Lemma and Cut Strategies for Two-Sided Propositional Resolution
(extended abstract)

Allen Van Gelder   Fumiaki Kamiya
Computer Science Dept., 225 AS
University of California
Santa Cruz, CA 95064 U.S.A.
E-mail avg@cs.ucsc.edu, kamiya@cs.ucsc.edu.

April 7, 1996

Abstract

Resolution has not been an effective tool for deciding satisfiability of propositional CNF formulas, due to explosion of the search space, particularly when the formula is satisfiable. However, a new pruning method, which is designed to eliminate certain refutation attempts that cannot succeed, has been shown to eliminate much of the redundancy of propositional model elimination. The pruning method exploits the concept of "autarky", which was introduced by Monien and Speckenmeyer. Informally, an autarky is a "self-sufficient" model for some clauses, but which does not affect the remaining clauses of the formula. Autarky pruning permits the algorithm, called "Modoc", to be "two-sided" in the sense that it constructs a model if the formula is satisfiable and constructs a refutation proof if it is not.

This talk describes new "lemma" and "cut" strategies that are efficient to apply in the setting of propositional resolution. It builds upon the C-literal strategy proposed by Shostak, and studied further by Letz, Mayr and Goller. Methods for "eager" lemmas, "quasi-persistent" lemmas, and two forms of controlled cut have been incorporated into Modoc. While these strategies are not necessary for the theoretical completeness of Modoc, experiments show them to greatly increase the efficiency in practice.

Experimental data based on an implementation in C is reported. On random 3CNF formulas at the "hard" ratio of 4.27 clauses per variable, Modoc is not as effective as recently reported model-searching methods. On more structured formulas from applications, such as circuit-fault detection, it is superior. This performance is achieved in spite of the fact that Modoc incorporates almost no heuristics to guide its searches.

Key Words: Satisfiability, Boolean formula, propositional formula, autarky, resolution, refutation, model, theorem proving, model elimination.

1 Introduction

The decision problem of Boolean, or propositional, satisfiability is the "original" NP-hard problem. We assume the reader is generally familiar with it. We shall consider exclusively propositional formulas in *conjunctive normal form* (CNF), also called *clause form*. As a practical consideration, a program may be required to produce "evidence", or a "certificate", to support its decision, an issue discussed by John Slaney at CADE-12 [Sla94]. If critical decisions will be based on the program's output, such evidence should permit an independent, and straightforward, verification of the program's conclusion. To our knowledge, no previously existing implementations can produce evidence for both "sat" and "unsat" decisions.

Two basic methods have been developed for satisfiability testing: refutation search and model search (see [VG95] for additional bibliography).

1. Refutation search seeks to discover a proof that a formula is unsatisfiable, usually employing resolution. If a complete search for a refutation fails, the formula is pronounced satisfiable. Model elimination and SL-resolution typify these methods [Lov69, KK71, Lov72].

2. Model search seeks to discover a satisfying assignment, or model, for the formula. If a complete search for a model fails, the formula is pronounced unsatisfiable. The DPLL algorithm, due to Davis, Putnam, Loveland and Logemann is the basis for many modern refinements. A different approach is to treat the problem in terms of integer linear programming.

   Several methods use incomplete model searches, so they can only report "don't know" and give up based on resource limits, when they fail to discover a model. However, they have succeeded in finding models on much larger formulas than current complete methods can handle.

Current propositional methods are "one-sided" in the sense that they can produce "evidence", or a "certificate", for one side of the decision problem, but not the other. In the first-order arena there has been some work on searching for a model as well as a refutation, but it does not seem to carry over effectively to the propositional case. See [VG95] for bibliography. Refutation methods do not produce a model on satisfiable formulas, and model-search methods do not produce a refutation on unsatisfiable formulas. This paper describes an integrated approach that simultaneously searches for either a refutation or a model of a propositional formula.

One problem with existing propositional refutation methods was illustrated dramatically in a study of behavior on random 3CNF formulas (4.27 clauses per variable) and other small formulas [VGK96]. Model elimination (abbreviated M.E., and regarded as one of the most efficient refutation strategies) is able to solve *unsatisfiable* random 3CNF formulas with up to 100 variables, but it bogs down on *satisfiable* formulas at about 17 variables. It should be noted that, for modern model-search methods, these formulas (both unsatisfiable and satisfiable) are considered *easy* at 100 variables and *trivial* at 20 variables.

