A Resolution-based Prover for General Inequalities

by

W. W. Bledsoe

July 1979 ATP-52

TABLE OF CONTENTS

A Resolution-based Prover for General Inequalities

W. W. Bledsoe

Abstract. A variation of Resolution is described which has been designed to prove theorems about general inequalities. Special clauses and resolvents are introduced to avoid the explicit use of certain axioms, such as the transitivity and interpolation axioms for inequalities, which tend to explode the search space. Several examples are given along with results from a computer implementation.

1. Introduction

The purpose of this paper is to describe a resolution-based theorem prover which has been designed to prove theorem about real inequalities.

An important motive for building special inequality provers is to avoid the explicit use of axioms such as

TRANSITIVITY: $\forall x \forall y \forall z \quad (z \leq y \wedge y \leq z \rightarrow x \leq z)$

INTERPOLATION: $\forall x \forall y \quad (x < y \rightarrow \exists z \ (x < z < y))$,

$$\forall x > 0 \ \forall y > 0 \ \exists z > 0 \ (z < x \wedge z < y).$$

Such axioms tend to lengthen the proof search because they can match with other formulas in so many unproductive ways. Also, the explicit use of the field axioms for the real number present similar problems.

To avoid these difficulties special "built-in" procedures have been suggested and used with varying degrees of success. Some of these procedures are

(1) the built-in partial ordering of Slagle and Norton [2];

(2) the ground inequality packages of King [3], Oppen, et al. [4], Shostak [5], Bledsoe, et al. [6] (these tend to be in the Presburger mode);

(4) the methods of Hodes [7];

(5) the Restriction Intervals Method [8, 6].

In [6] we combined a ground inequality package and a restriction interval method with our existing natural deduction prover to obtain a general inequality prover capable of handling a variety of inequality theorems without the explicit use of any axiom about inequalities or the real numbers.

Even though this prover has met with a degree of success on some rather difficult theorems, still further changes are necessary to handle many other inequality theorems. One such change is the inequality chaining described below.

The reason for this paper is to show one way in which these features can be built into a resolution prover and to compare the results with those from a natural deduction prover with similar features.

Resolution is particularly suited for the restricted variable method (see variable-elimination resolvents, Sect. 2.2, below) because each clause has its own unique variable. Some other advantages of Resolution are that no substitution needs to be returned from the proof of a subgoal, no backtracking is needed, the clausal data type is uniform and simple, and completeness results are easier to obtain. It remains to be seen whether these advantages offset disadvantages that have been articulated elsewhere, but it seems a safe bet that a well-tailored resolution system will be best for inequality theorems of limited difficulty where human interaction is not required, and it is hoped that such a limited capacity prover can be coded on a mini-computer to work in parallel with and support a larger system.

2. Resolution$\leq$

Resolution$\leq$ is much like ordinary resolution [1], except that in addition to the traditional clauses there is a special clause (only one) called TY which is essentially a conjunction of ground inequality literals, and four different types of resolvents are used. These are

- Ordinary Resolvents
- TY-Resolvents
- Variable-elimination Resolvents
- Chain Resolvents

2.1 TY-Resolvents

A TY-Resolvent is obtained by conjoining a ground inequality literal with the special clause TY and checking the result for consistency by calling the routine, CONTRADICTION. If CONTRADICTION succeeds, then the resolvent is $\square$; otherwise, it is the augmented TY.

2.2 Variable-elimination Resolvents

A literal, $x \leq a$, is called an RL-literal if x is a variable which does not occur in a, and the variable x is called an RL-variable for that literal. This definition is extended to include the cases $x < a$, $a \leq x$, $a < x$, in a similar way. (As an example of a variable which is not an RL-variable, consider the x in $f(x) \leq c$ or $f(x) < x$.)

If a variable x occurs only as an RL-variable in a clause, it is said to be eligible (and can be eliminated from the clause, as we will see shortly).

We will assume the following interpolation axioms.

