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

\( \beta \)
(1) \( (k \leq 2 \rightarrow A[k] \leq A[k+1]) \)
\( \wedge (3 \leq m \leq 7 \rightarrow A[m] \leq A[m+1]) \)
\( \wedge (6 \leq n \leq j \rightarrow A[n] \leq A[n+1]) \)
\( \rightarrow K \leq j \rightarrow A[K] \leq A[K+1] \)

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

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

(1 h1 H) \( ([K: 0 j][j: K \infty] \wedge \alpha \wedge \beta \wedge \gamma \Rightarrow K \leq 2) \)
SET-TYPE(\( \sim (K \leq 2) \)), 3 \( \leq K \)
TY' = \( ([K: 3 j][j: K \infty]) \), has no contradiction
Returns (T TY')  
I 11.3

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

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

H 6.4

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

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

H 7, 7.2

(1 h2 h1 H1) \( (TY' \wedge (\beta \wedge \gamma) \Rightarrow 3 \leq K) \)
SET-TYPE(\( \sim (3 \leq K) \)),  \( K \leq 2 \)
TY' = \( ([K: 3 \min(2, j)]) \), has a contradiction
Returns (T NIL)  
I 11.1
\[(1\ h2\ h1\ H2)\quad (TY' \land (\beta \land \gamma) \Rightarrow K \leq 7)\]

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

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

Returns \((T\ TY')\)

Returns \((T\ TY')\) for \((1\ h2\ h1\ H)\)

Returns \((K/m\ TY')\) for \((1\ h2\ h1)\)

---

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

---

\[(1\ h2\ h2\ H)\quad (TY'' \land \gamma \Rightarrow 6 \leq K \land K \leq j)\]

---

\[(1\ h2\ h2\ H1)\quad (TY'' \land \gamma \Rightarrow 6 \leq K)\]

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

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

Returns \((T\ NIL)\)

---

\[(1\ h2\ h2\ H2)\quad (TY'' \land \gamma \Rightarrow K \leq j)\]

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

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

Returns \((T\ NIL)\)

Returns \((T\ NIL)\) for \((1\ h2\ h2\ H)\)

Returns \((K/n\ NIL)\) for \((1\ h2\ h2)\)

Returns \([(K/m, K/n)\ NIL)\) for \((1\ h2)\)

Returns \([(K/k, K/m, K/n)\ NIL)\) for \((1)\)

---

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).
\[(\text{UNIFY}(\Rightarrow (\text{Array } B (+ k 1))(\text{Amax } B j (k + 1))))\]
\[\Rightarrow (\text{Array } A_0 i_0)(\text{Amax } A_0 1 i_0))\]
\[(\text{UNIFY } (\text{Array } B (+ k 1)))\]
\[\text{Array } A_0 i_0 )\]

\[(\text{UNIFY } B A_0) , A_0 /B\]
\[(\text{UNIFY } (+ k 1) i_0) \quad \text{It deduces that}\]
\[(+ k (+ (-i_0) 1)) = 0, \text{ and returns the substitution}\]
\[(+ i_0 -1)/k\]
\[\text{UNIFY Amax}(A_0, j, i_0)\]
\[\text{Amax}(A_0, 1, i_0) \quad 1/j\]

Returns \{A_0 /B, (i_0 -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
\[i_0 \text{ precedes } j_0\]
in the canonical ordering, even though \(i_0 \) precedes \(j\). 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, INFPRT and XEVAL, are used to augment the prover. INFPRT 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. \((x_o \leq 5)\)
2. \((\neg(x_o \leq 1) \lor C)\)
3. \((\neg(2 \leq x_o) \lor \neg(x_o \leq 7) \lor C)\)
4. \(\neg C\),

and then converted by SET-TYPE to
1. \( \{x_o : 0 \ 5\} \)
2. \( \{x_o : 2 \ \infty\} \lor C \)
3. \( \{x_o : 0 \ 1\} \lor \{x_o : 8 \ \infty\} \lor C \)
4. \( \neg C \)

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

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

the resolvent of 1 and 2 is

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

Similarly we get

6. \( \{x_o : 2 \ 5\} \)
7. \( \{x_o : 0 \ 1\} \lor \{x_o : 8 \ \infty\} \)
8. \( \{x_o \ 2 \ 1\} \lor \{x_o : 8 \ \infty\} \)
9. \( \{x_o \ 8 \ 5\} \lor \square \)