Three reasons have been mentioned for the lack of high performance propositional refutation systems:

1. The search space is too large, particularly for satisfiable formulas, as mentioned.

2. The method cannot produce models.

3. The most efficient methods are only guaranteed to find a refutation when some *top clause* is known to be in a minimal unsatisfiable subset of clauses; in some applications such a top clause is not known *a priori*.

The integrated approach of this paper addresses all three of these problems.

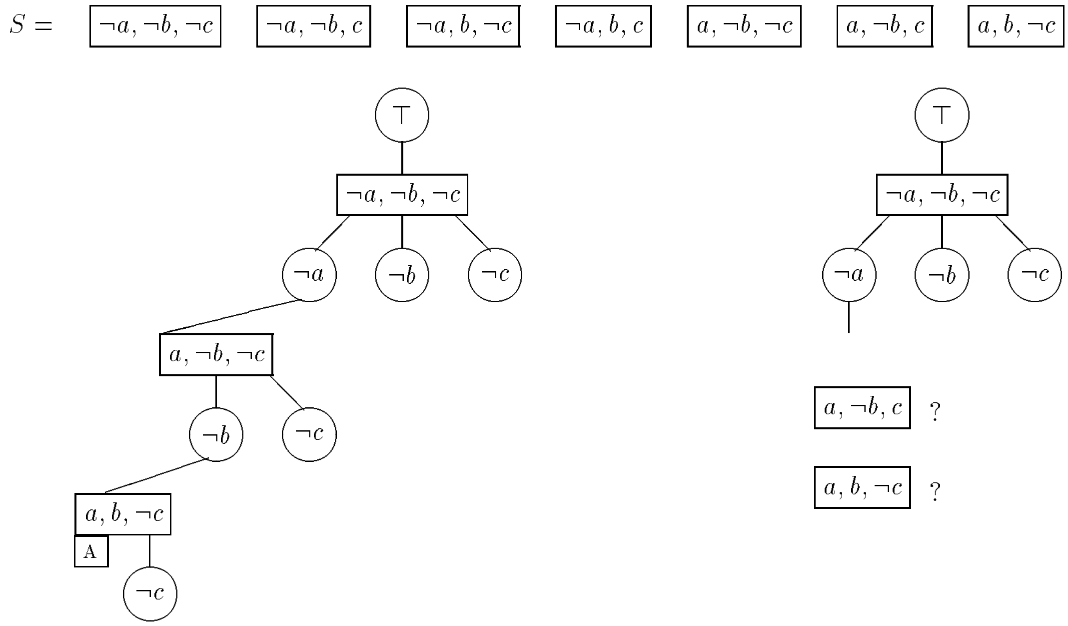

Figure 1: Model elimination search for Example 1.1. (Left) Search fails at lowest $\neg c$ goal. (Right) After backtracking to alternative choices at $\neg a$ goal.

The main results are summarized in Section 1.1 and informally illustrated with examples in Section 1.2. Tree structures for M.E., other background, and *autarky pruning* are described elsewhere [VG95, VGK96]. New lemma creation mechanisms are described in Section 2, which also briefly reviews previous work on lemmas in M.E. Experimental results based on an efficient C implementation are reported in Section 3. Conclusions and future work are in Section 4.

1.1 Summary of Results

This paper reports on the combination of autarky pruning with extended lemma creation methods. Roughly speaking, lemmas and autarkies are duals of each other: a lemma permits curtailment of a resolution attempt that is certain to succeed, while an autarky permits curtailment of a resolution attempt that is certain to fail. Experimental data is based on an efficient C implementation.

New lemma creation mechanisms, called "eager" lemmas and "quasi-persistent" lemmas are described in Section 2. That section also briefly reviews previous work on lemmas in M.E., and discusses the compatibility of autarky pruning and lemma creation. Quasi-persistent lemmas are a variant of the "C-literal" strategy, adapted for efficiency in the propositional case. Eager lemmas supplement them to provide substantial further reductions in proof searching by providing early identification of refutations that will succeed. Derivation of eager lemmas is closely related to unit clause propagation.

