 and 3 are called "AND-SPLIT's". See [2] and [17] of [1].

4. If $\theta$ has two entries, $a/x, b/x$ with $a \neq b$, then two $\lambda$'s, $\lambda_1$ and $\lambda_2$ are computed, one for each case, and $\lambda_1 \circ \lambda_2$ is returned for $\lambda$.

5. This is just (APPEND $\theta \lambda$). If $\theta$ has an entry $a/x$ and $\lambda$ 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 $\theta \circ \lambda = (a/x b/y c/x d/z)$. 
<table>
<thead>
<tr>
<th>IF</th>
<th>ACTION</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>6. (C \equiv (A \lor B))</td>
<td>(\text{HOA}(H, C))</td>
<td></td>
</tr>
<tr>
<td>7. ((\text{PROMOTE}) \ C \equiv (A \rightarrow B))</td>
<td>(\text{IMPLY}(H \land A, B)^6)</td>
<td></td>
</tr>
<tr>
<td>7.1 Forward Chaining</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.2 PEEK forward chaining</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. (C \equiv (A \leftarrow\rightarrow B))</td>
<td>(\text{IMPLY}(H, (A \rightarrow B) \land (B \rightarrow A)))</td>
<td></td>
</tr>
<tr>
<td>9. (C \equiv (A = B))</td>
<td>Put (\emptyset: = \text{UNIFY}(A, B))</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>9.1 (\emptyset \neq \text{NIL})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.2 (\emptyset \equiv \text{NIL})</td>
<td>Go To 10</td>
<td></td>
</tr>
<tr>
<td>10. (C \equiv (\sim A))</td>
<td></td>
<td>(\text{IMPLY}(H \land A, \text{NIL}))</td>
</tr>
<tr>
<td>11. INEQUALITY*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. (\text{call \text{HOA}})</td>
<td>Put (\emptyset: = \text{HOA}(H, C))</td>
<td></td>
</tr>
<tr>
<td>12.1 (\emptyset \neq \text{NIL})</td>
<td>(\emptyset)</td>
<td></td>
</tr>
<tr>
<td>12.2 (\text{(PEEK) } \emptyset \equiv \text{NIL})</td>
<td>Put PEEK(^7) light &quot;ON&quot;</td>
<td></td>
</tr>
<tr>
<td>12.2</td>
<td>Put (\emptyset: = \text{HOA}(H, C))</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>12.3 (\emptyset \neq \text{NIL})</td>
<td></td>
<td>Go To 13</td>
</tr>
<tr>
<td>12.4 (\emptyset \equiv \text{NIL})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

\(^6\) Actually we call \(\text{IMPLY(OR-OUT}(H \land A), \text{AND-OUT}(B))\). See p. 13 of [1].

\(^7\) See p. 26 of [1]. The PEEK Light is turned off at the entry to \(\text{IMPLY}\).
IMPLY(H, C) Cont'd

<table>
<thead>
<tr>
<th>IF</th>
<th>ACTION</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>13. (Define C)</td>
<td>Put C' := DEFINE(C)</td>
<td></td>
</tr>
<tr>
<td>13.1 C' := NIL</td>
<td>Go To 14</td>
<td></td>
</tr>
<tr>
<td>13.2 C' ≠ NIL</td>
<td>IMPLY(H, C')</td>
<td></td>
</tr>
<tr>
<td>14. (See Section 2.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. ELSE</td>
<td>NIL</td>
<td></td>
</tr>
</tbody>
</table>
Table II
ALGORITHM
HOA(B,C)