Since \( \{x_o : 2 \ 1\} \) and \( \{x_o : 8 \ 5\} \) 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 \( \{x_o : 2 \ 1\}, \)

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

The algorithm INTERSECT when applied to type lists

\[
\{(x_1:a_1~b_1)\} \cdot \{x_2:a_2~b_2\} \cdots \{x_n:a_n~b_n\},
\]

\[
\{(x_1:c_1~d_1)\} \cdot \{x_2:c_2~d_2\} \cdots \{x_n:c_n~d_n\},
\]

simply intersects the corresponding entries, getting

\[
\{(x_1:e_1~f_1)\} \cdot \{x_2:e_2~f_2\} \cdots \{x_n:e_n~f_n\},
\]

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

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

1. \([k: 3 \infty] \lor A[k] \leq A[k+1]\)
2. \([m: 0 \infty \lor m: 8 \infty] \lor A[m] \leq A[m+1]\)
3. \([n: 0 \infty \lor [n: j_0 + 1 \infty][j_0: 0 \infty]\] \lor A[n] \leq A[n+1]\)
4. \([(K_o: 0 \infty)[j_0: K_o \infty]]\)
5. \(\neg(A[K_o] \leq A[K_o + 1])\)

Some of the resolvents of 1-5 are

6. \([(K_o: 3 \infty)]\) 1, 5
7. \([(K_o: 0 \infty \lor K_o: 8 \infty)]\) 2, 5
8. \([(K_o: 0 \infty \lor [K_o: j_0 + 1 \infty][j_0: 0 \infty -1]]\) 3, 5
9. \([(K_o: 0 \infty \lor K_o: 8 \infty)]\) 6, 7
10. \([(K_o: 0 \infty \lor [K_o: j_0 + 1 \infty][j_0: 0 \infty -1]]\) 8, 9
11. \([(K_o: j_0 + 1 \infty)[j_0: K_o \infty -1]) or L\) 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

\([A[k]: 0 \ A[k+1]]\).

This is controlled in the program by having a list \((j_0 \ K_o \ k \ m \ n)\) 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>IF</th>
<th>ACTION</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( C \equiv &quot;T&quot; ) or ( H \equiv &quot;FALSE&quot; )</td>
<td></td>
<td>&quot;T&quot;</td>
</tr>
<tr>
<td>2. TYPELIST*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. ( H \equiv (A \lor B)^3 )</td>
<td>IMPLY(NIL, ( (A \rightarrow C) \land (B \rightarrow C) ))</td>
<td></td>
</tr>
<tr>
<td>4. (AND-SPLIT) ( C \equiv (A \land B) )</td>
<td>Put ( \Theta = \text{IMPLY}(H, A) )</td>
<td></td>
</tr>
<tr>
<td>4.1 ( \Theta = \text{NIL} )</td>
<td>NIL</td>
<td></td>
</tr>
<tr>
<td>4.2 ( \Theta \neq \text{NIL} )</td>
<td>Put ( \lambda = \text{IMPLY}(H, B \Theta)^4 )</td>
<td>NIL</td>
</tr>
<tr>
<td>4.3 ( \lambda = \text{NIL} )</td>
<td></td>
<td>( \Theta \circ \lambda^5 )</td>
</tr>
<tr>
<td>4.4 ( \lambda \neq \text{NIL} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. (REDUCE)</td>
<td>Put ( H = \text{REDUCE}(H) )</td>
<td></td>
</tr>
<tr>
<td>5.1 ( C \equiv &quot;T&quot; ) or ( H \equiv &quot;FALSE&quot; )</td>
<td>Put ( C = \text{REDUCE}(C) )</td>
<td>Go to 1</td>
</tr>
<tr>
<td>5.2 ( H \equiv (A \lor B) )</td>
<td>Go to 3</td>
<td></td>
</tr>
<tr>
<td>5.3 ( C \equiv (A \land B) )</td>
<td>Go to 4</td>
<td></td>
</tr>
<tr>
<td>5.4 ELSE</td>
<td>Go to 6</td>
<td></td>
</tr>
</tbody>
</table>

*See Sections 1 and 2.