$\exists\, x\ (x \leq a)$

$\exists\, x\ (a \leq x \leq b) \leftrightarrow a \leq b$

$\exists\, x\ (a \leq x \leq b \wedge x \leq c) \leftrightarrow a \leq b \wedge a \leq c$

etc., where x does not occur in a, b, c. We also assume the appropriate modifications of these axioms, such as

$\exists\, x\ (a \leq x < b) \leftrightarrow a < b$,

when some or all of the $\leq$'s are replaced by $<$'s.

We will (implicitly) use these axioms to eliminate eligible variables in clauses.

<u>Variable Elimination Rule</u>

If x is eligible in a clause C and x occurs in C in the literals

$$(1) \qquad \begin{array}{ll} a_i \not\leq x\ ; & i = 1, n \\ x \not\leq b_j; & j = 1, m \end{array}$$

then C is replaced by its "resolvent" C' which is gotten by removing the literals (1) from C and replacing them by the literals

$$a_i \not\leq b_j\ ; \qquad i = 1, n; \quad j = 1, m.$$

It should be noted that if either n or m is 0, then no literal is added to replace those deleted. The rule is extended appropriately to include the symbol "<" . It should also be noted that C' would have been obtained by resolving C against one of the interpolation axioms.

Example. $C = a \not\leq x \vee x \not< b$

$C' = a \not< b$

Example. $C = a \not\leq x \vee b \not< x \vee \ell$

$C' = \ell$

Example. $C = a \not\leq x \vee x \not\leq b \vee f(x) \leq c$

x is not eligible so it cannot be eliminated.

When an RL-variable is not eligible, as in clause C of this last example, the variable cannot be eliminated. However, it might become eligible in a later resolvent, as, for example, when C is resolved with the clause $(f(x') \not\leq c \vee D)$ where x' does not occur in D.

Reference [6, pp. 7-8, 13-14] and [7].

2.3 Chain Resolvents

A chain resolvent is obtained when two or more literals (from different clauses) are joined (with unification) to form a contradictory inequality chain. For example, the three clauses

$$f(\ell) \leq f(Z_y) \vee D(y)$$

$$f(Z_a) \leq L$$

$$L < f(\ell) \vee E$$

have the resolvent

$$D(a) \vee E.$$

We impose the following restrictions. No chain is "continued past a variable"; for instance, in the above example, if L is a variable, then the resolvent of the first two clauses is

$$f(\ell) \leq L \vee D(a).$$

Furthermore, when one of the links of the chain has the form

$$A \leq B + x,$$

where x is a variable, the chain is continued and the variable "carried along." For example, the clauses

1. $f(\ell) \leq f(Z_y) + \epsilon \vee \epsilon \leq 0$
2. $f(Z_y) \leq L$
3. $L \leq f(t_0)$
4. $f(t_0) < f(\ell)$

where ϵ is a variable, have the resolvent

5. $f(\ell) \leq f(t_0) + \epsilon \vee \epsilon \leq 0.$

This can be converted (see Sect. 2.2) to

6. $f(\ell) \leq f(t_0)$

which resolves with 4. to $\square$. Notice that if we had (incorrectly) resolved 1-4 to

5. $0 < \epsilon \vee \epsilon \leq 0,$

(as would seem natural), the result is a useless tautology.

A better understanding of this procedure can be had by studying the algorithm CHAINER given below and examining the examples of Section 4.

Of the four types of resolvents only two, regular resolvent and chain resolvents, are constructed during the regular resolution cycles; the other two, TY-resolvents and variable-elimination resolvents, are produced by processing other resolvents and preprocessing the initial clauses.

A "splitting" procedure is used here (see below) which insures that, among other things, ground literals occur only in unit clauses. This greatly enhances the usefulness of the TY clause.

