Ex. 12. \((\forall y (y \leq \ell \rightarrow \exists z(y \leq z \leq b)) \land a \leq \ell \rightarrow a \leq b)\).

1. \(z \leq b \lor \ell \leq y\)
2. \(y < z \lor \ell \leq y\)
3. \(a \leq \ell\)
4. \(b < a\) (top clause)

TY: \([a \leq \ell, b < a]\)

5. \(y < z \lor b < y\)  \hspace{1cm} 4, 3, 2, CHAIN\(^+\)
6. \(y < b \lor \ell \leq y \lor b < y\)  \hspace{1cm} 5, 1, CHAIN\(^+\)
7. \(\ell \leq b \lor b < b\)  \hspace{1cm} 6, eliminate \(y\)
8. \(\ell \leq b\)  \hspace{1cm} 7, REDUCE
9. \(\square\)  \hspace{1cm} 8, TY

\(^+\)Note that this step falls under the RL-CASE of CHAIN (see RL-CASE, Section 3).
Ex. 13.  \((\forall \varepsilon \ (0 < \varepsilon \rightarrow A \leq B + \varepsilon) \rightarrow A \leq B)\)

1.  \(A \leq B + \varepsilon \vee \varepsilon \leq 0\)  \((\varepsilon \text{ is a variable})\)

2.  \(B < A\)  \((\text{top clause})\)

TY:  \([B < A]\)

3.  \(0 \leq B - A\)  \(1, \text{eliminate variable } \varepsilon\)

4.  \(\square\)  \(3, \text{TY}\)

In this example, the literal \(A \leq B + \varepsilon\) is rewritten as \(A - B \leq \varepsilon\) so that \(\varepsilon\) can be seen as eligible in clause 1, and then eliminated, leaving \(0 \leq B - A\).

Ex. 14.  \((\forall \varepsilon \ (\varepsilon > 0 \rightarrow A \leq B + \varepsilon) \land B \leq C \rightarrow A \leq C)\)

1.  \(A \leq B + \varepsilon \vee \varepsilon \leq 0\)

2.  \(B \leq C\)

3.  \(C < A\)

TY:  \([B \leq C, C < A]\)

4.  \(0 \leq B - A\)  \(1, \text{eliminate } \varepsilon\)

5.  \(\square\)  \(4, \text{TY}\)
Ex. 15. \( \forall \varepsilon \left( (\varepsilon > 0 \rightarrow A \leq B + \varepsilon) \land B \leq C \rightarrow A \leq C \right) \)

1. \( A \leq B + \varepsilon \lor \varepsilon \leq 0 \)
2. \( B \leq C \)
3. \( C < A \) (top clause)

TY: \([C < A]\)

4. \( A \leq C + \varepsilon \lor \varepsilon \leq 0 \) 3, 1, 2, CHAIN

(See Section 2, RL-CASE, Case \( \text{DIR} = T, \ v \neq 0 \))

5. \( 0 \leq C - A \) 4, eliminate \( \varepsilon \)
6. \( \square \) 5, TY

Ex. 16. \( \exists \varepsilon \left( (\varepsilon > 0 \rightarrow A \leq B + \varepsilon) \land B \leq C \rightarrow A \leq C \right) \) False.

1. \( A \leq B + \varepsilon \lor \varepsilon \leq 0 \)
2. \( B \leq C \)
3. \( C < A \)
4. \( A \leq C + \varepsilon \) 3, 1, 2, CHAIN

But now we cannot eliminate \( \varepsilon \) as we did in Ex. 15.

Ex. 17. \( [\forall \varepsilon (\varepsilon > 0 \rightarrow f(\ell) \leq f(z) + \varepsilon \land f(z) \leq f(t_0) \rightarrow f(\ell) \leq f(t_0))] \)

1. \( f(\ell) \leq f(z) + \varepsilon \lor \varepsilon \leq 0 \)
2. \( f(z) \leq f(t_0) \lor \varepsilon \leq 0 \)
3. \( f(t_0) < f(\ell) \) (top clause)

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

4. \( f(\ell) \leq f(t_0) + \varepsilon \lor \varepsilon \leq 0 \) 3, 1, 2, CHAIN
5. \( 0 \leq f(t_0) - f(\ell) \) 4, Eliminate \( \varepsilon \)
6. \( \square \) 5, TY
Ex. 18. \( \forall z (P(z,c_1) \lor P(z,c_2)) \rightarrow \exists x \exists y (P(x,y) \land a(y) < x) \).