3 By the expression "\( H \equiv (A \lor B) \)" 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 \( \text{APPEND } \Theta \lambda \). 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 \( \Theta \circ \lambda = (a/x \ b/y \ c/x \ d/z) \).
\textbf{IMPLY}(H,C) \textit{Cont'd}

\begin{tabular}{|l|l|l|}
  \hline
  \textbf{IF} & \textbf{ACTION} & \textbf{RETURN} \\
  \hline
  6. & \(C \equiv (A \lor B)\) & \text{HOA}(H,C) \\
  7. & (PROMOTE) \(C \equiv (A \rightarrow B)\) & \text{IMPLY}(H \land A, B)^6 \\
  7.1 & Forward Chaining & \\
  7.2 & PEEK forward chaining & \\
  8. & \(C \equiv (A \leftarrow \rightarrow B)\) & \text{IMPLY}(H, (A \rightarrow B) \land (B \rightarrow A)) \\
  9. & \(C \equiv (A = B)\) & \text{Put } \emptyset_{i} = \text{UNIFY}(A,B) \\
  9.1 & \emptyset_{i} \neq \text{NIL} & \emptyset \\
  9.2 & \emptyset_{i} = \text{NIL} & \text{Go To 10} \\
  10. & \(C \equiv (\neg A)\) & \text{IMPLY}(H \land A, \text{NIL}) \\
  11. & \text{INEQUALITY}\^* & \\
  12. & (call HOA) & \text{Put } \emptyset_{i} = \text{HOA}(H,C) \\
  12.1 & \emptyset_{i} \neq \text{NIL} & \emptyset \\
  12.2 & (PEEK) \emptyset_{i} = \text{NIL} & \text{Put PEEK}^7 \text{ light "ON"} \\
  & & \text{Put } \emptyset_{i} = \text{HOA}(H,C) \\
  12.3 & \emptyset_{i} \neq \text{NIL} & \emptyset \\
  12.4 & \emptyset_{i} = \text{NIL} & \text{Go To 13} \\
  \hline
\end{tabular}

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

\footnote{\textit{See p. 26 of [1]. The PEEK Light is turned off at the entry to IMPLY.}}
<table>
<thead>
<tr>
<th>IF</th>
<th>ACTION</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>13. (Define C)</td>
<td>Put ( C' ) = ( \text{DEFINE}(C) )</td>
<td></td>
</tr>
<tr>
<td>13.1 ( C' = \text{NIL} )</td>
<td>Go To 14</td>
<td></td>
</tr>
<tr>
<td>13.2 ( C' \neq \text{NIL} )</td>
<td></td>
<td>( \text{IMPLY}(H, C') )</td>
</tr>
<tr>
<td>14. (See Section 2.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. ELSE</td>
<td></td>
<td>( \text{NIL} )</td>
</tr>
<tr>
<td>IF</td>
<td>ACTION</td>
<td>RETURN</td>
</tr>
<tr>
<td>----</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>1.</td>
<td>Time limit Exceeded</td>
<td>NIL</td>
</tr>
<tr>
<td>2.</td>
<td>(MATCH)</td>
<td>Put $\theta' = \text{UNIFY}(B,C)$</td>
</tr>
<tr>
<td>2.1</td>
<td>$\theta \neq \text{NIL}$</td>
<td>$\theta$</td>
</tr>
<tr>
<td>2.2</td>
<td>PEEK (See Section 4 of [1])</td>
<td>HOA($B,C$)</td>
</tr>
<tr>
<td>3.</td>
<td>PAIRS (See Section 4 of [1])</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>
</tr>
<tr>
<td>4.1</td>
<td>$C' \neq C$</td>
<td>IMPLY($H,C'$)</td>
</tr>
<tr>
<td>4.2</td>
<td>$C' \equiv C$</td>
<td>$\theta$</td>
</tr>
<tr>
<td>4.3</td>
<td>$\theta \neq \text{NIL}$</td>
<td>HOA($B \land \neg D,A$) (^8)</td>
</tr>
<tr>
<td>4.4</td>
<td>$\theta \equiv \text{NIL}$</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>$C \equiv (A \rightarrow D)$</td>
<td>IMPLY($B,C$)</td>
</tr>
<tr>
<td>5.2</td>
<td>$C \equiv (A \land D)$</td>
<td>IMPLY($B,C$)</td>
</tr>
<tr>
<td>6.</td>
<td>$B \equiv (A \land D)$</td>
<td>Put $\theta = \text{HOA}(A,C)$</td>
</tr>
<tr>
<td>6.1</td>
<td>$\theta \neq \text{NIL}$</td>
<td>$\theta$</td>
</tr>
<tr>
<td>6.2</td>
<td>$\theta \equiv \text{NIL}$</td>
<td>HOA($D,C$)</td>
</tr>
</tbody>
</table>