When the refutation of a literal succeeds, the complement of that literal becomes a quasi-persistent lemma and the complements of certain eager lemmas become quasi-persistent lemmas also. The eager lemmas whose complements survive as quasi-persistent lemmas are shown to correspond to articulation points of a graph related to the successful refutation. Articulation points can be identified in linear time by a standard graph algorithm.

1.2 Informal Overview

For readers familiar with model elimination adapted to tree structures [MZ82, LMG94], we now give simple examples of how autarky pruning and eager lemmas work. The tree data structure is called a propositional derivation tree (PDT). Complete details may be found in technical reports [VG95, VGK96].

Example 1.1: The formula S consists of all 3CNF clauses on variables a, b, and c, except for the all positive clause, as shown at the top of Figure 1. Suppose the top clause is all negative. Let us trace out a model elimination search for a refutation. As shown on the left of Figure 1, literal $\neg a$ resolves with clause $[a, \neg b, \neg c]$, then literal $\neg b$ resolves with clause $[a, b, \neg c]$. In the latter clause, literal a can be reduced with the ancestor (or A-literal) $\neg a$, as indicated by the boxed "A". Literal $\neg c$ remains to be refuted, but the search procedure now fails, because each clause containing literal c also contains an ancestor literal.

If we stop and reflect on the meaning of this failure, we see that every clause containing the literal c is satisfied by a partial assignment consisting of the ancestors (A-literals) on this branch, specifically:

$$M = \{\neg a, \neg b, \neg c\}.$$

(In this case the partial assignment happens to be a total assignment.) But obviously, every clause containing the literal $\neg c$ is also satisfied by M, so we conclude that every clause involving the *variable* c is satisfied by M.

Model elimination now backtracks and looks for another clause that resolves with $\neg b$, and does not contain the ancestor $\neg a$. There are none. We can now extend the conclusion of the previous paragraph to say that every clause containing either of the *variables* b or c, either positively or negatively, is satisfied by the partial assignment M.

Model elimination again backtracks and looks for another clause that resolves with $\neg a$ (Figure 1, right). There are two such clauses, as indicated. The standard algorithm would continue trying to construct a refutation using one of these clauses, then the other. But notice that both of these clauses are satisfied by the partial assignment M mentioned above.

After a few moments thought, we can predict that these refutation attempts must fail, without carrying out the search. Intuitively, the reason is that we cannot use a clause that is satisfied by M to "get outside of M". Every extension will have a subgoal that is satisfied by M; no such subgoal can be handled by reduction because all the ancestors are also in M. Eventually, some goal is generated that has no eligible extensions.

Finally, we conclude that the partial assignment M satisfies all clauses in which any of the variables a, b, or c appears. This conclusion holds up even if we add additional clauses to S that do not involve the variables a, b and c. We call such a partial assignment an *autarky* [VG95].

This example illustrates, in an over-simplified way, that:

1. Autarky analysis can predict that certain refutation attempts must fail;

2. A model for a satisfiable formula can be constructed as a series of autarkies.

□

Example 1.2: Figure 2 illustrates the main idea of eager lemmas, introduced in Section 2.4. To avoid clutter, parts of the derivation that are not germane to the discussion are omitted. In Modoc, when subgoal $\neg k$ is selected, clause $[\neg e, k, f]$ is "pre-reduced" through unit implication (unit-clause resolution without further propagation). Then goal $\neg k$ is extended with clause $[a, k]$, creating subgoal a. Subgoal a creates an eager literal (E-literal) $\neg m$ by unit clause propagation. In the process, four clauses are "pre-reduced" as

