VARIABLE ELIMINATION AND CHAINING IN A RESOLUTION-BASED PROVER FOR INEQUALITIES

by

W. W. Bledsoe and Larry M. Hines

April 1980 ATP-56A

This work was supported by NSF Grant MCS 77-20701.

This paper will appear in the proceedings of the 5th Conference on Automated Deduction, Les Arcs, France, July 8-11, 1980.

Variable Elimination and Chaining in a Resolution-based Prover for Inequalities

By

W. W. Bledsoe and Larry M. Hines
The University of Texas, Austin

Abstract. A modified resolution procedure is described which has been designed to prove theorems about general linear inequalities. This prover uses "variable elimination", and a modified form of inequality chaining (in which chaining is allowed only on so called "shielding terms"), and a decision procedure for proving ground inequality theorems. These techniques and others help to avoid the explicit use of certain axioms, such as the transitivity and interpolation axioms for inequalities, in order to reduce the size of the search space and to speed up proofs. Several examples are given along with results from a computer implementation. In particular this program has proved some difficult theorems such as: The sum of two continuous functions is continuous.

1. Introduction

The purpose of this paper is to describe a resolution-based theorem prover which has been designed to prove theorems about linear real inequalities. It is an improvement of the prover described in [1] with a different, more powerful, concept of chaining.

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

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

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

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 numbers 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, etal. [4], Shostak [5], Bledsoe, etal. [6] (these tend to be in the Presburger mode);
(3) the methods of Hodes [7];
(4) the Restriction Intervals Method [6].

Even though these provers have met with a degree of success, still further changes are necessary to handle more difficult inequality theorems. Two such changes are the inequality chaining and variable elimination described below.

The class of formulas dealt with here are those from the theory of dense linear order without endpoints. This theory is decidable. But, we also permit arbitrary quantification and uninterpreted function symbols and therefore can encode all of predicate calculus. Our intention, however, was not to provide a general-purpose prover for first-order logic but rather to be able to more easily prove naturally arising inequality theorems.

We also permit the + sign with its usual semantics, but we do not, in this version, allow the + sign to occur within the arguments of an uninterpreted function symbol (e.g., $x + f(x_0)$ is allowed but $f(x + x_0)$ is not). (Such cases could be handled by including an algebraic unifier such as [8].)

The strategy used by this prover is to eliminate variables from literals, so that these ground literals can then be split off and proved by a ground inequality prover. However, in order to eliminate a variable from a clause, it must be eligible in that clause (see Section 2.2), so a first objective is to make variables eligible by removing shielding terms (see Section 2.3). This is done by chaining (only) on shielding terms.

The use of this principle in difficult examples like Example 17 (sum of two continuous functions is continuous), Example 13, and others, apparently makes the difference in whether or not a proof will be obtained in reasonable time.

Except for the ground inequality prover, where equality substitution is used [6], we have avoided the use of the equality symbol by substituting $(a \leq b \wedge b \leq a)$ for each instance of $a = b$.

2. Resolution $\leq$

Resolution$\leq$ is much like ordinary resolution [9], 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 three different types of resolvents are used. These are

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

Ordinary resolvents are not explicitly computed. However, they can be implicitly produced by SELF-CHAIN (see below).

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, GROUND-PROVER. If GROUND-PROVER 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.

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

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

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

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

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

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

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

Variable Elimination Rule

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

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

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

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

It should be noted that if either n or m is zero, 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 the clauses from one of the interpolation axioms.

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

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.

2.3 Shielding Terms

If L is a literal which is equivalent to $t \leq A$ (or $t < A$, $A \leq t$, $A < t$) and t is of the form $f(t_1, t_2, \ldots, t_n)$, where f is an uninterpreted function symbol, and t contains at least one variable, then we say that t is a shielding term of L. In the examples of Section 2.2, the shielding terms are $f(x)$ and $f(x')$. A shielding literal is one that contains a shielding term.

The chaining procedure below is designed to remove such shielding terms from clauses.

2.4 Chain Resolvents

Chaining is a procedure which effects a limited application of the transitivity axiom

$$\forall xyz\,(x \leq y \wedge y \leq z \rightarrow x \leq z)$$

so that if b and b' are unifiable, with mgu θ, then $(a < c)\theta$ is inferred from $a < b$ and $b' < c$.

The chaining procedure in this paper is applied only when b is a shielding term.

If C and C' are the two clauses

$$A \leq B \vee P$$

and

$$B' \leq C \vee Q\ ,$$

where B or B' is a shielding term, and B and B' are unifiable with mgu θ, then

$$(A \leq C)\theta \vee P\theta \vee Q\theta$$

is called a chain-resolvent of clauses C and C'. (Similarly for $A < B$, $B' < C$, etc.)