1. \( P(z,c_1) \lor P(z,c_2) \)

2. \( \sim P(x,y) \lor a(y) \notin x \)  
   (top clause)

3. \( P(z,c_2) \lor a(z) \notin x \)  
   1, 2, \( z/x, c_1/y \)

4. \( a(c_1) \notin z \lor a(c_2) \notin z \)  
   3, 2, \( z/x, c_2/y \)

5. \( \square \)  
   4, eliminate \( z \)

When a new resolvent is computed it is processed and used as the new top clause. It is simplified by REDUCE and duplicate literals are merged, eligible literals are eliminated, and the clause is split if possible. If the result is a ground unit clause, it is resolved with TY and checked for contradiction. Finally, it is sorted, with RL-literals put last, before being used as the top clause in a new call to RESOLVE-CHAIN (see PROCESS-RESOLVENT, Section 3).

In Ex. 19 and later examples, we indicate the literal being resolved upon by outlining it with a rectangle, \( \text{[L]} \). This is usually the first literal of the top clause after it has been processed and sorted.
Ex. 19. \[a \leq b \land a \leq t_0 < \ell\]
\[\land \forall \epsilon(\epsilon > 0 \rightarrow \exists r(r < \ell \land \forall s(s \leq \ell \rightarrow f(\ell) \leq f(s) + \epsilon)))\]
\[\land \forall y(a \leq y \leq \ell \rightarrow \exists z(y < z \leq \ell \land \forall t(a \leq t \leq z \rightarrow f(z) \leq f(t))))\]
\[\rightarrow f(\ell) \leq f(t_0)\]

1. \(a \leq b\)
2. \(b \leq b\)
3. \(a \leq t_0\)
4. \(t_0 < \ell\)
5. \(r_\epsilon < \ell\)
6. \(\forall \epsilon \leq 0\)
7. \(\forall y < z, y < a \land y \leq y\)
8. \(\forall y < a \land y < y\)
9. \(f(y) \leq f(t)\)
10. \(f(\ell) \leq f(t_0)\) (top clause)

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

11. \(f(\ell) \leq f(t_0) + \epsilon \land \epsilon \leq 0 \land z, y < r_\epsilon \land \ell < z, y\)

\[\land \forall y < a \land y \land t_0 < a \lor y \land z, y \leq t_0\]

Clause 11 splits on \((t_0 < a)\) into clauses 12.1 and 12.2.

12.1 \(t_0 < a\)

13. \(\square\)

12.2 \(f(\ell) \leq f(t_0) + \epsilon \land \epsilon \leq 0 \lor z, y < r_\epsilon \land \ell, y \land z, y \leq t_0\)

12.1, TY
13. \( f(\ell) \leq f(t_0) + \varepsilon \land \varepsilon \leq 0 \land \forall y < \varepsilon \land \ell \leq z_y \land \\
\lor y < a \land \ell \leq y \land z_y \leq t_0 \)

14. \( f(\ell) \leq f(t_0) + \varepsilon \land \varepsilon \leq 0 \land \forall y < \varepsilon \land \ell \leq z_y \land \\
\lor y < a \land \ell \leq y \land y < t_0 \)

15. \( f(\ell) \leq f(t_0) + \varepsilon \land \varepsilon \leq 0 \land \forall y < a \land \\
\lor y < a \land \ell \leq y \land y < t_0 \)

16. \( f(\ell) \leq f(t_0) + \varepsilon \land \varepsilon \leq 0 \land \\
\lor \ell \leq \varepsilon \land \ell \leq a \land \ell \leq t_0 \)

Clause 16 splits on \((\ell \leq a)\) and \((\ell \leq t_0)\) into clauses 17.1, 17.2, and 17.3