2.4 Processing Resolvents

When a new resolvent, R, is formed:

1. If R is $\square$ the proof is successfully terminated.
2. If R is a unit inequality ground clause it is "resolved" with TY.
3. If R has an eligible variable, that variable is eliminated by the methods of Section 2.2.
4. Otherwise R is (factored and) added to the set of clauses (with new standardized apart variables). Ordinary subsumption and tautology removal are used.
5. If R can be split into two or more independent (no variables in common) sub-clauses $R_1,\ldots,R_n$, then RESOLUTION is called on $S \cup \{R_i\}$, for $i = 1,2,\ldots,n$.

2.5 Sequencing

At the beginning of a proof the theorem to be proved is converted to clausal form. All unit ground inequality clauses are "conjoined" together to form the special clause, TY. These unit clauses are also retained as separate clauses. TY is checked for consistency by a call to the function CONTRADICTION. If it is inconsistent, the proof is successfully terminated.

Also, at the beginning any variable x that is eligible in a clause is eliminated in that clause by the procedure of Section 2.2. And splitting is performed where possible. (It should be noted at this point that ground clauses can be split completely, and that this causes an excessive amount of splitting when the set S contains only ground clauses. However, this prover is designed to handle difficult non-ground theorems where very little splitting takes place. See the examples of Section 4.)

The procedure is then to perform Resolution by the level saturation method until $\square$ is obtained or until the computation is aborted. However, there are certain restrictions which are detailed in the algorithms of Section 3. Loosely speaking, a literal L is selected from a clause and "resolved away" by one of the methods. RL-units are not selected to be resolved upon until all other literals in the clause are selected. Only ground inequality literals are resolved against TY (and then only when a new resolvent is generated). Paramodulation [9] is used to affect equality substitution.

3. The Principal Parts

(RESOLUTION$\leq$ Th)

This is the top-level function, where Th is the theorem to be proved. It returns T or NIL.

1. Convert Th to clausal form, getting the set S of clauses.
2. Call INITIAL-TY

   This constructs the special clause TY (not a member of S). If TY is $\square$ the calculation is successfully terminated.
3. Call INITIAL-RL

   This eliminates any eligible variables from the original clauses. Then each clause is REDUCED and then ordered, with RL-literals last. If $\square$ is obtained, the calculation is successfully terminated. The result is a set S of clauses. Subsumption and tautology removal are also used.
4. Call (SPLIT S)

   If L is a literal of a non-unit clause C of S, and L has no variable in common with C $\sim \{L\}$, then call both

   $$(\text{SPLIT } (S \sim \{C\} \cup \{\{L\}\}))$$

   and

   $$(\text{SPLIT } (S \sim \{C\} \cup \{C \sim \{L\}\}).$$
5. Else put S_0 = S, and
6. Let C be the next clause of S_0 (the top clause).
7. Call (RESOLUTION-CHAIN S C)

   If it returns T, return T. Else go to 8.
8. If S_0 has no more clauses, return NIL. Else go to 6.

Once the splitting of S (if any) is completed in step 4 above, the program does Resolution to complete the proof, where we include chain resolvents as well as ordinary resolvents, and where these resolvents are processed to produce, in some cases, TY-resolvents and variable elimination resolvents.

The algorithm presented here in steps 5-8 and in the routines RESOLUTION-CHAIN and RESOLVE-CHAIN are given to show one way that these new resolvents can be used in an actual resolution program. This procedure resembles linear resolution with ordering of literals. Completeness is not claimed (see Section 6).

(RESOLUTION-CHAIN S C)

Called by RESOLUTION$\leq$. Returns T or NIL. C is called the "top clause."

```
1.  Put $C_0$ = C.
2.  Let L be the next literal in $C_0$.
3.  Call (RESOLVE-CHAIN S C L)
        If it returns T, return T. Else go to 4.
4.  If $C_0$ has no more literals, return NIL. Else go to 2.
```

(RESOLVE-CHAIN S C L)

Called by RESOLUTION-CHAIN and PROCESS-RESOLVENT. Returns T or NIL.

