01 Jan 1992

Experimentation with Proof Methods for Non-Horn Sets

Christopher J. Merz
Missouri University of Science and Technology, merzc@mst.edu

Ralph W. Wilkerson
Missouri University of Science and Technology, ralphw@mst.edu

Recommended Citation

C. J. Merz and R. W. Wilkerson, "Experimentation with Proof Methods for Non-Horn Sets," *Applied Computing: Technological Challenges of the 1990's*, pp. 530 - 535, Association for Computing Machinery, Jan 1992.
The definitive version is available at https://doi.org/10.1145/143559.143683

Experimentation with Proof Methods for non-Horn Sets

Chris Merz and Ralph Wilkerson
Department of Computer Science
University of Missouri-Rolla

Abstract

Two resolution proof strategies developed by Peterson [4] are implemented by modifying Otter, an existing automated theorem prover. The methods, Lock-T refutation and LNL-T refutation, are generalizations of unit refutation and input resolution, respectively, to non-Horn sets and represent independent, equivalent but opposite ways of searching. The algorithms used are based on a corrected version of the foundational work. The strategies have been tested on various non-Horn challenge problems from the Tarskian Geometry and the Non-Obvious problem, with the results being in some cases quite favorable when compared to other resolution techniques.

Introduction

In its purest form resolution is a brute force, semi-decidable, and combinatorially explosive approach to automated theorem proving. Numerous strategies have been devised to help focus this process, such as the set of support strategy, input resolution, unit resolution, linear resolution, binary resolution, hyperresolution, unit-resolvent resolution, and lock resolution. It is well known that some of these procedures are only complete for Horn sets, and since many problems cannot be formulated as Horn sets, an efficient method for directing the resolution process on non-Horn sets is needed.

Peterson [4] generalized input resolution and unit resolution to apply to non-Horn sets by integrating them with a specialized lock resolution procedure. The Lock-T proof method generalizes unit resolution and is a breadth-first search for the empty clause, while the LNL-T proof method generalizes input resolution and is a depth-first search for the empty clause. Peterson postulated that these two strategies would work well together if run in parallel and allowed to share "important" intermediate results known as lemmas. The implementation of these methods was done by modifying the automated theorem prover, Otter, developed at Argonne National Laboratories by McCune [2]. This permitted the use of existing routines which handled the internal representation, manipulation, and bookkeeping of the clauses. Test problems from the Tarskian Geometry [5] and the Non-Obvious problem [3] were used to evaluate algorithm performance.

As stated in the introduction, this work is based on the modification and integration of set of support resolution, input resolution, unit resolution, and lock resolution. Resolution procedures take as their argument a set of expressions in a simplified version of predicate calculus known as clausal form. The symbols, terms, and atomic sentences of clausal form are the same as those in ordinary predicate calculus. However, all logical implications and quantifiers are removed leaving only sets of disjuncted positive or negative literals called *clauses*. For example, the predicate calculus sentence $A \Rightarrow B$ which is equivalent to $-A \vee B$ has the clausal form $\{-A, B\}$.

Given S, an unsatisfiable set of clauses, a *set of support* for S is a set of clauses which if removed from S results in a satisfiable subset of clauses. A *set of support resolvent* is one in which at least one parent is selected from the set of support or a descendant thereof. A *set of support deduction* is one in which each derived clause is a set of support resolvent. A *set of support refutation* is a set of support deduction of the empty clause .

In input resolution, an *input resolvent* is one in which at least one of the two parent clauses is a member of the initial database. An *input deduction* is one in which all derived clauses are input resolvents, and an *input refutation* is an input deduction of the empty clause . In unit resolution, a *unit resolvent* is one in which at least one of the parent clauses is a unit clause; i.e., one containing a single literal. A *unit deduction* is one in which all derived clauses are unit resolvents, and a *unit refutation* is a unit deduction of the empty clause .