17.1 \( \ell \leq a \)

18. \( \Box \)

17.2 \( \ell \leq t_0 \)

19. \( \Box \)

17.3 \( f(\ell) \leq f(t_0) + \varepsilon \land \varepsilon \leq 0 \land \ell \leq \varepsilon \)

20. \( f(\ell) \leq f(t_0) + \varepsilon \land \varepsilon \leq 0 \)

21. \( f(\ell) \leq f(t_0) \)

22. \( \Box \)
**Ex. 20.** \( \exists a \exists b \forall x \exists u \left( (x < b \rightarrow u \leq a) \land x \leq u \right) \land (u \leq a \lor x \neq b) \)

1. \( x_0 < b \lor x_0 \leq u \land u \leq a \)
2. \( x_0 < b \lor x_0 \leq u \lor u \leq x_0 = b \)
3. \( u \leq a \lor x_0 \leq u \lor u \leq a \)
4. \( u \leq a \lor x_0 \leq u \lor x_0 = b \)

\( a, b, u \) are variables, \( x_0 \) stands for the expression \( x_{ab} \)

5. \( x_0 < b \lor x_0 \leq a \) \hspace{2cm} 1, eliminate \( u \)
6. \( x_0 < b \lor x_0 = b \) \hspace{2cm} 2, eliminate \( u \)
7. \( x_0 \leq a \) \hspace{2cm} 3, eliminate \( u \)
8. \( x_0 \leq a \lor x_0 = b \) \hspace{2cm} 4, eliminate \( u \)

(subsumed by 7)

9. \( b \leq a \lor x_0 < b \) \hspace{2cm} 7, 6, sub =
10. \( b \leq a \) \hspace{2cm} 9, 7, CHAIN
11. \( \Box \) \hspace{2cm} 10, eliminate \( a \) (or \( b \)).

Examples 20 and 21 arise in the search for counterexamples in a proof.
Ex. 21.  \[ \exists a \exists b \exists \ell \exists w \forall s \exists u \]
\[(a \leq b \land [u \leq 0 \lor s \neq a] \land [0 \leq u \lor s \neq b] \land [s \leq w \lor 0 < u] \land w \leq \ell) \]
\[ \ldots \]
\[ a, b, \ell, \text{ and } w \text{ are variables, and } s_0 \text{ stands for } s_{ab\ell w}. \]

1. \[ b < a \lor 0 < u \lor u < 0 \lor w < s_0 \lor \ell < w \]
2. \[ " \quad \quad \quad \quad \quad \lor u \leq 0 " \]
3. \[ " \quad \quad \quad \quad \quad s_0 = b \lor w < s_0 " \]
4. \[ " \quad \quad \quad \quad \quad \lor u \leq 0 " \]
5. \[ " \quad \quad \quad \quad \quad s_0 = a \lor u < 0 \lor w < s_0 " \]
6. \[ " \quad \quad \quad \quad \quad \lor u \leq 0 " \]
7. \[ " \quad \quad \quad \quad \quad s_0 = b \lor w < s_0 " \]
8. \[ " \quad \quad \quad \quad \quad \lor u \leq 0 " \]

Preprocessing removes clauses 2 and 4 which are tautologies, and eliminates the variable \( u \) from the other clauses.

1. \[ b < a \lor 0 < 0 \quad \lor w < s_0 \lor \ell < w \]
3. \[ " \quad \quad \quad \quad \quad s_0 = b \lor w < s_0 " \]
5. \[ " \quad \quad \quad \quad \quad \lor s_0 = a \quad \lor w < w_0 " \]
6. \[ " \quad \quad \quad \quad \quad \quad " \]
7. \[ " \quad \quad \quad \quad \quad s_0 = b \lor w < s_0 " \]
8. \[ " \quad \quad \quad \quad \quad s_0 = b " \]

The literal \((0 < 0)\) is removed from clause 1 by REDUCE. Then clause 3 is subsumed by clause 1, and clauses 5, 7, and 8 are subsumed by clause 6.

Thus preprocessing reduces these eight clauses to two.