<table>
<thead>
<tr>
<th></th>
<th>IF</th>
<th>ACTION</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Time limit Exceeded</td>
<td></td>
<td>NIL</td>
</tr>
<tr>
<td>2.</td>
<td>(MATCH)</td>
<td>Put $\emptyset = \text{UNIFY}(B,C)$</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>$\emptyset \neq \text{NIL}$</td>
<td></td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2.2</td>
<td>PEEK (See Section 4 of [1])</td>
<td></td>
<td>HOA(B,C)</td>
</tr>
<tr>
<td>3.</td>
<td>PAIRS (See Section 4 of [1])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>(OR-SPLIT) $C \equiv (A \lor D)$</td>
<td>Put $C' = \text{AND-OUT}(C)$</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>$C' \neq C$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>$C' = C$</td>
<td>Put $\emptyset = \text{HOA}(B \land \neg D, A)$(^8)</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>$\emptyset \neq \text{NIL}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>$\emptyset = \text{NIL}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>$C \equiv (A \rightarrow D)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td>$C \equiv (A \land D)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>$B \equiv (A \land D)$</td>
<td>Put $\emptyset = \text{HOA}(A, C)$</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>$\emptyset \neq \text{NIL}$</td>
<td></td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>6.2</td>
<td>$\emptyset = \text{NIL}$</td>
<td></td>
<td>HOA(D,C)</td>
</tr>
</tbody>
</table>

\(^8\)In Step 4.2, the "\(\neg\)" in \((\neg D)\) is pushed to the inside; e.g., \(\neg(\neg P)\) goes to \(P\), and \((\neg P \rightarrow Q)\) goes to \(P \land \neg Q\). If \(D\) contains no "\(\neg\)" or "\(\rightarrow\)" then \((\neg D)\) is omitted and the call is made \(\text{HOA}(B,A)\). Similarly in Step 4.4.
\[ \text{HOA}(B,C) \text{ Cont'd} \]

<table>
<thead>
<tr>
<th>IF</th>
<th>ACTION</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>7. (Back-chaining) (B \equiv (A \rightarrow D))</td>
<td>Put (\Theta \equiv \text{ANDS}(D,C)^*)</td>
<td>(\Theta \cdot \lambda)</td>
</tr>
<tr>
<td>7.1 (\Theta \equiv \text{NIL})</td>
<td>Go To 7E</td>
<td>(\Theta \cdot \lambda)</td>
</tr>
<tr>
<td>7.2 (\Theta \neq \text{NIL})</td>
<td>Put (\lambda \equiv \text{IMPLY}(H,A\Theta)^4)</td>
<td>(\Theta \cdot \lambda)</td>
</tr>
<tr>
<td>7.3 (\lambda \equiv \text{NIL})</td>
<td>Go To 8</td>
<td>(\Theta \cdot \lambda)</td>
</tr>
<tr>
<td>7.4 (\lambda \neq \text{NIL})</td>
<td></td>
<td>(\Theta \cdot \lambda)</td>
</tr>
<tr>
<td>7E. (B \equiv (A \rightarrow a = b))</td>
<td>Put (\Theta \equiv \text{HOA}(a = b,C))</td>
<td>(\text{NIL})</td>
</tr>
<tr>
<td>7E.1 (\Theta \equiv \text{NIL})</td>
<td></td>
<td>(\text{T})</td>
</tr>
<tr>
<td>7E.2 (\Theta \neq \text{NIL})</td>
<td>Put (\lambda \equiv \text{IMPLY}(H,A\Theta)^4)</td>
<td>(\Theta \cdot \lambda)</td>
</tr>
<tr>
<td>7E.3 (\lambda \equiv \text{NIL})</td>
<td>Go To 8</td>
<td>(\Theta \cdot \lambda)</td>
</tr>
<tr>
<td>7E.4 (\lambda \neq \text{NIL})</td>
<td></td>
<td>(\text{NIL})</td>
</tr>
</tbody>
</table>
| 8. \(B \equiv (A \leftrightarrow D)\) | \(\text{HOA}((A \rightarrow D) \land (D \rightarrow A),C)\) | |}

| 9. \(B \equiv (a = b)\) | Put \(Z \equiv \text{MINUS-ON}(a,b)\) | \(\text{NIL}\) |
| 9.1 \(Z \equiv 0\) | | \(\text{T}\) |
| 9.2 \(Z\) is a number | | \(\text{T}\) |
| 9.3 \(Z\) is not a number | Put \(a' \equiv \text{CHOOSE}(a,b),\ b' \equiv \text{OTHER}(a,b)\) (see p.16 of [1]) | \(\text{IMPLY}(H',C')\) |
| | Put \(H' \equiv H(a'/b'),\ C' \equiv C(a'/b')\) | \(\text{IMPLY}(B,C)\) |
| 10. \(B \equiv (A \lor D)\) | | \(\text{IMPLY}(H,A \lor C)^8\) |
| 11. \(B \equiv \neg A\) | | \(\text{NIL}\) |
| 12. ELSE | | \(\text{NIL}\) |