For example, $f(y)$ is a shielding term in the clause

$$C = f(\ell) \leq f(y) \vee y < \ell$$

which can be removed by resolving C against the clause

$$f(s) \leq A \vee b < s$$

to obtain the chain-resolvent

$$R = f(\ell) \leq A \vee y < \ell \vee b < y \ .$$

Although y was not eligible in C, it is in R and, therefore, can be eliminated to obtain

$$f(\ell) \leq A \vee b < \ell \ .$$

Another example is found in Example 17 below, where clauses

$$C_1 = f(z_y) + g(z_y) < f(x_0) + g(x_0) + \varepsilon_0 \vee y \leq 0 \ ,$$
$$C_2 = f(x_0) + \varepsilon \leq f(s)$$

are resolved to obtain the chain-resolvent

$$f(x_0) + \varepsilon + g(z_y) < f(x_0) + g(x_0) + \varepsilon_0 \vee y \leq 0 \ .$$

(Actually, in order to use the definition of chain-resolvent as given above, we had to first rewrite C_1 in its equivalent forms $f(z_y) < -g(z_y) + f(x_0) + g(x_0) + \varepsilon_0 \vee y \leq 0$.)

We also allow "self-chaining" resolvents (see SELF-CHAIN) whereby the resolvent $(P\theta)$ is obtained from the clause $(a < a' \vee P)$ if θ is the mgu of a and a'.

2.5 Processing Resolvents

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

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

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 simplified 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) subclauses $R_1, \ldots, R_n$, the process is continued for each of $S \cup \{R_i\}$, for $i = 1, 2, \ldots, n$.

2.6 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 GROUND-PROVER. 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 to compute resolvents by a method similar to linear resolution.

A top clause C is chosen from S which is not ground. There is such a C, since otherwise S is a set of unit ground clauses and would have already been processed by GROUND-PROVER. Furthermore, at least one shielding term occurs in a literal L of C.

Loosely speaking, such an L is selected from C and the chaining algorithm is applied to the shielding terms of L. As variables become eligible, they are eliminated. If ground literals arise they are split off and resolved against TY. Also, each new resolvent R is simplified by REDUCE and duplicate literals are merged, and the process is repeated with R as the new top clause.

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-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.

3. 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

(SPLIT (S ~ {C} ∪ {C ~ {L}})) .

4. Call INITIAL-TY

   This constructs the special clause TY (not a member of S). If TY is □ the calculation is successfully terminated.

5. Return (REMOVE-SHIELDING-TERMS S).

Once the splitting of S (if any) is completed in step 3 above, and the TY is initialized, the program trys to produce "chain resolvents" which are processed to produce, in some cases, TY-resolvents and variable elimination resolvents.

The algorithms presented here in the routines REMOVE-SHIELDING-TERMS, CHAIN, and SELF-CHAIN, are given to show one way that these new resolvents can be used in an actual prover. This implementation resembles linear resolution with ordering of literals. Completeness is not claimed.

(REMOVE-SHIELDING-TERMS S)

This is called by RESOLUTION$\leq$ and by PROCESS-RESOLVENT. S has at least one non-ground clause. (Otherwise, GROUND-PROVER would have already handled such a case.)

1. While S ≠ NIL do

1.1 Select a non-ground clause, C from S (a new one not yet chosen). C has at least one shielding literal. (Otherwise C is either ground or any variable occurring in it would be eligible and hence would have been eliminated.) Put S = S ~ {C}. Put C' = C.

1.2 While C' ≠ NIL do

1.21 Select a new shielding literal L from C'. Put C' = C' ~ {L}. Let STR be the set of shielding terms of L. Let LE be the predicate of L (i.e., "<" or "$\leq$"). Let LS be the "left side" of L. (i.e., LS is such that (LE LS 0) is equivalent to L).

1.22 While STR ≠ NIL do

1.221 Select a new shielding term TR from STR. Put STR = STR ~ {TR}.

1.222 Call (CHAIN TR LS LE L C S). If it returns T, return T.

END;

END;

END;

2. Return NIL.

(CHAIN TR LS LE L C S)

This is called by REMOVE-SHIELDING-TERMS. The literals in each clause have already been put into a "left side" form (e.g., A $\leq$ B is transformed to A ~ B $\leq$ 0).