1. \[ w < s_0 \lor b < a \lor \ell < w \] \hspace{1cm} (top clause)
6. \[ s_0 = a \lor b < a \lor \ell < w \]
9. $w < a \lor b < a \lor \ell < w \quad 1, \text{ sub }$
10. $\ell < w \quad 9, \text{ eliminate } a$
11. $\Box \quad 10, \text{ eliminate } \ell$

This example shows the power of variable elimination in reducing a messy, but not hard, problem to manageable size.

**Ex. 22.** $(\forall x(f(x) \geq 0) \wedge \forall s(f(s) \geq 0 \land \ell \leq s \rightarrow t \leq s) \rightarrow t \leq \ell)$

1. $f(x) \geq 0 \lor x \leq \ell$ (top clause)
2. $f(s) < 0 \lor s \leq \ell \lor t_0 \leq s$
3. $\ell < t_0$

TY: $[\ell < t_0]$

4. $s < \ell \lor t_0 \leq s \lor s \leq \ell \quad 1, 2, \text{ s/x}$
5. $t_0 \leq \ell \quad 4, \text{ eliminate s}$
6. $\Box \quad 5, \text{ TY}$

Notice that in this example clause 3 cannot serve as top clause, although one would naturally choose it because it represents the conclusion of the theorem.
Ex. 23. \( f(\ell) < 0 \land f(b) \geq 0 \land \forall s (0 \leq f(s) \land \ell \leq s \rightarrow t \leq s) \land \forall x (x \leq b \land f(x) \leq 0 \rightarrow x \leq \ell) \land \forall y (\forall z (z \leq b \land f(z) \leq 0 \rightarrow z \leq y) \rightarrow \ell \leq y) \rightarrow t \leq \ell \)

1. \( f(\ell) < 0 \)
2. \( 0 \leq f(b) \)
3. \( f(s) < 0 \lor s < \ell \lor t_0 \leq s \) (top clause)
4. \( 0 < f(x) \lor b < x \lor x \leq \ell \)
5. \( z_y \leq b \lor \ell \leq y \)
6. \( f(z_y) \leq 0 \lor \ell \leq y \)
7. \( y < z_y \lor \ell \leq y \)
8. \( \ell < t_0 \)

TY: \([f(\ell) < 0, 0 \leq f(b), \ell < t_0] \)

Note: Clause 8 will not succeed as top clause.

9. \( s < \ell \lor t_0 \leq s \lor b < s \lor s \leq \ell \) 3, 4, s/x
10. \( b \leq \ell \lor t_0 \leq \ell \) 9, eliminate s

SPLIT 10 into 10.1 and 10.2

10.1 \( b \leq \ell \)

TY: \([f(\ell) < 0, 0 \leq f(b), \ell < t_0, b \leq \ell] \)

11. \( \ell < y \lor \ell \leq y \) 10.1, 5, 7, CHAIN
12. \( \square \) 11, eliminate y

10.2 \( t_0 \leq \ell \)
13. \( \square \) 10.2, TY
Ex. 24. \[ a''(z_0) < b''(z_0) \land \forall u (a(u) < b(u) \land a'(u) < z_0 < b'(u)) \]

\[ \rightarrow \exists x \exists y \exists z \ (a'(y) < z < b'(x)) \land (a(x) < y < b(x) \land a''(z) < x < b''(z)) \]

1. \[ a'(y) \not< z \lor z \not< b'(y) \lor a(x) \not< y \lor y \not< b(x) \]
\[ \lor a''(z) \not< x \lor x \not< b''(z) \]

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, and clause 1 is the top clause. Notice that no variable is eligible in clause 1.

6. \[ z_0 \not< b'(y) \lor a(x) \not< y \lor y \not< b(x) \]
\[ \lor a''(z_0) \not< x \lor x \not< b''(z_0) \]

7. \[ a(x) \not< y \lor y \not< b(x) \lor a''(z_0) \not< x \lor x \not< b''(z_0) \]

Now \( y \) is eligible in 7.