\(^*\text{ANDS is explained on p.11. of [1].}\)

\(^8\text{Actually we use AND-PURGE}(H,\neg A)\) instead of \(H\), which removes \(\neg A\) 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

\[(\forall TY' \land TY \land H \theta \to C\theta)\]

is a valid formula.

This is equivalent to the informal statement that \((TY \land H \theta \to C\theta)\) 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 hypotheses 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

\[(TY \land H \theta \to C\theta)\].

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 ≠ 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.

**GOAL**  \((TY \land H \rightarrow A \land B)\)

**Rule I-T 4.4.** If \((TY \land h \rightarrow A)\) returns \((\emptyset TY1)\) and \((TY \land H \rightarrow B\theta)\) returns \((\emptyset TY2)\) then return \((\emptyset \theta \circ \theta (TY1 \lor TY2))\) for \((TY \land H \rightarrow A \land B)\).

**Theorem.**  
\((\neg TY1 \land TY \land H\emptyset \rightarrow A\emptyset)\)

\((\neg TY2 \land TY \land H\emptyset \rightarrow (B\emptyset)\emptyset)\)

\(\rightarrow (\neg (TY1 \lor TY2) \land TY \land H \rightarrow A \land B)\)

**Proof.** By Resolution

1. \(TY1 \lor \neg TY \lor \neg H \emptyset \lor A \emptyset\)
2. \(TY2 \lor \neg TY \lor \neg H \emptyset \lor B \emptyset \emptyset\)
3. \(\neg TY1\)
4. \(\neg TY2\)
5. \(TY\)
6. \(H\)
7. \(\neg A \lor \neg B\)
8. \(A \emptyset\) 1, 3, 5, 6
9. \((B\emptyset)\emptyset\) 2, 4, 5, 6
10. \(\neg B \emptyset\) 7, 8
11. \(\square\) 9, 10
GOAL. ((TY' ∨ TY'') ∧ H → C)

Rule I-T 2.4. If (TY' ∧ H → C) returns (θ TY1) and (TY'' ∧ H → C) returns (λ TY2) then return (θ ∘ λ (TY1 ∨ TY2)) for ((TY' ∨ TY'') ∧ H → C).

Theorem. (∼TY1 ∧ TY' ∧ Hθ → Cθ)

(∼TY2 ∧ TY'' ∧ Hλ → Cλ)

→((∼TY1 ∨ TY2) ∧ (TY' ∨ TY'') ∧ H → C)

Proof. By Resolution.
1. TY1 ∨ ∼TY' ∨ ∼Hθ ∨ Cθ
2. TY2 ∨ TY'' ∨ Hλ ∨ Cλ
3. ∼TY1
4. ∼TY2
5. TY' ∨ TY''
6. H
7. ∼C
8. ∼TY' 1,3,6,7
9. ∼TY'' 2,4,6,7
10. ⊆ 5,8,9

GOAL. (TY ∧ H → a ≤ b)

RULE III. Return (NIL ∼(a ≤ b) ∧ TY)

Theorem. ∼[(∼(a ≤ b) ∧ TY)] → (TY ∧ H → a ≤ b)

Proof. ∼[(∼(a ≤ b) ∧ TY)] ↔[a ≤ b ∨ ∼TY]

↔(TY → a ≤ b)

→(TY ∧ H → a ≤ b)

GOAL. (TY ∧ B → A ∨ D)

Rule H-T 4.5. If (TY ∧ B ∧ ∼D → A) returns (θ TY1) and (TY1 ∧ B ∧ ∼A → D) returns NIL, then return (θ TY1) for (TY ∧ B → A ∨ D).
Theorem. \((\neg TY1 \land TY \land B \theta \land \neg D \theta \rightarrow A \theta)\)
\[\rightarrow (\neg TY1 \land TY \land B \theta \rightarrow A \theta \lor D \theta)\]

Proof. These are equivalent.