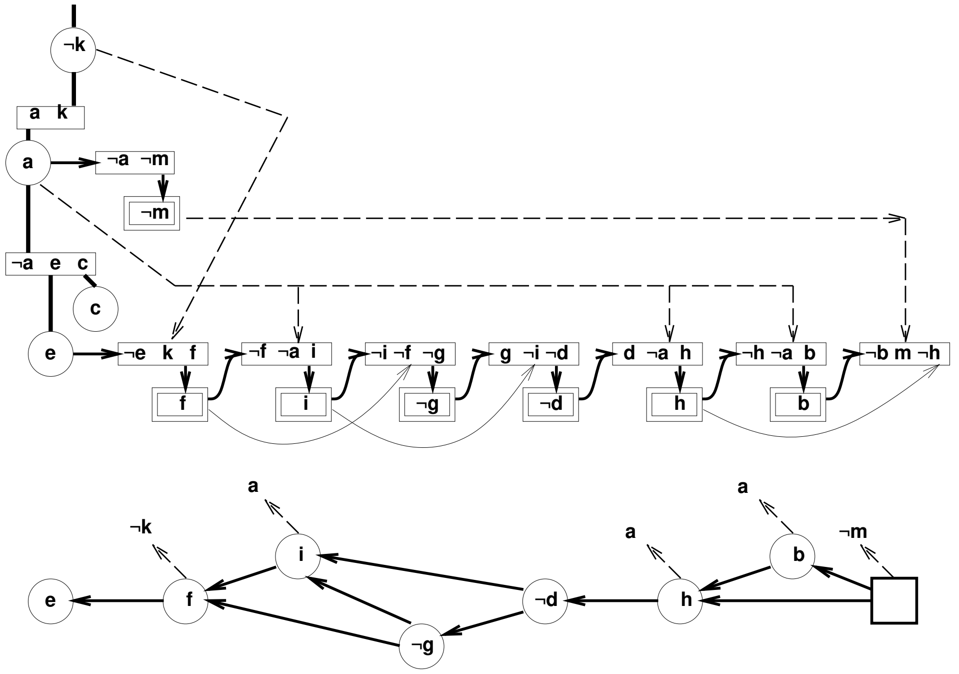

Figure 2: Eager lemmas, discussed in Example 1.2.

---

shown by the dashed arrows from a and $\neg m$. Rather than extending with the clause $[\neg a, \neg m]$, we assume the program extends a with $[\neg a, e, c]$, as shown. This extension creates two subgoals and we assume e is chosen first. When e is used to initiate unit clause propagation, a series of six E-literals are derived by unit clause propagation, as shown in double boxes, and the empty clause is derived. In this process, additional E-literals might have been derived, but are not shown, and the E-literals normally are not derived consecutively.

Since the empty clause was derived, we know that e can be refuted. To find the clauses relevant to the refutation, we want to work back from the empty clause to e. Remember that there are normally irrelevant E-literals as well. As each E-literal is derived, the clause that led to its derivation, called its *eager parent*, is associated with it. Eager parents are shown immediately above their E-literals. Working back from the empty clause, a directed acyclic graph (DAG) is defined by constructing edges from an E-literal (e.g., $\neg d$) to the E-literals whose complements appear in the clause that derived it (e.g., i and $\neg g$), as shown in the lower part of the figure. This is called the *eager dependency DAG*. Edges to literals that are higher in the tree than e (shown as dashed lines) are not considered part of the DAG.

The key theorem (Theorem 2.1) is that articulation nodes of the DAG are precisely the E-literals whose complements become quasi-persistent lemmas (similar to C-literals). Of course, $\neg e$, the complement of the refuted subgoal, becomes a quasi-persistent lemma, as well. A further interesting property is that the dependencies are found by following the dashed edges from DAG nodes that have a path to the articulation node.

Thus in this example, $\neg h$, d and $\neg f$ become C-literals with dependency a. (The actual lemmas are $[\neg h, \neg a]$, etc.) The E-literal $\neg m$ is not itself a dependency, but the ancestors upon which it depends are; in this case, $\neg m$ depends on a only. Also, $\neg e$ becomes a C-literal with dependencies a and $\neg k$. These C-literals could have been derived by the normal operations of model elimination, but possibly only after lengthy fruitless side trails. □

2 Lemmas and C-Literals

In the model elimination procedure, a "lemma" may be recorded upon the completion of any (sub)refutation [Lov69, FLSY74, Lov78], although this is not necessary for completeness. Shostak proposed an efficient "C-literal" mechanism to maintain such lemmas in model elimination chains [Sho74, Sho76]; Letz *et al.* generalized it to trees with an ingenious time-stamping method, and also proposed a pruning strategy based on C-literals, called "strong regularity" [LMG94]. Lemma strategies have been studied empirically for first order theorem proving using chains [FLSY74, AS92, Sti94], as well as trees [LMG94], but we are aware of no empirical studies of lemma strategies on propositional problems. This section sketches how lemmas are incorporated into the implementation of the Modoc algorithm. We introduce and describe a strategy for "quasi-persistent" lemmas and "eager" lemmas as well as two forms of "cuts" that permit refutations to be accelerated.