Lock resolution is described in detail in [1]. A literal is said to be *locked* if it has been assigned a positive integer which is then called its *lock*. A clause is locked if each of its literals are locked, and a set of clauses is locked if each clause therein is locked. A *lock resolution* is a resolution of locked clauses in which the literals resolved upon have the minimum lock in their respective clauses and the literals of the resolvent inherit the locks they had in the parents. For example, let $C = \{A_1(x), -B_2(x,y), C_2(y)\}$ and $D = \{-A_3(a), B_4(c,x)\}$, where the subscripts represent locks. The only possible lock resolvent of C and D would be $\{-B_2(a,y), C_2(y), B_4(c,x)\}$. A *lock proof* is one in which all resolvents are lock resolvents. A *lock refutation* is a lock proof of the empty clause, {}.

A *Horn set* is a set of Horn clauses, where a *Horn clause* is a clause with at most one positive literal. Each of the resolution techniques defined above is complete only for Horn sets. That is, for an unsatisfiable set S, a refutation is guaranteed only if S is a Horn set.

Extending input and unit resolution

This section presents a summary and correction to Peterson's [4] foundational work on this subject. The first step is to generalize unit and input resolution to apply to Horn sets. The following theorem can be thought of as a generalization of unit resolution and states that lock resolution (without any restrictions on the locking of literals) is complete for Horn sets.

Theorem 1 *If S is an unsatisfiable set of locked Horn clauses, then there exists a refutation of S by lock resolution alone (no factoring).*

Just as Theorem 1 is a generalization of unit resolution, Theorem 2 is a generalization of input resolution.

Theorem 2 *Suppose S is an unsatisfiable set of locked Horn clauses in which the lock of any positive literal is the minimum lock in its clause. Then there exists a linear input refutation of S with top clause negative by lock resolution alone.*

Recall that a *linear refutation with top clause C* is a refutation in which one parent of each resolvent, the center parent, is the result of the previous resolution. One parent of the first resolvent is C. Since it is desirable to use these strategies without restricting the clause types, it is necessary to extend Theorems 1 and 2 to non-Horn sets. To build up to this, a few more terms must be introduced.

In a set S of clauses, a single literal may occur in several different places (i.e., in the same clause or in other clauses); each of these will be called an *occurrence* of that literal. In proofs by resolution alone, each literal L may be traced back through its parent clauses to a single occurrence of a literal called the *ancestor* of L. If T is a set of occurrences of literals in S, a *T-literal* is any literal whose ancestor is in T and a *T-clause* is any clause composed entirely of T-literals.

A non-Horn set S can also be thought of as a $Horn_T$ set where T is a set of literal occurrences chosen from the nonunit clauses of S such that the deletion from S of all the literal occurrences in T changes S into a Horn set. Any literal in T will be called a T-literal, and any clause composed entirely of T-literals is a T-clause or a *T-lemma*. Using Peterson's terminology, a *lock negative linear T-lemma (LNL-T) proof of a clause C* is a lock proof of C in which the only lemmas are T-clauses, and each lemma and C are proved linearly. In such proofs, the side clauses are either members of S or earlier proved lemmas, and the center clauses are composed only of negative literals and T-literals. The top clause in an LNL-T proof must be negative, and factoring is performed only on lemmas and only on literals with the same ancestor and with the lowest lock in their clause. A *lock-T proof of a clause C* is a lock proof of C in which factoring only occurs in T-clauses and only on literals of smallest lock and of common ancestery. We now show how to change input resolution, which is know to be complete for Horn sets, into a method which is complete for $Horn_T$ sets. The method is stated in the following theorem and then illustrated with an example.

Theorem 3 *Suppose S is an unsatisfiable HornT set which is locked in such a way that the literals of T in any clause are locked greater than or equal to the other literals of that clause, and that if a clause contains a positive literal which is not in T then that literal is locked less than or equal to the other literals of that clause. Then S has an LNL-T refutation.*

For example, consider the set of clauses which show that any number greater than 1 has a prime divisor [1]. The clauses are listed as sets of literals disjuncted by the "|" symbol where the subscripts are locks and the T-literals are also subscripted with a T. Factoring will only take place under the conditions laid out for LNL-T proofs. The set S is:

