Memoization and DPLL: Formula Caching Proof Systems

Paul Beame*
Computer Science and Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350
beame@cs.washington.edu

Russell Impagliazzo†
Computer Science and Engineering
UC, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0114
russell@cs.ucsd.edu

Toniann Pitassi‡
Computer Science Department
University of Toronto
Toronto, Ontario
Canada M5S 1A4
toni@cs.toronto.edu

Nathan Segerlind§
Computer Science and Engineering
UC, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0114
nsegerli@cs.ucsd.edu

Abstract

A fruitful connection between algorithm design and proof complexity is the formalization of the $DPLL$ approach to satisfiability testing in terms of tree-like resolution proofs. We consider extensions of the $DPLL$ approach that add some version of memoization*, remembering formulas the algorithm has previously shown unsatisfiable. Various versions of such* formula caching *algorithms have been suggested for satisfiability and stochastic satisfiability ([10, 1]). We formalize this method, and characterize the strength of various versions in terms of proof systems. These proof systems seem to be both new and simple, and have a rich structure. We compare their strength to several studied proof systems: tree-like resolution, regular resolution, general resolution, and $Res(k)$. We give both simulations and separations.*

1 Introduction

An abstract propositional proof system can be defined as a nondeterministic algorithm for accepting propositional tautologies (or, equivalently, refuting contradictions). We can also view such a nondeterministic algorithm as a *general approach* to divising *deterministic algorithms*, where the designer of the deterministic algorithm replaces the nondeterministic choices with a deterministic rule.

Many of the most interesting and productive algorithmic approaches to satisfiability can be classified by such nondeterministic algorithms. The classic example is the DPLL backtracking approach to satisfiability. This approach can be summarized by the following nondeterministic algorithm, whose input is a CNF formula F:

DPLL(F){
  If F is empty
    Report *satisfiable* and **halt**
  If F is contains the empty clause Λ
    return
  Else *choose a literal* x
    DPLL($F|_x$)
    DPLL($F|_{\overline{x}}$)}

The key nondeterministic step is when the algorithm chooses the branching literal x. To create a deterministic DPLL algorithm, a deterministic rule must be given for this choice. In fact, many such deterministic rules have been suggested, and the performance, empirically, has been found to be quite sensitive to the choice of this rule.

Since there are unlimited numbers of deterministic versions, it seems impossible to exactly analyze all possible variations. However, the performance of the *nondeterministic* version of this algorithm can be characterized as a conventional proof system, tree-like resolution. Lower bounds for tree-like resolution refutations (e.g. [9, 7, 4, 5, 6, 3])

* Research supported by NSF CCR-0098066 and ITR-0219468.
† Research supported by NSF CCR-0098197.
‡ Research supported by NSERC and an Ontario PRE Award.
§ Research supported by NSF DMS-0100589 and CCR-0098197.

Proceedings of The 18th IEEE Annual Conference on Computational Complexity
ISBN 0-7695-1879-6/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

then can be used to prove the limitations of any deterministic instantiation.

Recently many variations of DPLL have been introduced (both for satisfiability and stochastic satisfiability). One recent emerging idea is to cache intermediate results as the DPLL tree is searched. The technique of clause-learning, for which there have been many good implementations [11, 16, 13, 17], can be viewed as a form of memoization of DPLL where the algorithm caches, in the form of learned clauses, partial assignments that force contradictions. This technique which can be efficiently simulated by Resolution is studied from the proof complexity point of view in [2]. More generally, *memoization*, saving solved sub-problems, is a modification that is useful in a variety of back-tracking algorithms. For example, Robson uses memoization to speed up a back-tracking algorithm for maximum independent set [14].

The methods that we are interested here involve caching unsatisfiable residual formulas rather than caching partial assignments. They were first defined in [10] where DPLL-based algorithms with caching are studied and implemented to solve large probabilistic planning problems. In that paper, there are no analytic runtime guarantees, although the empirical results are very promising. Very recently, [1] define DPLL-based algorithms with caching for counting satisfying assignments and Bayesian inference and give time and space bounds that are as good as any known algorithm for these problems in terms of a connectivity measure of the underlying set of clauses/Bayes network.

Thus while caching in many different guises for DPLL has been studied in the past, this paper is the first to specifically formalize *proof systems for SAT* based on adding memoization of residual formulas to DPLL, and to analyze the complexity of these systems relative to standard systems. Many of our results are surprising, since at first glance it seems that adding memoization to DPLL cannot strengthen the system beyond Resolution.