\(^8\)In Step 4.2, the "\(\neg\)" in \(\neg \neg D\) 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 HOA($B,A$). Similarly in Step 4.4.
HOA(B, C) Cont'd

IF

ACTION

RETURN

7. (Back-chaining)
   \( B \equiv (A \rightarrow D) \)
   Put \( \theta := \text{ANDS}(D, C) \)^*

7.1 \( \theta \equiv \text{NIL} \)
   Go To 7E

7.2 \( \theta \neq \text{NIL} \)
   Put \( \lambda := \text{IMPLY}(H, A\theta) \)^4

7.3 \( \lambda \equiv \text{NIL} \)
   Go To 8

7.4 \( \lambda \neq \text{NIL} \)
   \( \theta \ast \lambda \)

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

7E.1 \( \theta \equiv \text{NIL} \)
   NIL

7E.2 \( \theta \neq \text{NIL} \)
   Put \( \lambda := \text{IMPLY}(H, A\theta) \)^4

7E.3 \( \lambda \equiv \text{NIL} \)
   Go To 8

7E.4 \( \lambda \neq \text{NIL} \)
   \( \theta \ast \lambda \)

8. \( B \equiv (A \leftarrow D) \)
   \( \text{HOA}((A \rightarrow D) \land (D \rightarrow A), C) \)

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. \( B \equiv (A \lor D) \)

11. \( B \equiv \lnot A \)

12. ELSE
   NIL

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

^8Actually we use AND-PURGE(H, \lnot A) instead of H, which removes \lnot 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

\[(\forall TY' \land TY \land Hθ \rightarrow Cθ)\]

(*)

is a valid formula.

This is equivalent to the informal statement that \((TY \land Hθ \rightarrow Cθ)\) 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

\[(TY \land Hθ \rightarrow Cθ).\]

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 \( \theta \) is ever substituted (i.e., a case where \( a/x \) and \( b/x \) are both in \( \theta \), 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.

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

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

**Theorem.** \((\forall TY1 \land TY \land H \Theta \rightarrow A \Theta)\)

\((\forall TY2 \land TY \land H \Theta \rightarrow (B \Theta ) \Theta 2)\)

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

**Proof.** By Resolution

1. \(TY1 \lor \neg TY \lor \neg H \Theta \lor \neg A \Theta\)
2. \(TY2 \lor \neg TY \lor \neg H \Theta \lor \neg B \Theta \Theta 2\)
3. \(\neg TY1\)
4. \(\neg TY2\)
5. \(TY\)
6. \(H\)
7. \(\neg A \lor \neg B\)
8. \(A \Theta\)
9. \((B \Theta ) \Theta 2\)
10. \(\neg B \Theta\)
11. \(\square\)

1, 3, 5, 6
2, 4, 5, 6
7, 8
9, 10
GOAL. \(((TY' \lor TY'') \land H \rightarrow C)\)

Rule I-T 2.4. If \((TY' \land H \rightarrow C)\) returns \((\emptyset \Rightarrow TY1)\) and \((TY'' \land H \Rightarrow C)\) returns \((\lambda \Rightarrow TY2)\) then return \((\emptyset \circ \lambda \Rightarrow (TY1 \lor TY2))\) for \(((TY' \lor TY'') \land H \Rightarrow C)\).

Theorem. \((\neg TY1 \land TY' \land H \theta \rightarrow C \theta)\)

\((\neg TY2 \land TY'' \land H \lambda \rightarrow C \lambda)\)

\(\rightarrow (\neg (TY1 \lor TY2) \land (TY' \lor TY'') \land H \rightarrow C)\)

Proof. By Resolution.