1 $\{D_1(x,x)\}$
2 $\{-D_2(x,y)| - D_3(y,z)|D_1(x,z)\}$
3 $\{P_1(x)|D_{T2}(g(x),x)\}$
4 $\{P_1(x)|L_{T2}(l,g(x))\}$
5 $\{P_1(x)|L_{T2}(g(x),x)\}$

6 $\{L_1(l,a)\}$
7 $\{-P_3(x)| - D_2(x,a)\}$
8 $\{-L_5(l,x)| - L_4(x,a)|P_1(f(x))\}$
9 $\{-L_7(l,x)| - L_6(x,a)|D_1(f(x),x)\}$

It can be seen that S is a Horn$_T$ set by removing the T-literals of clauses 3, 4, and 5. One possible LNL-T refutation might proceed as follows (with parent clauses in brackets). Note the derivation of each of the lemmas (clauses 11, 14, and 15) and the LNL-T proof of the empty clause.

10	[7 1]	$\{-P_3(a)\}$
11	[10 3]	$\{D_{T2}(g(a),a)\}$
12	[7 1]	$\{-P_3(a)\}$
13	[12 4]	$\{L_{T2}(l,g(a))\}$
14	[7 1]	$\{-P_3(a)\}$
15	[14 5]	$\{L_{T2}(g(a),a)\}$
16	[7 2]	$\{-D_2(x,y)| - D_3(y,a)| - P_3(x)\}$
17	[16 9]	$\{-D_3(y,a)| - P_3(f(y))| - L_7(l,y)| -L_6(y,a)\}$
18	[17 11]	$\{-P_3(f(g(a)))| - L_7(l,g(a))| -L_6(g(a),a)\}$
19	[18 8]	$\{-L_5(l,g(a))| - L_4(g(a),a)| -L_7(l,g(a))| - L_6(g(a),a)\}$
20	[19 15]	$\{-L_5(l,g(a))| - L_7(l,g(a))| -L_6(g(a),a)\}$
21	[20 13]	$\{-L_7(l,g(a))| - L_6(g(a),a)\}$
22	[21 15]	$\{-L_7(l,g(a))\}$
23	[22 13]	$\{\}$

Theorem 4 *If S is an unsatisfiable Horn$_T$ set which is locked in such a way that the literals of T in any clause are locked greater than or equal to the other literals of that clause, then there exists a lock-T refutation of S.*

The main point of the last two theorems is that for a Horn$_T$ set there is a refutation in which the only lemmas are T-clauses. Unfortunately, the preconditions of Theorems 3 and 4 are too weak to guarantee the existence of their respective refutation types as demonstrated by the following counterexample to Theorem 4.

Let S be the non-Horn set

1 $\{A|B\}$
2 $\{-A|B\}$
3 $\{A| - B\}$
4 $\{-A| - B\}$

The set S is unsatisfiable by the following resolution proof (where factoring is allowed),

5	[1,2]	$\{B\}$
6	[3,4]	$\{-B\}$
7	[5,6]	$\{\}$

Hence, S is an unsatisfiable Horn$_T$ set by the following locking scheme (where the locks are the subscripts and the T-literals are subscripted with a "T" and a lock):

1 $\{A_1|B_{T2}\}$
2 $\{-A_1|B_{T3}\}$
3 $\{A_1| - B_3\}$
4 $\{-A_1| - B_3\}$

By noting that the literals of T in any clause are locked greater than or equal to the other literals of that clause, we have satisfied all of the preconditions of Theorem 4 which implies a lock-T refutation of S exists. But the complete list of lock-T resolvents is listed below, and it does not contain the empty clause

5	[1,2]	$\{B_{T2}\|B_3\}$
6	[1,4]	$\{B_{T2}\| - B_3\}$
7	[2,3]	$\{B_{T3}\| - B_3\}$
8	[3,4]	$\{-B_3\| - B_3\}$
9	[5,7]	$\{B_{T3}\|B_3\}$

The problem here is in clauses 5 and 6, where the T-literal, B_{T2}, blocks the non-T-literals locked higher than it from being resolved and prevents the proof from continuing. Note also that in a Lock-T proof of a clause C factoring only occurs in T-clauses and only on the lowest locked literals of common ancestry. This is why clauses 8 and 9 cannot be factored and resolved with each other to obtain the empty clause.