In this paper, we present several different ways to introduce memoization into the nondeterministic DPLL algorithm, to get nondeterministic versions of the beforementioned algorithmic approaches. We then characterize the strength of these nondeterministic algorithms in terms of proof systems. Then we compare these proof systems to each other and to standard proof systems. This gives a good sense of the relative strengths of the various approaches.

1.1 Summary of Results

The standard hierarchy of resolution-like proof systems is: DPLL time (DPLL), which is equivalent to tree-like resolution proofs ; regular resolution, (REG); general resolution (RES); and RES(k), for each $k \geq 2$. This hierarchy is known to be strict under polynomial-simuability; in fact, exponential gaps are possible for each level.

In section 2, we will give three lattices of memoized DPLL algorithms, the basic lattice, the more powerful nondeterministic lattice, and the intermediate "reason" lattice. The basic lattice represent transformations one could make to incorporate memoization to a generic DPLL algorithm. These are the "off-the-shelf" proof systems that correspond to taking your favorite DPLL algorithm and directly adding on memoization of different kinds, without otherwise modifying the original algorithm. This has the advantage of keeping them very close to possibly implementable algorithms.

However, a clever algorithm designer might be able to incorporate memoization in a way that could not be simulated in the basic systems. Our "nondeterministic" systems are designed to represent the "ultimate limits" of these forms of memoization. It would be highly non-trivial to incorporate these features into an existing DPLL algorithm. However, we feel that any algorithm that somehow incorporated memoization into DPLL would probably fall into this lattice somewhere. Thus, bounds on the strength of the nondeterministic lattice are bounds on the potential of the memoization technique. The "reason" proof systems fall somewhere between naive implementation and unbounded cleverness.

The basic lattice of algorithms are denoted $\mathrm{F\ C^T}$ for some $T \subseteq \{W, S, R\}$, which stand for Weakening, Subsumption, and Restriction. The obvious relationships between these are that subsets of operations can be simulated by supersets. The nondeterministic lattice will be denoted $\mathrm{F\ C^T_{nondet}}$. The nondeterministic versions will be at least as strong as the deterministic analog, and is again ordered by subset. The reason lattice will be intermediate between the two. When T is empty, all three variants will coincide.

The basic lattice will be the most natural viewed as an extension of DPLL. However, we shall show in section 3 that the nondeterministic lattice can be characterized as a corresponding set of proof systems, $CC + T$, with each $\mathrm{F\ C^T_{nondet}}$ polynomially equivalent to $CC + T$. Thus, it will be easier to reason about the power of the nondeterministic systems as proof systems than the basic systems.

We then compare these systems to each other and to the standard resolution-like proof systems. For the most interesting systems, our results can be summarized in Figure 1.1.

All of the proof systems can be simulated by Depth 2 Frege proofs. We do not know any weaker standard system that can simulate even basic $\mathrm{F\ C}$

2 Memoization and DPLL: Formula Caching

Memoization means saving previously stored subproblems and using them to prune a back-tracking search. In the satisfiability algorithms we consider, this will mean storing a list of previously refuted formulas and checking

Proceedings of The 18th IEEE Annual Conference on Computational Complexity
ISBN 0-7695-1879-6/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

Proof System	Systems it p-simulates	Systems it cannot p-simulate	Systems that cannot p-simulate it
F C	DPLL	REG (Theorem 4.6)	DPLL (Corollary 2.2)
F C^{WSR}	DPLL	REG (Theorem 4.6)	DPLL (Corollary 2.2)
F C^{WS}_{reason}	REG (Theorem 4.8)	-	Res(k) (Theorem 4.9)
F C^{W}_{nondet}	REG (Theorem 3.2)	-	Res(k) (Theorems 3.12,4.10)
F C^{WS}_{nondet}	REG (Theorem 3.2)	-	Res(k) (Theorems 3.12,4.10)
F $C^{\mathrm{WSR}}_{nondet}$	RES (Theorem 3.6)	-	Res(k) (Theorems 3.11/3.12,4.10)

Figure 1. Relationship of various formula-caching proof systems to other resolution-like proof systems

whether the unsatisfiability of some formula in the list allows us to conclude easily, before branching, that our current formula is unsatisfiable.