1. Set SS = S ~ {C}.

2. While SS ≠ NIL do

2.1 Let CC be the next clause in SS. Put SS = SS ~ {CC}.

2.2 While CC ≠ NIL do

2.21 Let LL be the next literal in CC. Put CC = CC ~ {LL}. Let TRS be the set of terms in LL which are not variables or numbers. Let LEE be the predicate of LL. Let LSS be the "left side" of LL (i.e., LSS is such that (LEE LSS 0) is equivalent to LL).

2.22 While TRS ≠ NIL do

2.221 Let TRR be the next term in TRS. Put TRS = TRS ~ {TRR}.

2.222 Put θ = (UNIFY TR (- TRR)).
If θ ≠ NIL then

2.2221 Let LE be "≤" if both L and LL have "≤" as their predicates. Else let LE be "<". Let New-L be (LE (+ LSθ LSSθ) 0). Let R be {New-L} ∪ (Rest of C)θ ∪ (Rest of CC)θ. Call (PROCESS-RESOLVENT R). If it returns T, then return T.

END;

END;

END;

3. Return (SELF-CHAIN C L TR).

(SELF-CHAIN C L TR)

Called by CHAIN.

1. Let TRS be the set of terms in L which are not variables or numbers. Put TRS = TRS ~ {TR}.

2. While TRS ≠ NIL do

2.1 Let TRR be the next term in TRS. Put TRS = TRS ~ {TRR}.

2.2 Put θ = (UNIFY TR (- TRR)).

2.3 If θ ≠ NIL and (PROCESS-RESOLVENT Cθ) ≠ NIL then return T. END;

3. Return NIL.

For example, if $C = (f(x)-f(x_0) \leq 0 \vee x \leq a)$, $L = (f(x)-f(x_0) \leq 0)$, $TR = f(x)$, then $R = (f(x_0)-f(x_0) \leq 0 \vee x_0 \leq a)$ which is simplified to $x_0 \leq a$.

(PROCESS-RESOLVENT R)

This is called by the routines CHAIN and SELF-CHAIN, 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 (GROUND-PROVER 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 ~ {L} is not empty and L and C ~ {L} have no variable in common).

  (PROCESS-RESOLVENT {L})

  and

  (PROCESS-RESOLVENT (C ~ {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 (but leave C in S).
- Return (REMOVE-SHIELDING-TERMS S).

(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 $(2 \leq 1) \vee (A \leq B))$ 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 (GROUND-PROVER 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.

GROUND-PROVER is called by INITIAL-TY and called as (GROUND-PROVER 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 GROUND-PROVER 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 $(\leq A\ B)$ 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 GROUND-PROVER. 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, the resultant clause is returned, and C is discarded.

4. Examples

Here we list some examples which have been proved by our LISP program. In most cases, the prover follows closely the presentation given here, with few non-useful resolvents being generated.

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

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

1. $a < b$
2. $b < a$

} 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.

Example 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.

Example 2.1.

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

---

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

Example 2.2.

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

---

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

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

1. $a \leq b$
2. $a < x$

---

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

Example 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.

Example 5. $(\forall x\ \forall y\,(f(x) \leq f(y) \rightarrow x \leq y) \wedge f(a) \leq f(b) \rightarrow a \leq b)$

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

TY:	$[b < a,\ f(a) \leq f(b)]$	
4.	$f(a) < f(x) \vee x \leq b$	1, 2 b/y (removing f(y))
5.	$a \leq b$	4, Self-chain
TY:	$[b < a,\ f(a) \leq f(b),\ a \leq b], \square$	process-resolvent, 5.

Notice that we did not resolve upon the literal $x \leq y$ of clause 1 (because it is an RL-literal), but did chain on the shielding term $f(y)$.

Example 6. $(f(\ell) < 0 \wedge 0 \leq f(b) \wedge \ell < c \wedge b \leq \ell$
$\rightarrow \exists y\,[\forall z\,(z \leq b \wedge f(z) \leq 0 \rightarrow z \leq y) \wedge y < \ell])$

1. $f(\ell) \leq 0$
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. $y < z_y \vee \ell \leq y$

TY:	$[f(\ell) < 0,\ 0 \leq f(b),\ \ell < c,\ b \leq \ell]$	
8.	$y < b \vee \ell \leq y$	7, 5 (removing z_y)
9.	$\ell \leq b$	8, variable elimination
10.	$\square$	9, TY.

Here z_y is a skolem expression, i.e., a skolem function applied to the variable y. We will use this method of representing skolem expressions throughout this paper. The reader can determine which symbols are variables by the quantification in the statement of the theorem.

Note that when clause 9 is added to TY, the program first infers that $b = \ell$ and then uses that to reach the contradiction, $0 \leq f(b) < 0$.