Rule H-T 4.6. If \((TY \land B \land \neg D \Rightarrow A)\) returns \(\theta \) and \((TY1 \land B \land \neg A \Rightarrow D)\) returns \(\lambda \) \(TY2\) then return \((\theta \circ \lambda \ TY2)\) for \((TY \land B \Rightarrow A \lor D)\)

Theorem. \((\neg TY1 \land TY \land B \theta \land \neg D \theta \rightarrow A \theta)\)
\[\neg TY2 \land TY \land B \land \neg A \lambda \rightarrow D \lambda\]
\[\rightarrow (TY2 \land TY \land B \rightarrow A \lor D)\]

Proof. By Resolution.

1. \(TY1 \lor TY \lor \neg B \theta \lor D \theta \lor A \theta\)
2. \(TY2 \lor \neg TY1 \lor \neg B \lambda \lor A \lambda \lor D \lambda\)
3. \(\neg TY2\)
4. \(TY\)
5. \(B\)
6. \(\neg A\)
7. \(\neg D\)
8. \(TY1 \ 1,4,5,7,6\)
9. \(\neg TY1 \ 2,3,5,6,7\)
10. \(\therefore \ 8,9\)

GOAL. \((TY \land H \land (A \Rightarrow D) \Rightarrow C)\)

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

Theorem. \((D \theta \rightarrow C \theta)\)
\[\land (\neg TY2 \land TY \land H \land (A \rightarrow D) \lambda \Rightarrow A \theta \lambda)\]
\[\rightarrow (\neg TY2 \land TY \land H \land (A \rightarrow D) \Rightarrow C)\]

Proof. By Resolution.

1. \(\neg D \theta \lor C \theta\)
2. \(TY2 \lor \neg TY \lor \neg H \lor A \lambda \lor A \theta \lambda\)
3. \( TY_2 \lor \sim TY \lor \sim H \lor \sim D\lambda \lor A\theta \lambda \)
4. \( \sim TY_2 \)
5. \( TY \)
6. \( H \)
7. \( \sim A \lor D \)
8. \( \sim C \)
9. \( \sim D\theta \sim 1,8 \)
10. \( A\lambda \lor A\theta \lambda \sim 2,4,5,6 \)
11. \( \sim D\lambda \lor A\theta \lambda \sim 3,4,5,6 \)
12. \( D\lambda \lor D\theta \lambda \sim 10,7 \)
13. \( D\lambda \sim 9,12 \)
14. \( A\theta \lambda \sim 13,11 \)
15. \( D\theta \lambda \sim 7,14 \)
16. \( \Box \sim 9,15 \)

GOAL. \( (TY \land A \land D \rightarrow C) \)

Rule H-T 6.5. If \((TY \land A \rightarrow C)\) returns \((0, TY_1)\) and \((TY_1 \land D \rightarrow C)\)
returns \(\text{NIL}\) then return \((0, TY_1)\) for \((TY \land A \land D \rightarrow C)\).

Theorem. \( (\sim TY_1 \land TY \land A\theta \rightarrow C\theta) \)

\[ \rightarrow (TY_1 \land TY \land A\theta \land D\theta \rightarrow C\theta) \]

Proof. Obvious

Rule H-T 6.6. If \((TY \land A \rightarrow C)\) returns \((0, TY_1)\) and \((TY_1 \land D \rightarrow C)\)
returns \((\lambda, TY_2)\) then return \((0 \circ \lambda, TY_2)\) for \((TY \land A \land D \rightarrow C)\).