A pure back-tracking algorithm usually corresponds to a tree-like proof system, since the recursive refutations are done independently and not reused. Our original intuition was that introducing memoization into a back-tracking algorithm would move from a tree-like proof system to the corresponding DAG-like system. However, the real situation turns out to be somewhat more complicated. There are actually several reasonable ways to introduce memoization into DPLL. None of them seem to be equivalent to DAG-like resolution, and many move beyond resolution.

The basic idea of the simplest memoized version of the DPLL algorithm, is as mentioned above to record the unsatisfiable residual formulas found over the course of the algorithm in a list and before applying recursion to include checking the list to see if F is already known to be unsatisfiable. This yields the following algorithm where L is the cache of residual formulas known to be unsatisfiable.

F C(F, L){
  If F is empty
    Report *satisfiable* and **halt**
  // Check if L **trivially implies that** F **is unsatisfiable**
  If F contains the empty clause Λ *or* F *is in* L
    return
  Else choose a literal x
    Formula-Caching($F|_x$,L)
    Formula-Caching($F|_{\overline{x}}$,L)
    Add F to L}

Running Formula-Caching(F,$\emptyset$) allows one to determine satisfiability of F as before.

While we present F C as a nondeterministic algorithm, one can also view it as a simple transformation for deterministic DPLL algorithms. We simply replace the nondeterministic branching rule with the rule used by the DPLL algorithm. (For memory efficiency, an implementation would probably also add a heuristic to decide whether to cache a restricted formula, or forget it.) This is a straight-forward way of adding memoization to DPLL, similar to other uses of memoization in back-tracking. For example, Robson's maximum independent set algorithm maintains a cache of medium-size subgraphs with known bounds on their maximum independent sets, and checks if the current subgraph is in the cache.

We call the nondeterministic algorithm above, viewed as a proof system, F C. It is obviously at least as powerful as DPLL, since the presence of the cache only prunes branches, never creates them.

In fact we show that it can be exponentially more powerful than DPLL. Ben-Sasson, Impagliazzo, and Wigderson [3], generalizing a construction of Bonet et al. [5], defined certain *graph-pebbling tautologies* $Peb_{G,S,T}$ to separate tree-like from regular resolution. They showed that for suitable choices of DAGs G with $O(n)$ edges and sets S and T the tree-like resolution complexity of these tautologies is $2^{\Omega(n/\log n)}$.

Lemma 2.1. *For any in-degree 2 dag G and sets S and T such that $Peb_{G,S,T}$ is unsatisfiable there is a polynomial-time* F C*refutation of $Peb_{G,S,T}$.*

This is proved in section 4.

Corollary 2.2. *There are formulas refutable in polynomial-time by* F C *that require time $2^{\Omega(n/\log n)}$ to refute by the DPLL algorithm.*

Proceedings of The 18th IEEE Annual Conference on Computational Complexity
ISBN 0-7695-1879-6/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

This shows that F C cannot be simulated by a non-memoized DPLL, but it does not show that F C is strong enough to efficiently simulate all of regular resolution. In fact, we will later show that there are unsatisfiable formulas with small regular resolution refutations that F C cannot efficiently refute.

Once we have the notion that we are checking the formula F against a cache of known unsatisfiable formulas there are other natural related checks that we might do. For example, it may be the case that F contains all the clauses of some formula in the list L and this is nearly as easy to check as whether or not F is in the list. We call such a test a *Weakening* test.

F $\mathrm{C}^{\mathrm{W}}(F, L)\{$
If F is empty
    Report *satisfiable* and **halt**
// Check if L **trivially implies that** F **is unsatisfiable**
If F contains the empty clause Λ *or* F *contains all clauses of some formula in* L
    return
Else choose a literal x
    F $\mathrm{C}^{\mathrm{W}}(F|_x, L)$
    F $\mathrm{C}^{\mathrm{W}}(F|_{\overline{x}}, L)$
    Add F to $L\}$

There is another way that the unsatisfiability of F can trivially follow from that of some formula in L. Given clauses C and D such that C *subsumes* D, i.e. $C \subset D$, we have that C is a stronger constraint than D. Therefore adding a subsumption test to Weakening we obtain an algorithm we denote F C^{WS} where the check whether L trivially implies F asks if there is a formula G in L such that every clause of G contains a clause of F. Again this is nearly as easy to test as membership of F in L.