Example 7. $a \leq 2 \leq b \rightarrow \exists x\,(0 \leq x \leq 5 \wedge a \leq x)$

1. $a \leq 2$	TY:	$[a \leq 2,\ 2 \leq b]$	
2. $2 \leq b$	4.	$0 \not\leq 5 \vee a \not\leq 5$	3, variable elimination
3. $0 \not\leq x \vee x \not\leq 5 \vee a \not\leq x$[†]	5.	$a \not\leq 5$	4, REDUCE
———	6.	$\square$	5, TY.

Example 8. $(\forall y\,(y < \ell \rightarrow \exists z\,(y < z \leq b)) \wedge a \leq \ell \rightarrow a \leq b)$

1. $z_y \leq b \vee \ell \leq y$	TY:	$[a \leq \ell,\ b < a]$	
2. $y < z_y \vee \ell \leq y$	5.	$y < b \vee \ell \leq y$	1, 2 (removing z_y)
3. $a \leq \ell$	6.	$\ell \leq b$	5, variable elimination
4. $b < a$	7.	$\square$	6, TY.
———			

Example 9. $(\forall \varepsilon\,(0 < \varepsilon \rightarrow A \leq B + \varepsilon) \rightarrow A \leq B)$

1. $A \leq B + \varepsilon \vee \varepsilon \leq 0$ (ε is a variable)	TY:	$[B < A]$	
2. $B < A$	3.	$0 \leq B - A$	1, eliminate variable ε
———	4.	$\square$	3, TY.

Example 10. $\exists \varepsilon\,[(0 < \varepsilon \rightarrow A \leq B_\varepsilon + \varepsilon) \wedge B_\varepsilon \leq C \rightarrow A \leq C]$

1. $A \leq B_\varepsilon + \varepsilon \vee \varepsilon \leq 0$	TY:	$[C < A]$	
2. $B_\varepsilon \leq C$	4.	$A \leq C + \varepsilon \vee \varepsilon \leq 0$	1, 2 (removing B_ε)
3. $C < A$	5.	$A \leq C$	4, variable elimination
———	6.	$\square$	5, TY.

Example 11. $\exists \varepsilon\,[(0 < \varepsilon \rightarrow A_\varepsilon \leq B_\varepsilon + \varepsilon) \wedge B_\varepsilon \leq C \rightarrow A_\varepsilon \leq C]$ (Not a Theorem)

1. $A_\varepsilon \leq B_\varepsilon + \varepsilon \vee \varepsilon \leq 0$
2. $B_\varepsilon \leq C$
3. $C < A_\varepsilon$

———

4. $A_\varepsilon \leq C + \varepsilon$	1, 2 (removing B_ε)	
5. $C < C + \varepsilon \vee \varepsilon \leq 0$	4, 3 (removing A_ε)	(Tautology)
4. $C < B_\varepsilon + \varepsilon \vee \varepsilon \leq 0$	1, 3 (removing A_ε)	
5. $C < C + \varepsilon \vee \varepsilon \leq 0$	4, 2 (removing B_ε)	(Tautology)

FAILURE.

———

[†] 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.

Example 12. $[\forall \varepsilon\ (\varepsilon > 0 \to f(\ell) \le f(z_\varepsilon) + \varepsilon \wedge f(z_\varepsilon) \le f(t_0)) \to f(\ell) \le f(t_0)]$

1.	$f(\ell) \le f(z_\varepsilon) + \varepsilon \vee \varepsilon \le 0$	4. $f(\ell) \le f(t_0) + \varepsilon \vee \varepsilon \le 0$	1, 2 (removing $f(z_\varepsilon)$)
2.	$f(z_\varepsilon) \le f(t_0) \vee \varepsilon \le 0$	5. $f(\ell) \le f(t_0)$	4, variable elimination
3.	$f(t_0) < f(\ell)$	6. $\square$	5, TY.

TY: $[f(t_0) < f(\ell)]$

In Example 13 and later examples, we often indicate the literal being resolved upon by outlining it with a rectangle, $\boxed{\text{L}}$. This is usually the first literal of the top clause after it has been processed and sorted.