This tries to prove that S is unsatisfiable by resolving upon the literal L of C.

```
1.  Put R = (RESOLVE C L)
        If R = $\square$, return T.
        If R = NIL go to 2.
        Else return (PROCESS-RESOLVENT R).

2.  If L is not an inequality literal, return NIL. Else put R = (CHAINER C L).
        If R = $\square$, return T.
        If R = NIL, return NIL.
        Else return (PROCESS-RESOLVENT R).
```

If neither RESOLVE nor CHAINER succeeds, then NIL is returned and RESOLVE-CHAIN is called with another L from C. If a resolvent R is produced, then PROCESS-RESOLVENT processes it, adds it to S, and recalls RESOLUTION-CHAIN with R as the new top clause.

(RESOLVE C L)

Called by RESOLVE-CHAIN. Returns a clause.

This is traditional Resolution. An attempt is made to resolve C upon L with each clause of S. The ordinary resolvent is returned.

In matching inequalities we require

$a \leq b$ to match against $b < a$ (as usual),

$a < b$ to match against $b < a$,

$a \leq b$ to match against $b \leq a$,

but in this last case the literal $(a = b)$ is added to the resolvent.

Paramodulation (or some form of equality substitution) is used.

An inequality literal of the form $(x \leq y)$ or $(x < y)$ where both x and y are variables is never allowed to match with another literal. (Hopefully, these cases are handled by variable-elimination.)

(CHAINER C L)

Called by RESOLVE-CHAIN. Returns T or NIL.

C is the top clause of S, L is a literal of C, L has the form ($\not\leq$ A B) or ($\not<$ A B), where A and B are terms, L is not an RL-literal (e.g., L is not of the form ($\leq$ x B) where x is a variable that does not occur in B).

An attempt is made to chain from left to right (e.g., $A \leq A_1 \leq A_2 \leq B$) and, if that fails, from right to left (e.g., $B \geq B_1 \geq B_2 \geq A$). If either succeeded, the resolvent R is formed by appending the non-chained-upon parts of the clauses used in the chain. The function CHAIN is used for this purpose. Put

- C_{LR} = (CHAIN T A B '$\leq$ NIL T 0)

If C_{LR} is not NIL return R = ((C ~ {L}) $\cup$ C_{LR}), else put

- C_{RL} = (CHAIN NIL B A '$\leq$ NIL T 0)

If C_{RL} is not NIL, return R = ((C ~ {L}) $\cup$ C_{RL}), else return NIL.

If A is a variable we omit calculating C_{LR}, and if B is a variable we omit calculating C_{RL}.

(CHAIN DIR AA BB SYM D θ E)

Called by CHAINER. Returns T or NIL.

DIR is T or NIL.

If DIR = T, the chaining direction is from left to right.

Else it is from right to left.

AA and BB are terms, which initially are A and B, respectively, if DIR is T, and B and A if DIR is NIL.

SYM is '$\leq$ or '$<$. It starts a '$\leq$, and changes to '$<$ if a '$<$ occurs in the chain.

D is a set of literals, which is part of the eventual resolvent.

θ is a substitution, which is the cumulative substitution to this point in the chain.

E is a term (a sum of variables).

ALGORITHM CHAIN

- If AA has the form (A'+x) where x is a variable, then return

  (CHAIN DIR A' BB SYM D θ (E+x)).

- If AA is a variable not occurring in BB, or BB is a variable not occurring in AA, return

  (RL-CASE)

- If AA and BB are unifiable with mgu θ, and DIR = T, return

  (MATCH-CASE)

The rest of the algorithm will be described as if DIR=T (i.e., as if the chaining is from left to right). A similar procedure is used when DIR = NIL.

```
1.  Put SS = S.
2.  Choose the next clause C' from SS.
3.  Choose the next literatl L' = (sym'A'B') from C'.
4.  Put $\Theta'$ = unify (AA,A').
    4.1  If $\Theta$ = NIL, go to 5.
    4.2  Else put
            $\Theta''$   = $\Theta \circ \Theta'$
            D''   = [D $\cup$ (C' $\sim$ L')]$\Theta$
            SYM'' = '< if sym = '< and sym' = '$\leq$, else '$\leq$.
            CH    = (CHAIN DIR B'$\Theta$ BB SYM'' D'' $\Theta''$ E)
    4.3  If CH = T, return T.
    4.4  Else go to 5.
5.  If no more literals in C' go to 6.  Else go to 3.
6.  If no more clauses in S, return NIL.  Else go to 2.
```

In the above algorithm we do not allow "chaining across a variable." To ensure this, we skip step 4 (and go to step 5) if either AA or A' is a variable.