In Modoc, autarky pruning is compatible with the use of lemmas, and largely orthogonal. Clauses that are pruned by an autarky cannot participate in a successful refutation (at the point where they are pruned), whether or not lemmas are used to shorten the refutation.

Suppose the refutation of a literal q is completed in a PDT. Let B be the subset of ancestors of q that were actually used for reductions in q's refutation (say $B = \{p_1, \ldots, p_m\}$, where m may be 0). Then a lemma clause, $[\neg q, \neg p_1, \ldots, \neg p_m]$, can be derived soundly [LMG94].

In the lemma $[\neg q, \neg p_1, \ldots, \neg p_m]$, literal $\neg q$ is called a "C-literal" and is attached to the *lowest* ancestor (say p_c) among $\{p_1, \ldots, p_m\}$. (If $m = 0$, attach it to the root of the PDT, which is normally $\top$.) The C-literal can only be used in the subtree of p_c, and in this context, its operational behavior is somewhat like reduction with an ancestor. Hence, the operation is sometimes called "C-reduction".

If the PDT is abandoned (because some other part of the refutation fails) then the lemma is forgotten. If the (sub)refutation of p_c is completed, the lemma is also forgotten in the sense that it is not used later in other (sub)refutations. Because of the limited application and lifetime of the lemma, it suffices to "attach" the C-literal $\neg q$ to the lowest ancestor p_c, and not record the dependencies B. Modoc cannot adopt this simple strategy because it does not completely abandon a PDT when some part of the refutation fails, as discussed in the next section.

2.1 Quasi-Persistent Lemmas

Our strategy varies from the C-literal strategy described above in that lemmas derived during failed (sub)-refutations are not necessarily forgotten. Normally, a PDT is not completely abandoned, but only the subtree where the refutation fails is abandoned. (In the first order case, substitutions need to be backed out, as well.) The lemma can function as a C-literal until the subtree rooted at p_c is abandoned, or the refutation of p_c is completed (where p_c and other terminology is continued from the previous subsection).

Modoc maintains lemmas attached at p_c until the tree rooted there is abandoned or its refutation is completed. The previously described strategy of Letz *et al.* effectively deletes the lemma as soon as the refutation of any clause ancestor of q fails. There are pros and cons of both strategies. Our strategy makes it unnecessary to re-derive the same lemma at the same attachment point so often, but it makes it necessary to record the full lemma. Also we did not see a way to adapt their time-stamping method to the situation in which only a small part of the tree is abandoned when a goal fails: some kind of "roll-back" of the time-stamps is needed.

Our strategy is incompatible with the heuristic called "strong regularity", introduced by Letz *et al.* That is, Modoc may undertake to refute a goal $\neg q$ in the subtree (rooted at p_c) where $\neg q$ is attached as a lemma. The "strong regularity" heuristic consists of avoiding such attempts. "Strong regularity" was shown to be complete under certain conditions, but quasi-persistent lemmas do not meet those conditions, and a counter-example can be constructed if the two heuristics are combined.

The quasi-persistent heuristic holds lemmas longer, but spends more time per lemma in bookkeeping, compared to the traditional C-literal method. There is no apparent way to determine which method performs better except empirical testing.

2.2 Lemma-Induced Cuts

We now describe the method by which Modoc exploits complementary C-literals. Suppose, as described above, the C-literal $\neg q$ is attached at p_c and the goal $\neg q$ occurs in a subtree of p_c. Should the refutation of $\neg q$ be successful, there results a new lemma whose C-literal is q. Now, complementary C-literals have been derived on one branch. Let the full form of the second lemma be $[q, \neg r_1, \ldots, \neg r_n]$; that is, $\{r_1, \ldots, r_n\}$ is exactly the set of ancestors used in the refutation of $\neg q$. Again, let r_c be the lowest ancestor among the r_i, or the root $\top$ if $n = 0$.

Now consider the lower of the two goal nodes p_c and r_c. The situation is symmetric, so let us suppose it is r_c. Let A' be the ancestors of r_c. Now add a "virtual clause" $[\neg r_c, q, \neg q]$ to the formula; this is a tautology, so it is harmless. However, extending r_c with this virtual clause creates goals q and $\neg q$, both of which are immediately closed by the lemmas. Thus the goal r_c is immediately refuted, *even though the tree in which the lemmas q and $\neg q$ were derived is never completed to a refutation.*

Introduction of the "virtual clause" described above is essentially a form of the cut rule [LMG94]. If S is the original set of clauses, we have discovered $(S + A') \vdash q$ and $(S + A') \vdash \neg q$. Now the cut rule infers $(S + A') \vdash \emptyset$.