Weakening and Subsumption are very natural additions to a memoized backtracking algorithm. Among other benefits, they allow a limited amount of "without loss of generality" reason in addition to logical implications of the constraints, because branches dominated by earlier ones get pruned. For example, consider a simple back-tracking algorithm for finding an independent set of size k, branching on a node x with one neighbor y. Without loss of generality, the algorithm should include x in the set. This can be simulated by the Weakening and Subsumption rules. The algorithm first branches on whether $x \in S$, then on whether $y \in S$, exploring the $x \in S$ branch first. The branch $x \in S$ forces $y \notin S$, so the sub-problem is to find an independent set of size $k-1$ in $G-\{x, y\}$. Assume this recursive search fails. The branch $x \notin S$, $y \notin S$ is to find an independent set of size k in $G-\{x, y\}$, a strengthening of the failed branch that gets pruned. The final branch $x \notin S, y \in S$ is to find an independent set of size $k-1$ in $G-\{x, y\}-N(y)$, again a strengthening of the failed branch. So only the branch where $x \in S$ gets recursively explored.

As the above example illustrates, when we have Weakening and Subsumption, the order we explore branches matters. So in additon to a deterministic branching rule, we would need a heuristic to determine the order of branches to construct a deterministic version of F C^{WS}. Otherwise, it is as easily implementable as F C.

Finally, we can observe that given an unsatisfiable formula G, the restricted formula $G|_x$ will also be unsatisfiable. This leads to a more complicated but still polynomial-time triviality test. Furthermore, it is now the case that when we derive the unsatisfiability of F from that of $G \in L$ there may be good reason to add F to L.

F $\mathrm{C}^{\mathrm{WSR}}(F, L)\{$
If F is empty
    Report *satisfiable* and **halt**
// Check if L **trivially implies that** F **is unsatisfiable**
If F contains the empty clause Λ *or there is some* G *in* L *and literal* x *such that every clause of* $G|_x$ *contains a clause of* F
    Add F to L
    return
Else choose a literal x
    F $\mathrm{C}^{\mathrm{WSR}}(F|_x, L)$
    F $\mathrm{C}^{\mathrm{WSR}}(F|_{\overline{x}}, L)$
    Add F to $L\}$

We will see that even this extended test does not suffice to efficiently simulate regular resolution however its new ideas will be useful. One drawback of this test is that some potentially useful information about unsatisfiable formulas may available to be learned but may be lost in the return from a recursive call. For example, if for some formula F the restricted formula $F|_x$ has a small unsatisfiable subformula G and $F|_{\overline{x}}$ has a small unsatisfiable subformula H then F will have a small subformula whose restrictions under x and $\overline{x}$ contain G and H respectively. However, F $\mathrm{C}^{\mathrm{WSR}}$ will learn the formula containing all of F, not just this subformula. In order to take advantage of this kind of information we can augment the algorithm with a return value consisting of a formula giving a "reason" that F is unsatisfiable. We describe this as an extension of F C^{WS}. We will see that this is strong enough to simulate regular resolution efficiently.

F $\mathrm{C}^{\mathrm{WS}}_{reason}(F, L)\{$
If F is empty
    Report *satisfiable* and **halt**
// Check if L **trivially implies that** F **is unsatisfiable**
If F contains the empty clause Λ
    return(Λ)
Else If there is a G in L such that

Proceedings of The 18th IEEE Annual Conference on Computational Complexity
ISBN 0-7695-1879-6/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

every clause of G contains a clause of F
return(G)
Else choose a literal x
$G \leftarrow$F $\mathrm{C}^{\mathrm{W}}_{reason}(F|_x, L)$
$H \leftarrow$F $\mathrm{C}^{\mathrm{W}}_{reason}(F|_{\overline{x}}, L)$
$J \leftarrow \bigwedge_{C \in G}(\overline{x} \vee C) \wedge \bigwedge_{D \in H}(x \vee D)$
Add J to L
Add F to L
return(J)