Theorem. \( (\sim TY_1 \land TY \land A\theta \rightarrow C\theta) \)

\[ (\sim TY_2 \land TY_1 \land D\lambda \rightarrow C\lambda) \]

\[ \rightarrow (\sim TY_2 \land TY \land A \land D \rightarrow C) \]

Proof. By Resolution

1. \( TY_1 \lor \sim TY \lor \sim A\theta \lor C\theta \)
2. \( TY_2 \lor \sim TY_1 \lor \sim D\lambda \lor C\lambda \)
3. ¬:TY2
4. TY
5. A
6. D
7. ¬C
8. TY1 1, 4, 5, 7
9. TY1 2, 3, 6, 7
10. □ 8, 9

GOAL. (TY \land H \land (A \land b \Rightarrow C))

Rule H-T 7E.4. If (TY \land H \land a = b \Rightarrow \C) returns (\theta TY1) and
(TY \land H \land (A \land a = b \Rightarrow A\theta)) returns (\lambda TY2) then
returns (\theta \circ \lambda (TY1 \lor TY2)) for (TY \land H \land (A \land a = b \Rightarrow C))

GOAL. (TY \land H \land (A \land a \leq b) \Rightarrow C)

Rule H-T 7LE.6. If (TY \land H \land a \leq b \Rightarrow \C) returns (\theta TY1) and
(TY \land H \land (A \land a \leq b) \Rightarrow A\theta) returns (\lambda TY2) then
return (\theta \circ \lambda (TY1 \lor TY2) for (TY \land H \land (A \land a \leq b) \Rightarrow C))

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

(\negTY1 \land TY \land H\theta \land D \Rightarrow C\theta)
(\negTY2 \land TY \land H\lambda \land (A\lambda \Rightarrow D) \Rightarrow A\theta\lambda)
\rightarrow (\neg(TY1 \lor TY2) \land TY \land H \land (A \Rightarrow D) \Rightarrow C)

Proof. By Resolution.
1. TY1 \lor \neg TY \lor \neg H\theta \lor \neg D \lor C\theta
2. TY2 \lor \neg TY \lor \neg H\lambda \lor A\lambda \lor A\theta\lambda
3. TY2 \lor \neg TY \lor \neg H\lambda \lor \neg D \lor A\theta\lambda
4. \neg TY1
5. \neg TY2
6. TY
7. H
8. \neg A \lor D
9. $\sim C$

10. $\sim D \quad 1, 4, 6, 7, 9$

11. $A \land \sim A \land \sim \lambda \quad 2, 5, 6, 7$

12. $\sim D \lor A \land \lambda \quad 3, 5, 6, 7$

13. $\sim A \quad 10, 8$

14. $\square \quad 13, 11$
Appendix 3

Output from the ISTI 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]

00050  %This program does binary search on the array A[1 .. P-1] trying
00060  %to locate the element X. If successful, then LOOKUP is set
00070  %such that A[LOOKUP]=X and ERROR is set FALSE. If unsuccessful,
00080  %ERROR is set TRUE. More on this problem may be found in
00090  %Section 5 of Igarashi, London, and Luckham.

00100  ENTRY 1 < P & SORTED(A) & A[I] LE X & X < A[P];
00200  EXIT (A[LOOKUP]=X) AND (ERROR=FALSE) OR NOTFOUND(X,1,P) AND ( ER
00200  %ERROR =TRUE);
00300  BEGIN M:=1;N:=P;ERROR:=FALSE;
00500  WHILE M+1<N DO BEGIN
00600    I:=(M+N)DIV 2;
00800    ELSE BEGIN LOOKUP:=I;GOTO 1 END
00900  END;
01000  IF A[M] NE X THEN GOTO 2 ELSE BEGIN LOOKUP:=M;GO TO 1 END;
01100  2: ASSERT NOTFOUND(X,1,P);ERROR:=TRUE;
01200  1: ASSERT (A[LOOKUP]=X) AND (ERROR=FALSE ) OR NOTFOUND(X,1,P) AN
01200  %ERROR=TRUE);
01300  END.

NIL

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

Parsing . . .