While the introduction of a tautologous clause is always sound, it normally is not practical because the prover has no way to anticipate that each of the complementary literals has a short refutation. However, if a pair of complementary C-literals have been derived, then the prover has that information in hand.

This methodology also can be applied to first order proofs where the prover is not using strong regularity. In this case, a most general unifier of the complementary C-literals would be applied before creating the "virtual clause".

2.3 C-Reduction-Induced Cuts

When Modoc selects an ancestor, it strengthens the formula with this ancestor. Strengthening is the combination of unit subsumption and unit implication (unit-clause resolution without further propagation). Similarly, when Modoc installs a C-literal, it applies unit implication with this C-literal, which amounts to a form of "eager C-reduction". If this creates an empty clause, then a *C-reduction-induced cut* may occur.

Suppose the original clause $C = [\neg r_1, \ldots, \neg r_n]$ shrinks to 0 length due to unit implication by a new C-literal r_c. Without loss of generality, assume that r_1 is the lowest ancestor or C-literal among $r_1, \ldots, r_n$, chosen to be an ancestor if possible. Let q_1 be the ancestor at the depth of r_1. (If r_1 is an ancestor, $q_1 = r_1 \neq r_c$.) If $q_1 = r_1$, then extension of q_1 by C refutes it immediately. If $q_1 \neq r_1$, we extend q_1 by the "virtual clause" $[\neg q_1, \neg r_1, r_1]$, again using a form of "cut". Subgoal $\neg r_1$ is C-reduced and subgoal r_1 is immediately refuted by extension with C.

The effect of this cut is similar to the lemma-induced cuts (Section 2.2) in that (possibly) an intermediate goal node can be refuted without completing the refutation in progress for it.

2.4 Eager Lemma Assertion

Unit-clause propagation is of interest because of its efficiency, and its frequent use as a subroutine in model-searching algorithms, such as DPLL. Dalal and Etherington have shown that unit-clause propagation can be implemented in linear time [DE92]. This implementation is practical, as well as theoretical, and is used by numerous implementers. Modoc uses it to derive *eager lemmas*.

When Modoc selects a new goal node e, besides performing unit implication with it, it performs unit-clause propagation with any clauses whose (strengthened) length reduces to 1. All such clauses are asserted as *E-literals* attached to the goal node e. Like C-literals, E-literals are logical consequences of the ancestors of e and can be used in the same manner in the refutation search below e.

An interesting situation arises when the above unit-clause propagation directly derives the empty clause. Clearly, this implies that a refutation of e exists. To extract the quasi-persistent lemma with $\neg e$, the appropriate dependencies must be determined. The derivation of E-literals creates a directed acyclic graph (DAG) of dependencies among E-literals and the selected goal node e, as illustrated in the lower part of Figure 2, and discussed in Example 1.2.

Let us call this the *eager dependency DAG*. To simplify the remainder of this discussion, let us call e an E-literal, too. In this DAG, the edge $p \rightarrow q$ denotes that q is an E-literal attached to the selected goal e, and the clause C_p that derived p by unit propagation contains $\neg q$. Further, let us call C_p the *eager parent* of p, and let us call the complements of the literals in $C_p - p$, *other than* E-literals, the *nonlocal dependencies* of p. Finally, let C_0 be the clause that became empty. Then C_0 is the source node of the DAG; edge $C_0 \rightarrow q$ denotes that q is an E-literal attached to the selected goal e, and C_0 contains $\neg q$; the complements of the literals of C_0, *other than* E-literals, are the *nonlocal dependencies* of C_0.

Theorem 2.1: With the foregoing notation, if p is an articulation point in the eager dependency DAG, then there is a refutation of e in which $\neg p$ becomes a C-literal at a depth less than the depth of e. Moreover, the dependencies of $\neg p$ are the nonlocal dependencies of nodes having a path to p in the DAG. ∎

Corollary 2.2: With the foregoing notation, dependencies of the C-literal $\neg e$ are the nonlocal dependencies of all nodes other than e in the eager dependency DAG. ∎

If p is not an articulation point, then the refutation implied by the eager dependency DAG uses some reduction with a node later in the DAG, and the C-literal $\neg p$ would be attached there, and would be discarded by the time e was refuted.