Given that we are using a cache of unsatisfiable formulas to prove that a formula is unsatisfiable, we may wish to apply the rules such as weakening, subsumption, or restriction a little earlier in the process so that we can be more efficient at generating formulas that we have seen previously to be unsatisfiable. We could for example allow the algorithm to nondeterministically apply weakening at any point in the algorithm. This is a generalization of the usual pure literal rule of DPLL which allows one to remove clauses containing a literal that occurs only positively (or only negatively) in the formula. (Of course, a bad early choice of weakening may suggest satisfiability when that is not the case, but the system will remain be sound for proofs of unsatisfiability.) Similarly, we can define an algorithm F $\mathrm{C}^{\mathrm{WS}}_{nondet}$ that, as well as allowing the removal of clauses, also allows the extension of some number of clauses of F by the addition of extra literals. Finally, we make the manipulation of F more extreme by also allowing the repeated addition of some number of new clauses that contain a literal that does not appear positively or negatively in F. (That is, after we have added some such clauses, removed other clauses, and extended existing clauses, we are allowed to repeat this process.) We denote this system by F $\mathrm{C}^{\mathrm{WSR}}_{nondet}$. (This last rule seems the most unnatural, but it allows one to "forget" a variable branched on; this seems essential to simulating general resolution.) We give a description of F $\mathrm{C}^{\mathrm{WSR}}_{nondet}$; the other algorithms can be obtained by deleting appropriate lines.

F $\mathrm{C}^{\mathrm{WSR}}_{nondet}(F, L)\{$
If F is empty
Report *possibly satisfiable* and **halt**
//Non-deterministic reverse weakening
Remove some subset of clauses of F (possibly none)
//Non-deterministic reverse subsumption
For each clause of F, add some variables (possibly none)
//Non-deterministic reverse restriction
Some number of times,
choose a literal x that does not occur in F,
add $\neg x$ to some subset of clauses of F,
and add a set of clauses all containing x to F.
// Check if L trivially implies that F is unsatisfiable
If F contains the empty clause Λ or F is in L
return
Else choose a literal x
F $\mathrm{C}^{\mathrm{WSR}}_{nondet}(F|_x, L)$
F $\mathrm{C}^{\mathrm{WSR}}_{nondet}(F|_{\overline{x}}, L)$
Add F to $L\}$

If this completes without reporting that F is possibly satisfiable then F will be unsatisfiable. It is immediate that as a refutation system F $\mathrm{C}^{\mathrm{W}}_{nondet}$ is at least as powerful as F C^{W}. It could possibly be more powerful, since the weakened formula is remembered for later use. Similarly, F $\mathrm{C}^{\mathrm{WS}}_{nondet}$ efficiently simulates F C^{WS}, and F $\mathrm{C}^{\mathrm{WSR}}_{nondet}$ efficiently simulates F $\mathrm{C}^{\mathrm{WSR}}$. Furthermore, based on results of the next two sections we can show that essentially without loss of generality all of the modified formulas created during these algorithms can be taken to be sub-formulas of the original input formula being refuted.

It may seem that some of these new systems allowing nondeterministic manipulation of F itself are a little unnatural. However, we shall see that they correspond directly to some extremely natural inference systems for unsatisfiable CNF formulas that we define in the next section. Also, reasoning about such systems covers many algorithms that prune searches based on reasoning that identifies unnecessary constraints , e.g, the pure literal rule or its generalization to autarchs ([12]), or deleting a node of degree 2 or less from a 3-coloring problem. While such weakening only guides the choice of branching variables in a pure back-tracking search, caching the simplified formula may make a more dramatic difference. In fact, we shall see that F $\mathrm{C}^{\mathrm{W}}_{nondet}$, the simplest of these extensions of the basic F C algorithm, is surprisingly powerful; in particular it is capable of refuting formulas that are hard for systems more powerful than resolution.

3 Contradiction caching inference systems

We now define several inference systems for unsatisfiable formulas that are closely related to some of the formula caching algorithms in the previous section. The objects of thes proof systems will be conjunctive normal form (CNF) formulas. CNF formulas will be assumed to be sets of clauses and clauses will be assumed to be sets of literals so the order of clauses and of literals within each clause is immaterial. In the following, x, y, z denote literals which can be variables or their negations, φ, ψ will denote CNF formulas and C, D, E will denote clauses. (A clause also can be viewed as simple case of a CNF formula.) The (unsatisfiable) empty clause will be denoted Λ. Given a formula φ and literal x (or $\overline{x}$), the formula $\varphi|_x$ (respectively $\varphi|_{\overline{x}}$) denotes the simplified CNF formula in which all clauses containing x (respectively $\overline{x}$) have been removed and all clauses containing $\overline{x}$ (respectively x) are shortened by eliminating that literal. More generally given a sequence of literals xyz,

Proceedings of The 18th IEEE Annual Conference on Computational Complexity
ISBN 0-7695-1879-6/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

for example, we write $\varphi|_{xyz} = \varphi|_x|_y|_z$ and for a clause C we identify $\overline{C}$ with the sequence of negations of the literals in C and define $\varphi|_{\overline{C}}$ to be the restriction of φ in which every literal of C has been set to false.