8. \[ a(x) \not< b(x) \lor a''(z_0) \not< x \lor x \not< b''(z_0) \]
7, eliminate \( y \)

9. \[ a''(z_0) \not< x \lor x \not< b''(z_0) \]
8, 3, \( x/u \)

Now \( x \) is eligible in 9.

10. \[ a''(z_0) \not< b''(z_0) \]
9, eliminate \( x \)

11. \[ \square \]
10, 2.
5. Computer Implementation and Results

These ideas were tested by a program written in LISP and run on the DEC 10 computer at The University of Texas at Austin. In that program we varied somewhat the algorithms described in Section 3. For example, in step 2 of the algorithm RESOLUTION-CHAIN, we allowed $L$ to be only the first literal of the top clause $C$. This "first literal only" strategy is incomplete as can be seen by rearranging the literals of clause 1 of Ex. 24. However, we feel that examples like Ex. 24 are contrived and unlikely in "normal applications." Much work remains to be done to refine these procedures.

Ex. 1 - Ex. 24 were all proved by this program. We would be interested to know whether other automatic provers are able to prove these examples, especially Ex. 19, 21, 23, 24, and also Ex. 5, 6, 7 of [6].
6. **Completeness**

Completeness is not claimed for this system. For example, the ability to handle algebraic expressions must be further enhanced if it is to handle complicated polynomial terms.

A series of completeness results might be proved to show that the basic mechanisms of this system are correct (if indeed they are).

For example, the variable elimination rule, Section 2.2, requires that the variable being eliminated be "eligible." Then an interpolation axiom is used (in effect) to eliminate the variable. The underlying philosophy is that one can delay the use of the interpolation axiom until after the variable becomes eligible. To make this more precise, consider the clauses

1. \( P(x) \lor a \neq x \lor x \neq b \),
2. \( \neg P(x) \),
3. \( a < b \)

and the interpolation axioms

\[ CI1 \quad x \neq y \lor x < w(x, y) \]
\[ CI2 \quad x \neq y \lor w(x, y) < y. \]

We can proceed to \( \square \) in two different ways: by using CI1, CI2 first or last.

4. \( P(w(a, b)) \lor a \neq b \) 1, CI1, CI2
5. \( a \neq b \) 4, 2
6. \( \square \) 5, 3

or

4. \( a \neq x \lor x \neq b \) 1, 2
5. \( a \neq b \) 4, CI1, CI2
6. \( \square \) 5, 3
Our contention is that we can always delay using CI1, CI2 until last, as we did here, or at least delay until x becomes eligible. The statement of this contention is given (but not proved) as Theorem 1.

Theorem 1. Suppose that S is an unsatisfiable set of clauses which contains the interpolation clauses CI1 and CI2 and which does not otherwise contain the symbol w. Then there is a deduction of □ from S in which the function symbol "w" does not occur in a resolvent, except in cases where it is introduced in one step by resolving on one of CI1 or CI2 and eliminated in the next step by resolving on the other.

Of course, if Clauses 2 and 3 were replaced by

2. \( \sim P(c) \),
3. \( a < c \),
4. \( c < b \),

then CI1, CI2 could not be (profitably) used at all and then delaying is even more desirable.

Another argument (perhaps the main argument) for delaying the use of the interpolation axioms is that a premature use of CI1, CI2 might block their further use when more than one use is required. Consider the following example.

1. \( P(x) \lor a \neq x \lor x \neq b \)
2. \( \sim P(s) \lor c \neq s \)
3. \( a < b \)
4. \( c < b \)

Since x is not eligible in Clause 1, we prefer to first resolve it, upon \( P(x) \), with Clause 2, getting

5. \( c \neq x \lor a \neq x \lor x \neq b \)
and then eliminate it

6. $c \not\in b \lor a \not\in b$  \hspace{1cm} 5
7. \hspace{1cm} 6, 3, 4

If we had tried to use CI1, CI2 first we would get

5. $P(w(a, b)) \lor a \not\in b$  \hspace{1cm} 1, CI1, CI2
6. $P(w(a, b))$  \hspace{1cm} 5, 3
7. $c \not\in w(a, b)$  \hspace{1cm} 6, 2

But this is blocked.