3 Experimental Results

An efficient implementation of Modoc was programmed in C. Hereinafter, we refer to this C implementation of Modoc as `modoc`. Space prevents discussion of important data structure issues. This section reports on performance tests. The CPU time for a Sun Sparcstation 10/41 is reported.

The effects of various lemma and cut strategies are summarized, and compared with `2cl`, a model-search program, in Figure 3. Background on the formulas and `2cl` is reported elsewhere [VGT96]. The table shows significant improvements as various lemma strategies and cuts are incorporated into `modoc`. We observe that `modoc` runs 30–40 times faster than `2cl` on the hard classes of circuit formulas, but runs slower than `2cl` on random formulas.

Growth rates on random 3CNF formulas are shown in Figure 4, for `modoc` and two model-searching programs, `2cl` and an efficient `DPLL` (Davis-Putnam-Logemann-Loveland), which used linear-time techniques for unit-clause propagation [DE92], unit subsumption, and pure literal pruning. This growth plot indicates that `modoc` still has a higher asymptotic growth rate than model-searching methods, on random formulas.

formula class	num. fmlas	none avg	none max	quasi-persistent avg	quasi-persistent max	quasi-persistent, cuts avg	quasi-persistent, cuts max	quasi-persistent cuts, eager avg	quasi-persistent cuts, eager max	2cl avg	2cl max
		Lemma Features in modoc									
rand100	50	≥1200	≥1200	163	452	16	54	1.59	5.15	1.10	3.23
rand141	50	≥1200	≥1200	≥1087	≥1200	403	1243	35	119	9	22
bf2670	53	≥1175	≥1200	≥853	≥1200	311	5736	8	183	439	17725
ssa2670	12	≥1200	≥1200	≥1200	≥1200	205	592	29	163	1540	2424

Figure 3: Improvements due to various lemma strategies and cuts. Times are cpu seconds on a Sun sparc 10/41. Classes `rand100` and `rand141` consist of random 3CNF formulas at the "hard" clause/variable ratio of 4.27, with 100 and 141 variables, respectively. Classes `bf2670` and `ssa2670` derive from the most challenging circuit in the ISCAS-85 benchmark.

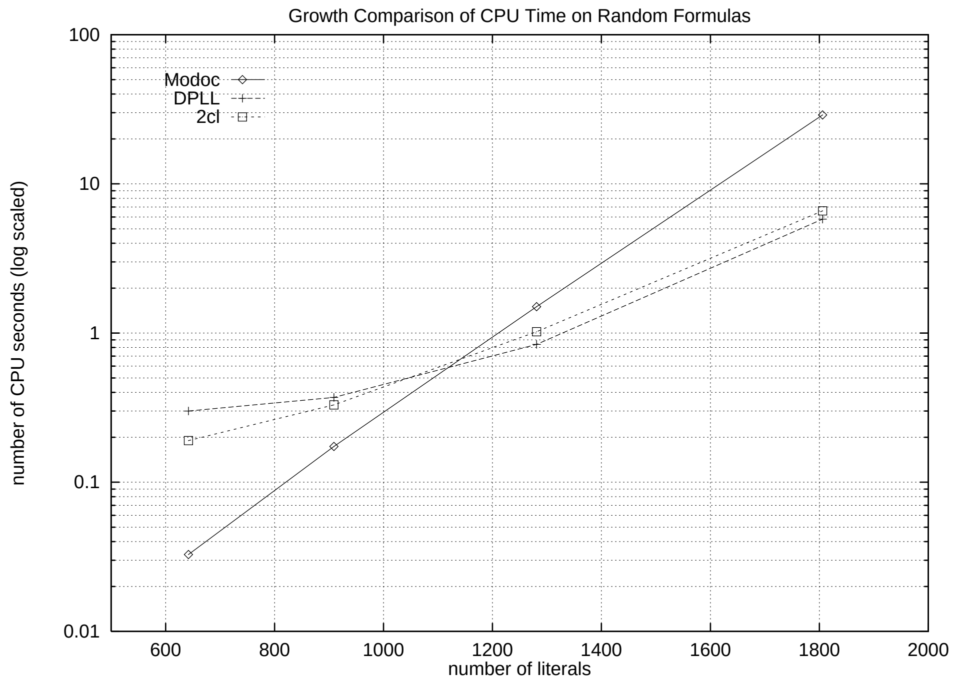

Figure 4: Growth comparison on samples of 50 random 3CNF formulas with 4.27 clauses per variable. (E.g., 100 variables corresponds to 1281 literals.)