We define several related proof systems for showing that CNF formulas are unsatisfiable based on the following inference rules.

1. *Axiom*: $\vdash \Lambda$

2. *Branching*: $\varphi|_x, \varphi|_{\overline{x}} \vdash \varphi$ where x is any variable and φ is any CNF formula.

3a. *Limited Weakening*: $\Lambda \vdash \Lambda \wedge \psi$ where ψ is any CNF formula.

3. *Weakening*: $\varphi \vdash \varphi \wedge \psi$ where φ and ψ are any CNF formulas.

4. *Subsumption*: $\varphi \wedge C \vdash \varphi \wedge D$ where $D \subseteq C$ are clauses and φ is any CNF formula.

5. *Restriction*: $\varphi \vdash \varphi|_x$ where x is any literal and φ is any CNF formula.

Note: Another way to look at the Branching rule is to begin with any two CNF formulas ψ_1 and ψ_2, neither of which contain a variable x. Let $\psi_{1\cap 2} = \psi_1 \cap \psi_2$ be the set of common clauses of the two formulas and let $\psi_i' = \psi_i - \psi_{1\cap 2}$ for $i = 1, 2$. Then the rule allows us to infer any CNF formula ϕ that for each $C \in \psi_1'$ contains the clause $(C \vee x)$, for each $C \in \psi_2'$ contains the clause $(C \vee \overline{x})$, and for each clause $C \in \psi_{1\cap 2}$ contains some subset of $\{C, (C \vee x), (C \vee \overline{x})\}$ that includes C or both the other two clauses.

A *CC (contradiction caching)* refutation of a CNF formula F is a sequence $\varphi_1, \ldots, \varphi_s = F$ of CNF formulas such that each φ_i for $i > 1$ follows from φ_j, $j < i$ using one of the proof rules (1)-(3a): Axiom, Branching, and Weakening. If in addition we allow some forms of the Weakening rule (3), the Subsumption proof rule (4), or the Restriction proof rule (5) we denote the system by some combination of CC+ some combination of letters W, R, and S.

In addition to these proof systems we will also discuss several other proof systems, DPLL, which is tree-like resolution, REG, which is regular resolution, Res, which is general resolution, and $Res(k)$ for integer $k > 1$ which is an extension of resolution that permits k-DNF formulas instead of clauses.

Given a proof system U for refutating an unsatisfiable CNF formula and let $s_U(F)$ be the minimum length of a refutation CNF formula F in system U.

It is clear that the basic CC proof system can efficiently simulate the execution of any DPLL algorithm and thus can polynomially simulate tree-like resolution proofs. (The Axiom and Limited Weakening together simulate the action at the leaves and the Branch rule simulates the action at the internal nodes of the proof.) Therefore we easily have

Lemma 3.1. *For any unsatisfiable CNF formula F, $s_{CC}(F) \leq 2 \cdot s_{DPLL}(F)$.*

We can also see that CC+W has the full power of regular resolution.

Theorem 3.2. *For any unsatisfiable CNF formula F, $s_{CC+W}(F) \leq 3 \cdot s_{REG}(F)$.*

Proof. Let $C_1, ..., C_s = \Lambda$ be a regular resolution refutation of F. By standard arguments, this refutation yields a directed acyclic graph P with a single root corresponding to clause Λ that forms a read-once branching program, whose leaves are labeled by clauses of F, and whose internal nodes are labeled by variables and whose edges are labeled 0 and 1, that is the d.a.g. analog of the DPLL search tree for an unsatisfied clause. Furthermore, for each clause C in the refutation, on every path π from the root to C the partial assignment defined by π falsifies (every literal of) C.

For each clause C in the refutation, define $V'(C)$ to be the set of variables queried at descendants of the node corresponding to C in P. By the read-once property of P, any variable in $V'(C)$ cannot appear on any path from the root to C in P. For each such clause C, define $F\#_C$ to be the CNF formula consisting of the clauses of $F|_{\overline{C}}$ having variables only in $V'(C)$.

We will show how to derive the sequence $F\#_{C_1}, \ldots, F\#_{C_s} = F\#_\Lambda$ which will be enough to derive F in one more step since F is (at worst) a weakening of $F\#_\Lambda$.