1. \(TY1 \lor \neg TY' \lor \neg H \theta \lor C \theta\)
2. \(TY2 \lor TY'' \lor H \lambda \lor C \lambda\)
3. \(\neg TY1\)
4. \(\neg TY2\)
5. \(TY' \lor TY''\)
6. \(H\)
7. \(\neg C\)
8. \(\neg TY'\)
9. \(\neg TY''\)
10. \(\neg\neg\)

GOAL. \((TY \land H \rightarrow a \leq b)\)

Rule III. Return \((\emptyset \Rightarrow (\neg (a \leq b) \land TY))\)

Theorem. \(\neg [(\neg (a \leq b) \land TY) \rightarrow (TY \land H \rightarrow a \leq b)]\)

Proof. \(\neg [(\neg (a \leq b) \land TY) \rightarrow [a \leq b \lor \neg TY]]\)

\(\quad \rightarrow (TY \rightarrow a \leq b)\)

\(\quad \rightarrow (TY \land H \rightarrow a \leq b)\)

GOAL. \((TY \land B \rightarrow A \lor D)\)

Rule H-T 4.5. If \((TY \land B \land \neg D \Rightarrow A)\) returns \((\emptyset \Rightarrow TY1)\) and \((TY1 \land B \land \neg A \Rightarrow D)\) returns \(\emptyset\), then return \((\emptyset \Rightarrow TY1)\) for \((TY \land B \Rightarrow A \lor D)\).
Theorem. \((\neg TYL \land TY \land B \theta \land \neg D \theta \rightarrow A \theta)\)

\[\rightarrow (\neg TYL \land TY \land B \theta \rightarrow A \theta \lor D \theta)\]

Proof. These are equivalent.

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

Theorem. \((\neg TYL \land TY \land B \theta \land \neg D \theta \rightarrow A \theta)\)

\[(\neg TY2 \land TYL \land B \lambda \land \neg A \lambda \Rightarrow D \lambda)\]

\[\rightarrow (TY2 \land TY \land B \rightarrow A \lor D)\]

Proof. By Resolution.

1. TYL \lor TY \lor \neg B \theta \lor D \theta \lor A \theta
2. TY2 \lor \neg TYL \lor \neg B \lambda \lor A \lambda \lor D \lambda
3. \neg TY2
4. TY
5. B
6. \neg A
7. \neg D
8. TYL 1,4,5,7,6
9. \neg TYL 2,3,5,6,7
10. \square 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 \((\lambda \land TY2)\) then return \((\theta \circ \lambda \land TY2)\) 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 \land A \lambda \lor A \theta \lambda
3. $TY_2 \lor \neg TY \lor \neg H \lor \neg D \lambda \lor A \theta \lambda$

4. $\neg TY_2$

5. $TY$

6. $H$

7. $\neg A \lor D$

8. $\neg C$

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

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

11. $\neg D \lambda \lor A \theta \lambda \ 3,4,5,6$

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

13. $D \lambda \ 9,12$

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

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

16. $\square \ 9,15$

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

Rule H-T 6.5. If $(TY \land A \Rightarrow C)$ returns $(\theta \ TY_1)$ and $(TY_1 \land D \Rightarrow C)$

returns NIL then return $(\theta \ TY_1)$ for $(TY \land A \land D \Rightarrow C)$.

Theorem. $(\neg TY_1 \land TY \land A \theta \Rightarrow C_0)$

$(TY_1 \land TY \land A \theta \land D \theta \Rightarrow C_0)$

Proof. Obvious

Rule H-T 6.6. If $(TY \land A \Rightarrow C)$ returns $(\theta \ TY_1)$ and $(TY_1 \land D \Rightarrow C)$

returns $(\lambda \ TY_2)$ then return $(\theta \circ \lambda \ TY_2)$ for $(TY \land A \land D \Rightarrow C)$.