(MATCH-CASE)

Called by CHAIN. It returns NIL or a resolvent clause R.

ALGORITHM MATCH-CASE

Put $D'' = D \cup (C \sim \{L\})\theta$

IF	RETURN
$E = \theta$	D''
$E \neq 0$	$(D'' \cup \{(\text{sym}''\ A\ (B + E))\})$[‡]
chain-length ≤ 2	NIL

(RL-CASE)

Called by CHAIN. It returns NIL or a resolvent clause R.

The arguments and variables of CHAIN and CHAINER are visible to it.

This case arises (in CHAIN) when either AA or BB is a variable x not occurring in the other.

[‡]Here we let sym" = '$<$ if sym = '$<$ and sym' = '$\leq$, else '$\leq$.

ALGORITHM RL-CASE

If chain-length = 1, return NIL. Else

$$\text{Put } R = (D \cup (C \sim \{L\})\theta \cup \{\{L''\}\}),$$

where: If AA = x (a variable not occurring in BB)

If	Put L"
DIR = T, E = 0	~ (sym" x BB)
DIR = T, E ≠ 0	(sym' A x + E)
DIR = NIL, E = 0	~ (sym" BB x)
DIR = NIL, E ≠ 0	(sym' x B + E)

and: If BB = x (a variable not occurring in AA)

If	Put L"
DIR = T, E = 0	~ (sym" AA x)
DIR = T, E ≠ 0	(sym' A AA + E)
DIR = NIL, E = 0	~ (sym" x AA)
DIR = NIL, E ≠ 0	(sym' AA B + E),

In RL-CASE we do not allow the chain-length to be 1 because that would cause, in certain cases, the input clause C to be returned as the resolvent. In MATCH-CASE we also do not allow the chain-length to be 2 because that case is handled by RESOLVE (ordinary resolution).

(PROCESS-RESOLVENT R)

This is called by the routines RESOLVE and RL-CASE, when a new resolvent R has just been produced.

- Put R = (REDUCE R).
- If R = $\square$, return T.
- If R is a ground inequality unit, call (CONTRADICTION TY R)
- Put R = (ELIMINATE-VARIABLES R).
- If R = $\square$, return T.
- If R is a tautology, return NIL.
- If R can be split on L (i.e., $C \sim \{L\}$ is not empty and L and $C \cap \{L\}$ have no variable in common)

  (PROCESS-RESOLVENT {L})

  and

  (PROCESS-RESOLVENT ($C \sim \{L\}$)).
- (SUBSUME R S)

  Returns NIL if R is subsumed by S, and removes from S clauses subsumed by R.
- Put R = (SORT R)

  Sort the literals of R so that RL-literals are last.
- Replace C by R in S.
- Return (RESOLVE-CHAIN S R).

(REDUCE R)

This is a procedure which rewrites certain formulas as others [10]. For example, each of the formulas $(0 < 1)$, $(A + 5 \leq A + 6)$ is rewritten as T, whereas each of $(2 \leq 1)$, $(f(x) + 1 \leq f(x))$, is rewritten as $\square$, and

$$((2 \leq 1) \ (A \leq B))$$

is rewritten as $((A \leq B))$.

An algebraic simplifier is used in various parts of the program.

THE SPECIAL CLAUSE TY

TY is a conjunction of ground inequality literals which may be altered, as the proof proceeds, by conjoining onto it additional ground inequality units. The initial value of TY is gotten by a call to INITIAL-TY which combines all the ground inequality unit clauses of S into one conjunction. If TY is or becomes contradictory, then the proof is successfully terminated. A function (CONTRADICTION TY L) is called to determine whether TY is indeed contradictory. If L is not NIL, it is first conjoined onto TY before the determination is made.