Now note that by strengthening the locking requirements on T-literals by requiring them to be locked greater than or equal to **ALL** other literals in S, a lock-T refutation can be found for the example above. The new locking of S is

1 $\{A_1|B_{T4}\}$
2 $\{-A_1|B_{T3}\}$
3 $\{A_1| - B_3\}$
4 $\{-A_1| - B_3\}$

The refutation proof is

5	[1,2]	$\{B_{T3}\|B_{T4}\}$
6	[1,4]	$\{-B_3\|B_{T4}\}$
7	[2,3]	$\{B_{T3}\| - B_3\}$
8	[3,4]	$\{-B_3\| - B_3\}$
9	[5,6]	$\{B_{T4}\}$
10	[8,9]	$\{-B_3\}$
11	[9,10]	$\{\}$

Locking the T-literal, B, in clause 1 higher than all non-T-literals of S, allows the proof to continue yielding the previously blocked result in clause 9 which leads to the generation of the empty clause. Recall that factoring is allowed on the T-lemma $\{B_{T4}|B_{T4}\}$ which allows clause 9 to be written as $\{B_{T4}\}$.

By adding this further restriction on the locking of T-literals, Theorem 4 can now be restated and proved. The proof remains the same except for an added note showing how the new condition was actually implicitly assumed by Peterson.

Theorem 5 *If S is an unsatisfiable Horn T set which is locked in such a way that the literals of T in any clause are locked greater than or equal to the all other literals of S, then there exists a lock-T refutation of S.*

Proof: Consider first the ground case. The proof will be by induction on the number of literals in T. If T is empty, the result we need reduces to the ground case of Theorem 1. Suppose that the result is true if T consists of N - 1 literals and suppose we now assume that T contains N literals. Let A be a literal of highest lock in T (and now of higher lock than any non-T-literal). Form S' by deleting A from the clause in which it appears and form T' from T by deleting A. Then S' is a Horn-T' set with the required type of locking. Thus S' has an LNL-T' refutation. When A is replaced in the clause from which it came, this refutation becomes an LNL-T proof of a clause consisting only of zero or more copies of A. (NOTE: In the original proof of Theorems 3 and 4, the previous statement made the implicit assumption that A was locked higher than any non-T-literals. The lock for literal A plays an important role since when A is replaced in a clause it must also be replaced in all the descendants of that clause. In order to obtain an LNL-T proof of a clause with zero or more copies of A, A must be locked higher than any of the non-T-literals of the the resolvents/descendants which contain it. If it is not, the LNL-T refutation may become a proof of a clause consisting of one or more copies of A followed by some other non-T-literals. Thus, if A is locked too low, it may prevent the proof of the desired result.) If this clause is {}, we are finished. Otherwise, factoring will yield the unit clause $\{A\}$. Now form S" by deleting from S all clauses containing A and adjoining the clause $\{A\}$. Then S" is also a Horn-T' set and the induction hypothesis yields an LNL-T refutation for S". This refutation combined with the earlier proof of $\{A\}$ gives the required refutation of S.

Theorem 3 also requires a stronger hypothesis. The proof is the same as for Theorem 4 except that it builds on Theorem 2 instead of Theorem 1.

Implementation Details

The most efficient way to implement the new proof strategies described above was to modify Otter 2.0, an automated theorem prover developed by William McCune at Argonne National Laboratories [2]. This allowed the use of existing routines to manage clauses (ie., input, output, factoring, unification, etc.) and to keep track of statistics (ie., number of clauses generated, number of clauses kept, etc.). Using the statistical routines allows for direct comparisons with results previously generated in Otter using other inference rules.

Peterson [4] suggested that an implementation of his two new proof strategies be based on Theorems 3 and 4, but that such an implementation allow the user to relax some of the strict conditions of the theorems set up to preserve completeness. The next few paragraphs outline those flexibilities.