Example 13. $[a \le \ell \le b \wedge a \le t_0 < \ell$
$\wedge \forall \varepsilon\ (\varepsilon > 0 \to \exists r\ (r < \ell \wedge \forall s\ (r \le s \le \ell \to f(\ell) \le f(s) + \varepsilon)))$
$\wedge \forall y\ (a \le y < \ell \to \exists z\ (y < z \le \ell \wedge \forall t\ (a \le t < z \to f(z) \le f(t))))$
$\to f(\ell) \le f(t_0)]$

1. $a \le \ell$ 2. $\ell \le b$ 3. $a \le t_0$ 4. $t_0 < \ell$
5. $r_\varepsilon < \ell \vee \varepsilon \le 0$
6. $f(\ell) \le f(s) + \varepsilon \vee \varepsilon \le 0 \vee s < r_\varepsilon \vee \ell < s$
7. $y < z_y \vee y < a \vee \ell \le y$
8. $z_y \le \ell \vee y < a \vee \ell \le y$
9. $\boxed{f(z_y) \le f(t)} \vee y < a \vee \ell \le y \vee t < a \vee z_y \le t$
10. $f(\ell) \not\le f(t_0)$

TY: $[a \le \ell,\ \ell \le b,\ a \le t_0,\ t_0 < \ell,\ f(t_0) < f(\ell)]$

11. $f(\ell) \le f(t) + \varepsilon \vee \boxed{z_y \le t} \vee t < a \vee \ell \le y$
$\vee y < a \vee z_y < r_\varepsilon \vee \ell < z_y \vee \varepsilon \le 0$
9, 6 (removing $f(z_y)$, z_y/s)

12. $y < t \vee f(\ell) \le f(t) + \varepsilon \vee t < a \vee \ell \le y$
$\vee y < a \vee \boxed{z_y < r_\varepsilon} \vee \ell < z_y \vee \varepsilon \le 0$
11, 7 (removing z_y from $(z_y \le t)$)

13. $y < r_\varepsilon \vee f(\ell) \le f(t) + \varepsilon \vee y < t \vee t < a \vee \ell \le y$
$\vee y < a \vee \boxed{\ell < z_y} \vee \varepsilon \le 0$
12, 7 (removing z_y from $(z_y < r_\varepsilon)$)

14. $y < r_\varepsilon \vee y < t \vee f(\ell) \le f(t) + \varepsilon$
$\vee t < a \vee \ell \le y \vee y < a \vee \varepsilon \le 0$
13, 8 (removing z_y from $(\ell < z_y)$)

15. $\ell \le r_\varepsilon \vee f(\ell) \le f(t) + \varepsilon \vee \ell \le t \vee \boxed{\ell \le a}$
$\vee t < a \vee \varepsilon \le 0$
14, variable elimination y

Clause 15 splits on $(\ell \le a)$ into clauses 16.1 and 16.2.

16.1. $\ell \leq a$ 17. $\square$ 16.1, TY

16.2. $\boxed{\ell \leq r_\varepsilon} \vee f(\ell) \leq f(t) + \varepsilon \vee \ell \leq t \vee t < a \vee \varepsilon \leq 0$

17. $f(\ell) \leq f(t) + \varepsilon \vee \ell \leq t \vee t < a \vee \varepsilon \leq 0$ 16.2, 5 (removing r_ε)

18. $\boxed{f(\ell) \leq f(t)} \vee \ell \leq t \vee t < a$ 17, variable elimination ε

19. $\ell \leq t_0 \vee t_0 < a$ 18, 10 (removing f(t))

Clause 19 splits on $(\ell \leq t_0)$ into clause 20.1 and 20.2.

20.1. $\ell \leq t_0$ 20.2 $t_0 < a$

21. $\square$ 20.1, TY 21. $\square$ 20.2, TY.

Examples 14 and 15 arise in the search for counterexamples in a proof.

Example 14. $\exists\, a\ \exists\, b\ \forall x\ \exists\, u\ ([(x < b \to u \leq a) \wedge x \leq u] \wedge (u \leq a \vee x \neq b))$

1. $x_0 < b \vee x_0 \not\leq u \vee u \not\leq a$
2.1. $x_0 < b \vee x_0 \not\leq u \vee x_0 \leq b$
2.2. $x_0 < b \vee x_0 \not\leq u \vee b \leq x_0$
3. $u \not\leq a \vee x_0 \not\leq u \vee u \not\leq a$
4.1. $u \not< a \vee x_0 \not\leq u \vee x_0 \leq b$
4.2. $u \not< a \vee x_0 \not\leq u \vee b \leq x_0$

a, b, u are variables, x_0 stands for the expression x_{ab}.

Clauses 4.1 and 4.2 are subsumed by clause 3, and clause 2.2 is a tautology.

5. $x_0 < b \vee x_0 \not< a$ 1, eliminate u
6.1. $x_0 \leq b$ 2.1, eliminate u
7. $x_0 \not\leq a$ 3, eliminate u
8. $a < b$ 7, 6.1 (removing x_0)
9. $\square$ 8, eliminate a (or b)

Example 15. $\exists\, a\ \exists\, b\ \exists\, \ell\ \exists\, w\ \forall s\ \exists\, u$

$(a \leq b \wedge [u \leq 0 \vee s \neq a] \wedge [0 \leq u \vee s \neq b] \wedge [s \leq w \vee 0 < u] \wedge w \leq \ell)$

The proof of this example, which is like that of Example 14 but longer, shows the power of variable elimination in reducing a messy, but not hard, problem to manageable size.