Theorem. $(\neg TY_1 \land TY \land A \theta \Rightarrow C_0)$

$(\neg TY_2 \land TY_1 \land D \lambda \Rightarrow C \lambda)$

$(\neg TY_2 \land TY \land A \land D \Rightarrow C)$

Proof. By Resolution

1. $TY_1 \lor \neg TY \lor \neg A \theta \lor C_0$

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

GOAL. \((TY \land H \land (A + A = b) \rightarrow C)\)

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

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

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

Theorem. (For both). \((D \lor a = b \lor a \lor b)\).
\((\neg TY1 \land TY \land H\emptyset \land D \rightarrow C\emptyset)\)
\((\neg TY2 \land TY \land H\lambda \land (A\lambda \rightarrow D) \rightarrow A\emptyset\lambda)\)
\((\neg (TY1 \lor TY2) \land TY \land H \land (A \rightarrow D) \rightarrow C)\)

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

10. \( \sim D \) 1,4,6,7,9

11. \( A \lambda \lor A \theta \lambda \) 2,5,6,7

12. \( \sim D \lor A \theta \lambda \) 3,5,6,7

13. \( \sim A \) 10,8

14. \( \square \) 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 0354:04v
@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.
00095
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
00300 &&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 GO TO 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
01300 &&D (ERROR=TRUE);
01300 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)
>SPROCEEDING

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)
>SPROCEEDING

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 LEPKV IMPKV LOGSUB SAVESTATE MPHYP EXPO 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?
>MAIN#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 (ROCEED) ($), +/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 LEP RV IMPR V LOGSUB SAVESTATE MHP Y EXP Q
NEW EQUALITY M + 1 = N
FROM: M < N
AND: N LE M + 1
EXP Q 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 IMPPRV
<<< 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:   PROCEED($), +/-N, VCGEN, ASSUME, END, DEFER, SWITCH, STATUS, RED(UCE)
>STATUS
    MAIN#1 ***PROVED***
    MAIN#2 HAS BEEN SIMPLIFIED TO
        MAIN#S2 (DEFERRED) TO BE PROVED
    MAIN#3 HAS BEEN SIMPLIFIED TO
        MAIN#S3 (DEFERRED) 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:   PROCEED($), +/-N, VCGEN, ASSUME, END, DEFER, SWITCH, STATUS, RED(UCE)
>END

PROVE: NO($), UN(DEFERRED), OR DEF(ERRRED) (VC'S)
>S
DUMP: DMP($), PRE, VC, VCS, NO, CLEAR (STRUCTURE)
>NO
NIL
>PROVEIT VCMS;

VERIFICATION CONDITION VCMS
(THEOREM TO BE PROVED)
NIL

\[
\text{SORTED}(M, \text{MIN}(N+1, 2), N)
\]
AND \(2 \leq N\)
AND \(A(M, 2, \text{MIN}(N, 1))\)
AND \(\text{IPILARGEST} \left(\text{MIN}(N, 1), M\right)\)
OR \(\emptyset = \text{MIN}(-N + 1, 0)\)

IMP SORTED(M, 1, N)
(backup point)
\(\wedge\) $>$ proceeds
(backup point)
(P->)
\(\wedge\) $>$ TP

\[N \in [2..\text{INFINITY}]\]
AND \(\text{SORTED}(M, \text{MIN}(N+1, 2), N)\)
AND \(A(M, 2, \text{MIN}(N, 1))\)
AND \(\text{IPILARGEST} \left(\text{MIN}(N, 1), M\right)\)
OR \(\emptyset = \text{MIN}(-N + 1, 0)\)

IMP SORTED(M, 1, N)
\(\wedge\) $>$ proceeds
....... (P-> ORH)
(P-> ORH 1)
(backup point)
(P-> ORH 1 P->)
\(\wedge\) $>$ TP

\[N \in [2..\text{INFINITY}]\]
AND \(\text{SORTED}(M, \text{MIN}(N+1, 2), N)\)
AND \(A(M, 2, \text{MIN}(N, 1))\)
AND \(\text{IPILARGEST} \left(\text{MIN}(N, 1), M\right)\)
IMP SORTED(M, 1, N)
\(\wedge\) $>$ proceeds
....... RAN OUT OF TRICKS
\(\wedge\) USE