In the actual implementation, factoring is done in either proof method only on the lowest locked T-literals in newly generated lemmas. For the sake of showing completeness, the factored literals were required to be instances of the same T-literal. The implementation does not check for this condition, but it can be forced by locking all T-literals uniquely so that any T-literals in a resolvent with the same lock must be instances of the same T-literal.

As shown in the previous section, another concern for completeness is that T-literals be locked greater than or equal to all non-T-literals in the set of clauses. This checking is left to the user, and, as the results will show, such a restriction can do more than just preserve completeness. For instance, literals which are known to produce many resolvents (ie., Equal(x,y)) are sometimes better off locked as high T-literals and thus put off until the later stages of the proof.

As Peterson pointed out, in a Lock-T proof, if in each clause negative literals are locked lower than positive literals and if T contains only positive literals, the result is a restriction of Robinson's P1-refutation [6]. This suggested heuristic for locking is merely echoed here for the reader's information and is not required

by the implementation.

Another suggested feature of the two methods is in allowing a restart of the proof procedure if memory overflows. One can simply retain the lemmas and the initial set of clauses, discard everything else, and start over. The specific algorithm for handling the restart procedure is given in the next section, but it should be pointed out that if the back subsumption flag is set and a non-T-lemma clause is generated which subsumes an input clause, the input clause will be deleted at the time of subsumption, and the non-T-lemma will be deleted upon restart of the proof procedure. Hence, to avoid this loss of completion the user should clear the back subsumption flag.

Chang and Lee [1] indicate that lock resolution, like most resolution strategies, is not complete with the set of support proof strategy. Therefore, this is a concern of the two newly implemented strategies. To overcome this when using the Lock-T proof strategy or regular lock resolution, all clauses should be put in the set of support list. This is not a concern of the LNL-T proof method because it does not actually use the set of support strategy.

The names for the subroutines which actually implemented Lock-T refutation and LNL-T refutation are unit-t-proof and t-lemma-proof, respectively. These pairs of terms will be used interchangeably throughout this work.

When using t-lemma-proof alone, the user should put at least one clause in the set of support list because Otter will not go into its main inference loop if this list is empty. Since t-lemma-proof treats the set of support list the same as the axiom list, this will have no effect on the proof strategy other than to just get it started. Otter will normally stop when there are no more clauses in the set of support list to move in to the axiom list and resolve with the other axioms. If this happens when the t-lemma-proof method is active, the proof may be prematurely halted if the t-lemma-proof strategy has not yet completed. To prevent this, the last given-clause is retained as a dummy clause to allow the loop to continue until t-lemma-proof either finishes or generates a clause which is kept and inserted in the set of support list.

Both of the methods are actually specializations of lock resolution where LNL-T proofs have restrictions on the locking of literals and for choosing clauses to be resolved. Lock-T proofs only have restrictions on the locking of literals. The implementation actually resulted in the addition of three resolution proof techniques, the first of which was lock resolution. The first modification was to allow Otter to accept a new input format for clauses with numbered/locked literals. When using any lock resolution based inference method, the new format requires literals in any list of clauses to be preceded by an "@", followed by a "T" if it is a T-literal, followed immediately by an integer and a blank. For instance, an acceptable form of

$$\{-P_3(x)| - L_6(x,y)|L_{T7}(g(z),y)\}$$

would be

$$\{@3\ -P(x)|@6\ -L(x,y)|@T7\ L(g(z),y)\}$$

Since Lock-T resolution is a specialization of lock resolution it was implemented with a simple call to the lock resolution routines. The preprocessing conditions for the initial clause set are handled at the time of input where the initial clause set is inspected to see if it satisfies the locking criteria and it is indeed a Horn$_T$ set. The postprocessing stipulations are that factoring is only allowed on T-lemma resolvents and only on the lowest locked literals, and that lemmas are cross-fed to the LNL-T proof strategy if it is active.