Example 16. $a''(z_0) < b''(z_0) \wedge \forall u\,(a(u) < b(u) \wedge a'(u) < z_0 < b'(u))$
$\to \exists\, x\ \exists\, y\ \exists\, z\,(a'(y) < z < b'(y) \wedge (a(x) < y < b(x) \wedge$
$a''(z) < x < b''(z)))$

1. $\boxed{a'(y) \not< z} \vee z \not< b'(y) \vee a(x) \not< y \vee y \not< b(x) \vee a''(z) \not< x \vee x \not< b''(z)$

2. $a''(z_0) < b''(z_0)$
3. $a(u) < b(u)$
4. $a'(u) < z_0$
5. $z_0 < b'(u)$.

x, y, z, and u are variables. Notice that no variable is eligible in clause 1.

6. $\boxed{z \not< b'(y)} \vee a(x) \not< y \vee y \not< b(x) \vee z < z_0 \vee a''(z) \not< x \vee x \not< b''(z)$ — 1, 4, y/u, (removing a(y))

7. $a(x) \not< y \vee y \not< b(x) \vee a''(z) \not< x \vee x \not< b''(z) \vee z_0 < z \vee z < z_0$. — 6, 5 (removing b'(y))

Now y is eligible in 7.

8. $\boxed{a(x) \not< b(x)} \vee a''(z) \not< x \vee x \not< b''(z) \vee z_0 < z \vee z < z_0$ — 7, eliminate y

9. $a''(z) \not< x \vee x \not< b''(z) \vee z_0 < z \vee z < z_0$ — 8, 3, x/u (removing b(x))

Now x is eligible in 9.

10. $a''(z) \not< b''(z) \vee z_0 < z \vee z < z_0$ — 9, eliminate x

11. $\square$ — 10, 2.

Example 17. (Sum of two continuous functions is continuous.)

If $\lim_{x \to x_0} f(x) = f(x_0)$ and $\lim_{x \to x_0} g(x) = g(x_0)$,

then

$$\lim_{x \to x_0} [f(x) + g(x)] = f(x_0) + g(x_0) ,$$

or $$\forall \varepsilon (\varepsilon > 0 \to \exists \delta (\delta > 0 \wedge \forall y (|x_0 - y| < \delta \to |f(x_0) - f(y)| \leq \varepsilon)))$$
$$\wedge \forall \varepsilon (\varepsilon > 0 \to \exists \delta (\delta > 0 \wedge \forall y (|x_0 - y| < \delta \to |g(x_0) - g(y)| \leq \varepsilon)))$$
$$\wedge \quad \varepsilon_0 > 0 \to \exists \delta (\delta > 0 \wedge \forall x (|x_0 - x| < \delta \to |(f(x) + g(x)) - (f(x_0) + g(x_0))| \leq \varepsilon_0)).$$

In the following we have used $(-E \leq A \wedge A \leq E)$ instead of $(|A| \leq E)$, in order to avoid the use of the absolute value sign.

1. $f(x_\delta) + g(x_\delta) + \varepsilon_0 < f(x_0) + g(x_0) \vee f(x_0) + g(x_0) + \varepsilon_0 < f(x_\delta) + g(x_\delta) \vee \delta \leq 0$
2. $0 < \varepsilon_0$
3. $0 < \delta_\varepsilon \vee \varepsilon \leq 0$
4. $0 < \delta'_{\varepsilon'} \vee \varepsilon' \leq 0$
5. $f(x_0) \leq f(y) + \varepsilon \vee \delta_\varepsilon + x_0 < y \vee \delta_\varepsilon + y < x_0 \vee \varepsilon \leq 0$
6. $f(y) \leq f(x_0) + \varepsilon \vee \delta_\varepsilon + x_0 < y \vee \delta_\varepsilon + y < x_0 \vee \varepsilon \leq 0$
7. $g(x_0) \leq g(y) + \varepsilon' \vee \delta'_{\varepsilon'} + x_0 < y \vee \delta'_{\varepsilon'} + y < x_0 \vee \varepsilon' \leq 0$
8. $g(y) \leq g(x_0) + \varepsilon' \vee \delta'_{\varepsilon'} + x_0 < y \vee \delta'_{\varepsilon'} + y < x_0 \vee \varepsilon' \leq 0$
9. $x_0 \leq x_\delta + \delta \vee \delta \leq 0$
10. $x_\delta \leq x_0 + \delta \vee \delta \leq 0$