\text{LEMMA:}

\[\text{SORTED}(M, I+1, N) \text{ AND } (M[I] \leq M[I+1]) \text{ IMP SORTED}(M, I, N);\]
\[==> (1)\]

\[\text{SORTED}(M, I+1, N)\]
\[\text{AND } M[I] \leq M[I+1]\]
IMP SORTED(M, I, N)
\[<= (1)\]

\[\text{SORTED}(M, I+1, N)\]
\[\text{AND } M[I] \leq M[I+1]\]
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>S 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>S PROCEEDING
...(P-> ORH 1 P-> U H 1)

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

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

SELECT(M, 2, N)
AND N IN [2..INFINITY]
AND SELECT(M, 2, N)
AND A(M, 2, 1)
AND R1LARGEST(1, M)
OK??
>OK
W>$PROCEEDING
..........RAN OUT OF TRICKS
W>TP

SELECT(M, 2, N)
AND N IN [2..INFINITY]
AND SELECT(M, 2, N)
AND A(M, 2, 1)
AND R1LARGEST(1, M)
W>USE
LEMMA:
>R1LARGEST(1,M) IMP (M[1] LE M[2]);
===> (1)

R1LARGEST(1, M)
<== (1)

R1LARGEST(1, M)
(LEMMA USED SAVED IN L241)

R1LARGEST(1, M)
OK??
>YES
(USE)=================================
(P-> ORH 1 P-> U H 2 U)
W>$PROCEEDING
.(P-> ORH 1 P-> U H 2 U)
......(P-> ORH 1 P-> U H 2 U)
R1LARGEST(1, M)
PROVED
W>$PROCEEDING
(P-> ORH 1 P-> U H 2)

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

SELECT(M, 2, N)
PROVED
\( \omega \rightarrow \$\text{PROCEEDING} \)
\((P \rightarrow \text{ORH 1}) \)

\[ \begin{align*}
\text{N IN [2..INFINITY]} \\
\text{AND SORTED(M, MIN(N+1, 2), N)} \\
\text{AND A(M, 2, MIN(N, 1))} \\
\text{AND IP1LARGEST(MIN(N, 1), M)} \\
\text{IMP SORTED(M, 1, N)}
\end{align*} \]

PROVED
\( \omega \rightarrow \$\text{PROCEEDING} \)
\((P \rightarrow \text{ORH 2}) \)
\((\text{BACKUP POINT}) \)
\((P \rightarrow \text{ORH 2 P \rightarrow}) \)
\( \omega \rightarrow \text{TP} \)

\[ \begin{align*}
\text{N IN [2..INFINITY]} \\
\text{AND SORTED(M, MIN(N+1, 2), N)} \\
\text{AND A(M, 2, MIN(N, 1))} \\
\text{AND } \emptyset = \text{MIN}(-N + 1, 0) \\
\text{IMP SORTED(M, 1, N)}
\end{align*} \]

\( \omega \rightarrow A \)

ASSUMED

\((P \rightarrow \text{ORH 2}) \)

\[ \begin{align*}
\text{N IN [2..INFINITY]} \\
\text{AND SORTED(M, MIN(N+1, 2), N)} \\
\text{AND A(M, 2, MIN(N, 1))} \\
\text{AND } \emptyset = \text{MIN}(-N + 1, 0) \\
\text{IMP SORTED(M, 1, N)}
\end{align*} \]

PROVED
\( \omega \rightarrow \$\text{PROCEEDING} \)
\((P \rightarrow \text{ORH}) \)

\[ \begin{align*}
\text{N IN [2..INFINITY]} \\
\text{AND SORTED(M, MIN(N+1, 2), N)} \\
\text{AND A(M, 2, MIN(N, 1))} \\
\text{AND IP1LARGEST(MIN(N, 1), M)} \\
\text{IMP SORTED(M, 1, N)} \\
\text{AND} \\
\text{N IN [2..INFINITY]} \\
\text{AND SORTED(M, MIN(N+1, 2), N)} \\
\text{AND A(M, 2, MIN(N, 1))} \\
\text{AND } \emptyset = \text{MIN}(-N + 1, 0) \\
\text{IMP SORTED(M, 1, N)}
\end{align*} \]

PROVED
\( \omega \rightarrow \$\text{PROCEEDING} \)
\((P \rightarrow \text{P \rightarrow}) \)

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

PROVED
\( \omega \rightarrow \$\text{PROCEEDING} \)

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