LNL-T refutation posed the most significant implementation problem because its depth-first pursuit of lemmas is not compatible with the set of support strategy which is the basis of Otter's main loop. This required Otter to be modified in such a way as to allow LNL-T proofs to be conducted independently of the other inference mechanisms. Recall that in each iteration of the main loop, a given-clause is moved from the set of support list to the list of axioms and each active inference mechanism is called upon to resolve the given-clause against the other axioms. Since Otter is a serial program, only one inference mechanism can be active at one time, so the pursuit of LNL-T proofs cannot be completely independent of the other active inference mechanisms. To accomplish the necessary time slicing, LNL-T-proof is called each time through the main loop allowing each LNL-T proof to be advanced a little further by an amount determined by the user. Lemmas generated by LNL-T-proof are crossfed to the other active inference mechanisms by placing them in the set of support. To sustain the development of the LNL-T proofs, the set of support list must never become empty before LNL-T-proof has finished. When this happens (if LNL-T-proof is turned on) the last given-clause is retained as dummy set of support clause until LNL-T-proof generates and crossfeeds a lemma to replenish the set of support which in turn also has the effect of reviving the active set of support based resolution strategies and the main loop continues.

One non-Horn problem used to test the newly implemented proof strategies is known as the "non-obvious problem" [3]:

$$\{-P(x,y)| - P(y,z)|P(x,z)\}$$

$$\{-Q(x,y)|-Q(y,z)|Q(x,z)\}$$
$$\{-Q(x,y)|Q(y,x)\}$$
$$\{P(x,y)|Q(x,y)\}$$
$$\{-P(a,b)\}$$
$$\{-Q(c,d)\}$$

In order to avoid the incorporation of any heuristics in the clause set, two of the most generic of the various possible Horn$_T$ locking schemes which satisfy only the mimimum locking restrictions for the two techniques were used. That is, in all of the Horn clauses, the negative literals are locked at the same level, and the positive literal in each clause has the lowest lock. As for the non-Horn clause, the first positive literal is locked lowest while the second literal is locked as a T-literal, and vice versa. The two unit clauses represent the negation of the theorem and are placed in the set of support.

All of the successful runs find proofs of about the same length (28 steps), however, the Lock-T-proof method exhibited the best overall performance by generating and keeping fewer clauses on both sets of clauses. The first locking for these clauses is:

$$\{-P_1(a,b)\}$$
$$\{-Q_1(c,d)\}$$
$$\{-P_2(x,y)|-P_2(y,z)|P_1(x,z)\}$$
$$\{-Q_2(x,y)|-Q_2(y,z)|Q_1(x,z)\}$$
$$\{-Q_2(x,y)|Q_1(y,x)\}$$
$$\{P_1(x,y)|Q_{T4}(x,y)\}$$

The second locking replaces the last clause by:

$$\{P_{T4}(x,y)|Q_1(x,y)\}$$

When run with the unlocked set of clauses, hyper-resolution produced 1995 clauses retaining 355 for a proof of length 18, binary resolution produced 673 clauses retaining 283 for a proof of length 22, and UR-resolution was unable to find a proof. Lock-T resolution generated 201 clauses retaining 87, thus Lock-T refutation outperformed each of these methods in clause generation and retention. Some other less general locking schemes were tried where the first negative literals (if any) in each clause were locked at 2 and the second negative literals (if any) were locked at 3 and all other literals were locked as before. This resulted in proofs of about the same length but with as few as 105 clauses generated and 72 clauses kept.

References

[1] Chang, C.L. and Lee, T.T.L.,(1973). *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, NY.

[2] McCune, William W.,(1990). "Otter 2.0 Users Guide", Argonne National Laboratory, Mathematics and Computer Science Division, Argonne, IL.

[3] Pelletier, F.J. and Rudnicki,(1986). "Non-Obviousness", AAR Newsletter, 6.

[4] Peterson, G. E.,(1976). "Theorem Proving with Lemmas", *Journal of the ACM*, 23(4), 574-581.

[5] Quaife, Art,(1989). "Automated development of Tarski's Geometry", *Journal of Automated Reasoning*, 5, 97-118.

[6] Robinson, J.A.,(1965). "Automatic deduction with hyper-resolution", *Int. J. of Computational Mathematics*, 1, 227-234.