---

11. $g(x_\delta) + \varepsilon_0 < g(x_0) + \varepsilon \vee f(x_0) + g(x_0) + \varepsilon_0 < f(x_\delta) + g(x_\delta) \vee \delta \leq 0$
$\underbrace{\vee \delta_\varepsilon + x_0 < x_\delta \vee \delta_\varepsilon + x_\delta < x_0 \vee \varepsilon \leq 0}_{\alpha_1}$

1, 5 x_δ/y (removing $f(x_\delta)$ from $(f(x_\delta) + g(x_\delta) + \varepsilon_0 < f(x_0) + g(x_0))$)

12. $\varepsilon_0 < \varepsilon + \varepsilon' \vee f(x_0) + g(x_0) + \varepsilon_0 < f(x_\delta) + g(x_\delta) \vee \delta \leq 0 \vee \alpha_1$
$\vee \underbrace{\delta'_{\varepsilon'} + x_0 < x_\delta \vee \delta'_{\varepsilon'} + x_\delta < x_0 \vee \varepsilon' \leq 0}_{\alpha_2}$

11, 7 x_δ/y (removing $g(x_\delta)$ from $(g(x_\delta) + \varepsilon_0 < g(x_0) + \varepsilon)$)

13. $g(x_0) + \varepsilon_0 < g(x_\delta) + \varepsilon \vee \varepsilon_0 < \varepsilon + \varepsilon' \vee \delta \leq 0 \vee \alpha_1 \vee \alpha_2$

12, 6 x_δ/y (removing $f(x_\delta)$ from $(f(x_0) + g(x_0) + \varepsilon_0 < f(x_\delta) + g(x_\delta))$)

14. $\varepsilon_0 < \varepsilon + \varepsilon' \vee \delta \leq 0 \vee \delta_\varepsilon + x_0 < x_\delta \vee \delta_\varepsilon + x_\delta < x_0 \vee \varepsilon \leq 0 \vee \alpha_2$

13, 8 x_δ/y (removing $g(x_\delta)$ from $(g(x_0) + \varepsilon_0 < g(x_\delta) + \varepsilon)$)

15. $\varepsilon_0 < \varepsilon + \varepsilon' \vee \delta \leq 0 \vee \delta_\varepsilon < \delta \vee \delta_\varepsilon + x_\delta < x_0 \vee \varepsilon \leq 0 \vee \alpha_2$

14, 10 (removing x_δ from $(\delta_\varepsilon + x_0 < x_\delta)$)

16. $\varepsilon_0 < \varepsilon + \varepsilon' \vee \delta \leq 0 \vee \delta_\varepsilon < \delta \vee \varepsilon \leq 0 \vee \delta'_{\varepsilon'} < \delta \vee \delta'_{\varepsilon'} + x_\delta < x_0 \vee \varepsilon' \leq 0$

15, 9 (removing x_δ from $(\delta_\varepsilon + x_\delta < x_0)$)

17. $\varepsilon_0 < \varepsilon + \varepsilon' \vee \delta \leq 0 \vee \delta_\varepsilon < \delta \vee \varepsilon \leq 0 \vee \delta'_{\varepsilon'} < \delta \vee \delta'_{\varepsilon'} + x_\delta < x_0 \vee \varepsilon' \leq 0$

16, 10 (removing x_δ from $(\delta'_{\varepsilon'} + x_0 < x_\delta)$)

18. $\varepsilon_0 < \varepsilon + \varepsilon' \vee \delta \leq 0 \vee \delta_\varepsilon < \delta \vee \varepsilon \leq 0 \vee \delta'_{\varepsilon'} < \delta \vee \varepsilon' \leq 0$

17, 9 (removing x_δ from $(\delta'_{\varepsilon'} + x_\delta < x_0)$)

19. $\varepsilon_0 < \varepsilon + \varepsilon' \vee \delta_\varepsilon \leq 0 \vee \delta'_{\varepsilon'} \leq 0 \vee \varepsilon' \leq 0 \vee \varepsilon \leq 0$

18, variable elimination

20. $\varepsilon_0 < \varepsilon + \varepsilon' \vee 0 < 0 \vee \delta'_{\varepsilon'} \leq 0 \vee \varepsilon' \leq 0 \vee \varepsilon \leq 0$

19, 3 (removing δ_ε)

21. $\varepsilon_0 \leq \varepsilon' \vee \delta'_{\varepsilon'} \leq 0 \vee \varepsilon' \leq 0$ — 20, variable elimination

22. $\varepsilon_0 \leq \varepsilon' \vee \varepsilon' \leq 0$ — 21, 4 (removing $\delta'_{\varepsilon'}$)

23. $\varepsilon_0 \leq 0$ — 22, variable elimination

24. □ — 23, TY.