CONTRADICTION is called by INITIAL-TY and called as (CONTRADICTION TY R) by PROCESS-RESOLVENT in the case when the resolvent R is a ground inequality unit clause. In that case TY is augmented, and this new value of TY is retained in the remainder of the proof. If CONTRADICTION does not return $\square$, it might infer from TY a set E of equality units (as, for example, would be the case if R was the unit ($\leq$ AB) and TY already had the conjunct ($\leq$ B A)). In this case, these equality units are applied to TY and all of S by a special equals substituting routine.

Any ground inequality package such as those described in [4, 5, 6] can be used to handle the functions of CONTRADICTION. Our implementation has used the one described in [6, pp. 7-8].

(ELIMINATE-VARIABLES C)

This is called by INITIAL-RL and PROCESS-RESOLVENT. If the clause C has variables which are eligible in C (see Variable-elimination Resolvents, Section 2.2), then they are removed from C using the methods of Section 2.2 and the resultant clause returned.

4. Examples

Here we list some examples along with their proofs by Resolution$\leq$. Some of these have been proved by our LISP program (see Section 5). Many (non-useful) resolvents are not listed in these proofs. The search space is in most cases much larger than that shown.

The first few examples are trivial and are listed only to illustrate the methods.

Ex. 1. $(a < b \rightarrow a \leq b)$

$$\left.\begin{array}{ll} 1. & a < b \\ 2. & b < a \end{array}\right\} \text{original set of clauses}$$

TY: $[a < b, b < a]$ added by preprocessing

Since TY is inconsistent, $\square$ is obtained during preprocessing and the proof is complete.

Ex. 2. $(\forall x\ (f(x) \leq c) \rightarrow f(a) \leq c \wedge f(b) \leq c)$

1. $f(x) \leq c$
2. $c < f(a) \vee c < f(b)$

Preprocessing splits clause 2, getting the two cases 2.1 and 2.2.

Ex. 2.1

1. $f(x) \leq c$
2. $c < f(a)$

3. $\square$ 1, 2, a/x

Ex. 2.2

1. $f(x) \leq c$

2. $c < f(b)$

3. $\square$ 1, 2, b/x

Ex. 3. $\exists x\ (x \leq a)$

1. $a < x$

2. $\square$ 1, variable elimination (note that x is "eligible" in clause 1, see Section 2.2)

Ex. 3A. $a \leq b \rightarrow \exists x\ (x \leq a)$

1. $a \leq b$

2. $a < x$

3. $\square$ 2, variable elimination (note that clause 1 is not used).

Ex. 4. $(a \leq b \rightarrow \exists x\ (a \leq x \leq b))$

1. $a \leq b$

2. $x < a \vee b < x$

3. $b < a$ 2, variable elimination (x is eligible in Clause 2)

4. $\square$ 1, 3

In this example we omitted writing the special clause TY since it was the single clause 1. The actual procedure is as follows:

TY: $[a \leq b]$ Preprocessing

3. $b < a$ 2, variable elimination

TY $[a \leq b,\ b < a]$, $\square$ process-resolvent, 3

<u>Ex. 5</u>. $(\forall x \forall y (f(x) \leq f(y) \rightarrow x \leq y) \wedge f(a) \leq f(b) \rightarrow a \leq b)$

1. $f(y) < f(x) \vee x \leq y$ (top clause)

2. $f(a) \leq f(b)$

3. $b < a$

TY: $[b < a]$

4. $a \leq b$ 1, 2 a/x, b/y

TY: $[b < a, a \leq b]$, $\square$ process-resolvent, 4

Notice that we did not resolve upon the literal $x \leq y$ of clause 1, because it is an RL-literal (so other literals are resolved upon first) and also because we forbid resolving upon a double-variable literal of the form $x \leq y$ where both x and y are variables.