As a general principle, if we delay using the interpolation axioms until the variable being interpolated becomes eligible, then we collect together all the requirements on $x$, and eliminate them all at once. If in this process $x$ is instantiated by a non-variable term, then the interpolation axiom could not have been used anyway.
7. Comments

The Special Clause $TY$

$TY$ is used simply 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 [11] to speedily process ground inequalities. A similar speed advantage can be obtained by the use of the ground inequality packages of [4] and [5].

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

Inequality chaining (see CHAINER, Section 3) is required for many of the examples given in Section 4 (unless one is to resort to using the transitivity axioms again), but it tends to enlarge the search space when it is used. We envision a system that uses chaining but in a sparing way under the control of heuristics.

The chaining we employ is, of course, similar to that used by Slagle and Norton [2, 12], except that we do not chain across variables (see p. 17), and also we do not retain the intermediate links in the chain as they do. For instance, in the example
\[ f(\delta) \subseteq f(z_y) \lor D(y) \]
\[ f(z_a) \leq L \]
\[ L < f(\delta) \lor E \]

We do not retain the (intermediate) resolvents
\[ f(\delta) \leq L \lor D(a) \]

and
\[ f(z_a) < f(\delta) \lor E \]

but only
\[ D(a) \lor E. \]

In this way it is somewhat like hyper-resolution.

**Fast Implementation**

Our intention here was not to present the best and fastest Resolution system but rather to present a few concepts that could make an existing Resolution prover faster for inequality proofs. We have not used here but would highly recommend such things as

- a pointer system which lets relevant clauses be accessed quickly [13],
- mechanisms that resolve more than one clause at once, and avoid the (explicit) use of substitutions [13, 14, 22],
- fast unifiers [15],
- interconnectivity graphs [16-20], which precompute various unification chains.

Some of these concepts are embodied in the provers of Overbeek-Wos-Lusk-Winker. Of course, other powerful resolution procedures such as Model Elimination [22] and SL-Resolution [23] might be used.
Hyper-resolution and Counterexamples

Unfortunately, ordinary hyper-resolution is not compatible with our chaining and variable-elimination techniques. For instance, the clauses
1. \( z_y \leq t_0 \)
2. \( y \leq z_y \)

resolve to \( \square \) as follows

3. \( y \leq t_0 \) \hspace{1cm} 1, 2
4. \( \square \) \hspace{1cm} 3, eliminate \( y \)

and yet there are no negative clauses at all. (Of course, if we included the interpolation axioms we would have at least one negative clause.)

A similar problem arises in trying to use models (counterexamples)) because that is just an extended form of hyper-resolution [21]. We expect to overcome this problem to some extent in a later paper.

Acknowledgement

This work was supported by NSF Grant MCS 77-20701. Thanks to Mike Ballantyne for his ideas and help.
References


2. J. R. Slagle. Automatic Theorem Proving with Built-in Theories Including 

   1969.

4. Greg Nelson and Derek Oppen. A Simplifier Based on Efficient Decision 
   (Also: Simplification by Cooperating Decision Procedures, Stanford A. I. 
   Memo.)

5. Robert Shostak. A Practical Decision Procedure for Arithmetic with Function 

   Inequalities. The Univ. of Texas Math. Dept. Memo ATP 40A, Feb. 1979. Also 

7. Louis Hodes. Solving Programs by Formula Manipulation in Logic and Linear 

   Proving. Machine Intelligence 9 (eds. J. E. Hayes, D. Michie, L. I. Mikhlich), 

   Automatic Demonstration, Versailles, France. Springer-Verlag, New York (1968), 
   276-310.


11. W. W. Bledsoe and Mabry Tyson. Typing and Proof by Cases in Program Verif-
    ication. Machine Intelligence 8 (eds. D. Michie and E. W. Elcock), Chichester: 
    Ellis Horwood Limited. 1977, 30-51.


    (1975), 201-214.

14. R. S. Boyer and J S. Moore. The Sharing of Structure in Theorem-proving Pro- 
    grams. Machine Intelligence 7 (eds. B. Meltzer and D. Michie), Edinburgh 
15. J. A. Robinson.