5. Comments

The Special Clause TY

TY is used to collect together all ground inequality literals. (Because of splitting, ground literals can only occur in unit clauses.) One could get the same effect by not using TY at all but instead collecting together all ground inequality literals each time a new ground inequality is produced as a resolvent and checking for a contradiction. We prefer the TY arrangement because it lets us use the sup-inf procedures of [5] to speedily process ground inequalities. A similar speed advantage can be obtained by the use of the ground inequality package of [4].

In any of these methods, a set of ground equality units might be inferred by TY, and these are applied to TY and S by an equality substitution mechanism.

If one did not use splitting, then a method could be devised whereby special TY-literals (a conjunction of ground inequality literals) would occur in clauses.

Chaining

The chaining we employ is, of course, similar to that used by Slagle and Norton [2], except that we do not chain across variables. In fact, we chain only across shielding terms, (see Sections 2.3, 2.4), thus greatly reducing the search space.

Completeness

This system is not complete. However we believe it will become complete is we add paramodulation [11] to handle equality substitution, factoring, and another inference rule whereby the clause $(a = b \vee P \vee Q)\theta$ is inferred from $(a \leq b \vee P)$ and $(b' \leq a' \vee Q)$, if θ is the mgu of $\{b,b'\}$. (This rule is like Slagle and Norton's modification of Rule 3 [2].)

No Added Axioms

Each of the proofs related here was done completely automatically, without human intervention. Also, no additional axioms or lemmas were added or needed, just the statement of the theorem in each case.

Example 17, on the sum of continuous function, was proved automatically several years ago using a special limit heuristic. Also Wos, Overbeek, Lusk and Winker, have proved a simplified version of it with their hyper-resolution program at Argonne Laboratory. But, it appears that this is the first time that a general purpose prover, without special heuristics, has proved this theorem or other inequality theorems of like difficulty.

Limitations

We were surprised by the fact that no non-used clauses (except tautologies which were immediately discarded), were generated in the proof of Example 17, or any of the other proofs in this paper.

However, it would be misleading to claim too much since the family of theorems dealt with includes all of first order logic, it is not surprising that this prover (any prover) will inevitably find difficulty in proving a wide class of theorems. Indeed, many non-used clauses are generated in some recent proofs of other (harder) theorems. So there is much to be done.

Resolution vs Natural Deduction

Resolution is particularly suited for the "variable elimination" method 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 back-

tracking 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.

Acknowledgment

This work was supported by NSF Grant MCS77-20701. We wish to thank Mike Ballantyne and Peter Bruell for their ideas and help.

References

1. W.W. Bledsoe. A Resolution-based Prover for General Inequalities. University of Texas, Math Department Memo ATP-52, July 1979.

2. J.R. Slagle and L. Norton. Experiments with an Automatic Theorem Prover Having Partial Ordering Rules. CACM 16(1973), 683-688.

3. J.C. King. A Program Verifier. Ph.D. Thesis, Carnegie-Mellon University, 1969.

4. Greg Nelson and Derek Oppen. A Simplifier Based on Efficient Decision Algorithms. Proc. 5th ACM Symp. on Principles of Programming Languages, 1978.

5. Robert Shostak. A Practical Decision Procedure for Arithmetic with Function Symbols. JACM, April 1979.

6. W.W. Bledsoe, Peter Bruell and Robert Shostak. A Prover for General Inequalities. University of Texas, Math Department Memo ATP-40A, February 1979. Also IJCAI-79, Tokyo, Japan, August 1979.

7. Louis Hodes. Solving Programs by Formula Manipulation in Logic and Linear Inequality. Proc. IJCAI-71, London, 1971, pages 553-559.

8. M.A. Stickel. A Complete Unification for Associative-Commutative Functions, IJCAI-75, Tbilisi, USSR, 1975, pages 71-76.

9. J.A. Robinson. A Machine-oriented Logic Based on the Resolution Principle. JACM 12(1965), 23-41.

10. W.W. Bledsoe. Splitting and Reduction Heuristics in Automatic Theorem Proving. AEJ 2(1971), 55-77.

11. L. Wos and G.A. Robinson. Paramodulation and Set of Support. Proc. Symp. Automatic Demonstration, Versailles, France. Springer-Verlag, New York, 1968, 276-310.