<u>Ex. 6</u>. $(\forall x \forall y\ (x \leq y \rightarrow f(x) \leq f(y))$

$\wedge \forall z (1 \leq z \rightarrow f(f(z)) < f(z+1))$

$\wedge\ 1 \leq x_0 \rightarrow f(x_0) < x_0 + 1)$

1. $f(x) \leq f(y) \vee y < x$ (top clause)

2. $f(f(z)) < f(z+1) \vee z < 1$

3. $1 \leq x_0$

4. $x_0 + 1 \leq f(x_0)$

TY: $[1 \leq x_0, x_0 + 1 \leq f(x_0)]$

5. $f(z) < z + 1 \vee z < 1$ 1, 2, $f(z)/y$, $z+1/x$

6. $x_0 < 1$ 5, 4, x_0/z

7. $\square$ 6, TY

Ex. 7. $(f(\ell) < 0 \wedge 0 \leq f(b) \wedge \ell < c \wedge b \leq \ell$

$\rightarrow \exists y[\forall z(z \leq b \wedge f(z) \leq 0 \rightarrow z \leq y) \wedge y < \ell])$

1. $f(\ell) < 0$
2. $0 \leq f(b)$
3. $\ell < c$
4. $b \leq \ell$
5. $z_y \leq b \vee \ell \leq y$
6. $f(z_y) \leq 0 \vee \ell \leq y$
7. $y < z_y \vee \ell \leq y$ (top clause)

TY: $[f(\ell) < 0,\ 0 \leq f(b),\ \ell < c,\ b \leq \ell]$

8. $\ell \leq b$ 7, 5, b/y
9. $\square$ 8, TY

Note that when clause 8 is added to TY, the program first infers that $b = \ell$ and then uses that to reach the contradiction, $0 \leq f(b) < 0$.

Ex. 8. $(0 \leq a \wedge \forall s\ (0 \leq s \rightarrow f(b) \leq f(s)) \wedge f(a) \leq f(c) \rightarrow f(b) \leq f(c))$

1. $0 \leq a$
2. $f(b) \leq f(s) \vee s < 0$
3. $f(a) \leq f(c)$
4. $f(c) < f(b)$ (top clause)

TY: $[0 \leq a,\ f(a) \leq f(c),\ f(c) < f(b)]$

5. $c < 0$ 4, 2, c/s

This resolvent is useless, so CHAINER is called.

6. $a < 0$ CHAIN: 4, 2, 3, a/s
7. $\square$ 6, TY

Ex. 9. $a \leq 2 \leq b \rightarrow \exists x \ (0 \leq x \leq 5 \wedge a \leq x)$

1. $a \leq 2$

2. $2 \leq b$

3. $0 \not\leq x \vee x \not\leq 5 \vee a \not\leq x$[‡]

TY: $[a \leq 2, \ 2 \leq b]$

4.	$0 \not\leq 5 \vee a \not\leq 5$	3, variable elimination
5.	$a \not\leq 5$	4, REDUCE
6.	$\square$	5, TY

Ex. 10. $\exists x \ \exists u \ \forall s \ (s = x \rightarrow s \leq u)$

1. $s_{x,u} = x$

2. $u < s_{xu}$

3.	$u < x$	1, 2, sub =
4.	$\square$	3, variable elimination[†]

Ex. 11. $(\forall y \ (\ell \leq y \vee y < b) \rightarrow \ell \leq b)$

1. $\ell \leq y \vee y < b$

2. $b < \ell$ (TY)

3.	$\ell \leq b$	1, variable elimination
4.	$\square$	3, 2 (= TY).

[‡]In this presentation we sometimes use the notation $x \not\leq y$ instead of its equivalent $y < x$, and $x \not< y$ instead of $y \leq x$, but the program always converts such expressions so that no negations (of inequalities) are used.

[†]Here x and u are variables. In this and other examples, the reader can determine which symbols are variables by the quantification in the statement of the theorem.