4 Conclusions and Future Work

We have introduced a method to incorporate autarky analysis into propositional resolution procedures. We have also introduced the eager lemma strategy and two forms of controlled cut.

Tests reported in Section 3, and others omitted for lack of space [VGK96], contain substantial evidence that the combination of autarky pruning and innovative lemma and cut strategies, as presented in this paper, overcomes the major inefficiency of model elimination. Modoc is not yet competitive with the leading model-search methods on random formulas, but outperforms them on formulas derived from a circuit fault testing application.

Future work should proceed along several directions, including heuristics for guiding the resolution search, further improvements to lemma caching, and an extension to first-order theorem proving. Another open question is the worst-case complexity of Modoc or an improved version of Modoc.

Acknowledgments

This work was supported in part by NSF grant CCR-95-03830, by equipment donations from Sun Microsystems, Inc., and software donations from Quintus Computer Systems, Inc. We thank Dr. Reinhold Letz for helpful discussions on many aspects of model elimination and tableau methods.

References

[AS92] O. L. Astrachan and M. E. Stickel. Caching and lemmaizing in model elimination theorem provers. In D. Kapur, editor, *Proc. 11th Int'l Conf. on Automated Deduction*, pages 224–38. Springer-Verlag, 1992.

[DE92] M. Dalal and D. Etherington. A hierarchy of tractable satisfiability problems. *Information Processing Letters*, 44:173–180, December 1992.

[FLSY74] S. Fleisig, D. W. Loveland, A. K. Smiley, and D. L. Yarmush. An implementation of the model elimination proof procedure. *JACM*, 21(1):124–139, 1974.

[KK71] R. Kowalski and D. Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2(2/3):227–260, Winter 1971.

[LMG94] R. Letz, K. Mayr, and C. Goller. Controlled integration of the cut rule into connection tableau calculi. *Journal of Automated Reasoning*, 13(3):297–337, December 1994.

[Lov69] D. W. Loveland. A simplified format for the model elimination theorem-proving procedure. *Journal of the Association for Computing Machinery*, 16(3):349–363, July 1969.

[Lov72] D. W. Loveland. A unifying view of some linear Herbrand procedures. *Journal of the Association for Computing Machinery*, 19(2):366–384, April 1972.

[Lov78] D. W. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, Amsterdam, 1978.

[MZ82] J. Minker and G Zanon. An extension to linear resolution with selection function. *Information Processing Letters*, 14(3):191–194, June 1982.

[Sho74] R. E. Shostak. *A Graph-Theoretic View of Resolution Theorem-Proving.* PhD thesis, Center for Research in Computing Technology, Harvard University, 1974. Also available from CSL, SRI International, Menlo Park, CA.

[Sho76] R. E. Shostak. Refutation graphs. *Artificial Intelligence*, 7:51–64, 1976.

[Sla94] J. Slaney. The crisis in finite mathematics: Automated reasoning as cause and cure. In A. Bundy, editor, *Proc. 12th Int'l Conf. on Automated Deduction*, pages 1–13. Springer-Verlag, 1994.

[Sti94] M. E. Stickel. Upside-down meta-interpretation of the model elimination theorem-proving procedure for deduction and abduction. *J. Automated Reasoning*, 13(2):189–210, October 1994.

[VG95] A. Van Gelder. Simultaneous construction of refutations and models for propositional formulas. Technical Report UCSC–CRL–95–61, UC Santa Cruz, Santa Cruz, CA., 1995.

[VGK96] A. Van Gelder and F. Kamiya. The partial rehabilitation of propositional resolution. Technical Report UCSC–CRL–96–04, UC Santa Cruz, Santa Cruz, CA., 1996.

[VGT96] A. Van Gelder and Y. K. Tsuji. Satisfiability testing with more reasoning and less guessing. In D. S. Johnson and M. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge.*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996.