VCGEN: P (PROCEED)($), UNIT
>$
MAIN#1
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 X < A[P]

IMP 1<P
   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 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]

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

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]

IMP M < (N+M) DIV 2
INSUB LEPREV IMPREV 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]

IMP M < (N+M) DIV 2

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

TRYING TO PROVE MAIN#S2
CHOICE: P(ROCEED)(S),+-N,VCGEN,ASSUME,
END, DEFER, SWITCH, STATUS, RED (UCE)

>DEFER

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

>2

VERIFICATION CONDITION MAIN#5

Simplification

>>> ENTERING RPV WITH

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

HCMATCH INSUB LEPRV IMPRV 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 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 NOT (X = A[M])
    IMP NOTFOUND(X, 1, P)
LEPRV IMPVR

<<<< LEAVING RPROVER WITH
    SORTED(A)
    AND X < A[M+1]
    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#2 (DEFERRED) TO BE PROVED
    MAIN#3 HAS BEEN SIMPLIFIED TO
    MAIN#3 (DEFERRED) TO BE PROVED
    MAIN#4  ***PROVED***
    MAIN#5 HAS BEEN SIMPLIFIED TO
    MAIN#5 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
VERIFICATION CONDITION VCMS
(THEOREM TO BE PROVED)
NIL

SORTED(M, MIN(N+1, 2), N)
AND 2 LE N
AND A(M, 2, MIN(N, 1))
AND IPILARGEST(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 IPILARGEST(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 IPILARGEST(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 L248)
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>SPROCEEDING
.(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>SPROCEEDING
...(P-> ORH 1 P-> U H 1)

SORTED(M, 2, N)
PROVED
W>SPROCEEDING
.(P-> ORH 1 P-> U H 2)
MORE TIME ? (TYPE NUMBER OR NO)
>N0

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)
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)
W>USE
LEMMA:
>IP1LARGEST(1, M) IMP (M[1] LE M[2]);
==> (1)

IP1LARGEST(1, M)
<= (1)

IP1LARGEST(1, M)
(LEMMA USED SAVED IN L241)

IP1LARGEST(1, M)
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)

PROVED
W>$PROCEEDING
(P-> ORH 1 P-> U H)

SORTED(M, 2, N)
PROVED
\[ P \rightarrow \text{ORH } 1 \]
\[
N \in [2..\infty] \\
\text{AND SORTED}(M, \text{MIN}(N+1, 2), N) \\
\text{AND } A(M, 2, \text{MIN}(N, 1)) \\
\text{AND } \text{IP1LARGEST} \text{MIN}(N, 1), M)
\]
IMP SORTED(M, 1, N)

PROVED
\[ P \rightarrow \text{ORH } 2 \]

(BACKUP POINT)

\[ P \rightarrow \text{ORH } 2 P \rightarrow \]

\[ \text{W} \rightarrow \text{TP} \]
\[
N \in [2..\infty] \\
\text{AND SORTED}(M, \text{MIN}(N+1, 2), N) \\
\text{AND } A(M, 2, \text{MIN}(N, 1)) \\
\text{AND } 0 = \text{MIN}(-N + 1, 0)
\]
IMP SORTED(M, 1, N)

\[ \text{W} \rightarrow \text{A} \]

ASSUMED

\[ P \rightarrow \text{ORH } 2 \]
\[
N \in [2..\infty] \\
\text{AND SORTED}(M, \text{MIN}(N+1, 2), N) \\
\text{AND } A(M, 2, \text{MIN}(N, 1)) \\
\text{AND } 0 = \text{MIN}(-N + 1, 0)
\]
IMP SORTED(M, 1, N)

PROVED
\[ P \rightarrow \text{S PROCEEDING} \]

(P ORH)
\[
N \in [2..\infty] \\
\text{AND SORTED}(M, \text{MIN}(N+1, 2), N) \\
\text{AND } A(M, 2, \text{MIN}(N, 1)) \\
\text{AND } \text{IP1LARGEST} \text{MIN}(N, 1), M)
\]
IMP SORTED(M, 1, N)

\[ \text{AND} \]
\[
N \in [2..\infty] \\
\text{AND SORTED}(M, \text{MIN}(N+1, 2), N) \\
\text{AND } A(M, 2, \text{MIN}(N, 1)) \\
\text{AND } 0 = \text{MIN}(-N + 1, 0)
\]
IMP SORTED(M, 1, N)

PROVED
\[ P \rightarrow \text{S PROCEEDING} \]

(P ORH)

\[ \text{SORTED}(M, 1, N) \]

PROVED
\[ P \rightarrow \text{S PROCEEDING} \]

NIL
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