If C is a clause of F, i.e. a leaf in the proof, then $F\#_C$ contains the empty clause and we can derive it in two steps using the Axiom and Weakening.

Suppose $C = (A \vee B)$ is the resolvent of $(A \vee x)$ and $(B \vee \overline{x})$ in the proof and that we already have derived $F\#_{(A\vee x)}$ and $F\#_{(B\vee\overline{x})}$.

Since every literal in $C = (A \vee B)$ appears on every/some path from the root to the node of P corresponding to C, no variable in A or B appears in $V'(A \vee x)$ or in $V'(B \vee \overline{x})$. Therefore $F\#_{(A\vee x)}$ does not contain any variable from B and $F\#_{(B\vee\overline{x})}$ does not contain any variable from A. Therefore $F\#_{(A\vee x)}|_{\overline{B}} = F\#_{(A\vee x)}$.

Now every clause of $F\#_{(A\vee x)} = F\#_{(A\vee x)}|_{\overline{B}}$ is a clause of $F|_{\overline{(A\vee B\vee x)}}$ by definition. Furthermore, since $V'(A \vee x)$ is a subset of $V'(C)$, each clause of $F\#_{(A\vee x)}$ is also entirely defined on $V'(C)$. Therefore by one step of Weakening from $F\#_{(A\vee x)}$ we derive the CNF formula consisting of the clauses of $F|_{\overline{(A\vee B\vee x)}} = (F|_{\overline{C}})|_{\overline{x}}$ that only contain variables in $V'(C)$. Similarly by one step of Weakening from $F\#_{(B\vee\overline{x})}$ we can derive the CNF formula consisting of the clauses of $F|_{\overline{(A\vee B\vee\overline{x})}} = (F|_{\overline{C}})|_x$ that only contain

Proceedings of The 18th IEEE Annual Conference on Computational Complexity
ISBN 0-7695-1879-6/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

the variables in $V'(C)$. Finally, using the Branching rule we derive $F\#_C$. □

Our definition of the complexity of the size of refutations in CC, CC+W, CC+WS, CC+WR, etc. requires a little more justification since we only count the number of lines in our proofs and we are allowing arbitrary CNF formulas for these lines. It is not clear *a priori* that the total number of symbols in these proofs will be polynomial even if the size of the original formula is small and the number of lines is small.

We say that a CNF formula F is a *sub-formula* of another CNF formula G if every clause of F is contained in some clause of G. We say that a CNF refutation system U has the *sub-formula property* if there is some constant c such that for any unsatisfiable formula F there is a refutation of F of size at most $c \cdot s_U(F)$ such that every line is a sub-formula of F. If $c = 1$ then we say that the proof system has the *exact* sub-formula property. Resolution clearly has the exact sub-formula property and it is immediate that CC and CC+W have the exact subformula property since Λ is a sub-formula of any F and for Branching and Weakening the given formulas are sub-formulas of the derived formula. However, the same result is not obvious for CC+WS or CC+WR since both Subsumption and Restriction have the converse property. Nonetheless we have:

Lemma 3.3. *CC, CC+W, CC+WS, CC+WR have the sub-formula property; for CC, CC+W and CC+WS this is the exact sub-formula property.*

Proof. For any CNF formulas F and G let G^F be the formula G with all clauses that are not contained in some clause of F removed. Given any refutation $\varphi_1, \ldots, \varphi_s = F$ of CNF formula F, we claim that $\varphi_1^F, \ldots, \varphi_s^F = F$ is a CC+S refutation of F, where we possibly may repeat a φ_i^F line as some φ_j^F instead of deriving it by an inference rule. The Axiom is immediate so we consider the other cases of the inference rule used:
Weakening: Clearly, an inference $\varphi \dashv \varphi \wedge \psi$ can be replaced by an inference $\varphi^F \dashv \varphi^F \wedge \psi^F$ if ψ^F is non-empty and simply by repeating φ^F otherwise.
Branching: Consider a formula φ derived from $\varphi|_x$ and $\varphi|_{\overline{x}}$ by Branching. By assumption we have already derived $(\varphi|_x)^F = \varphi^F|_x$ ($\varphi|_x$ is a sub-formula of φ by definition) and similarly $(\varphi|_{\overline{x}})^F = \varphi^F|_{\overline{x}}$ and so we can use Branching to derive φ^F. (Observe that if variable x does not appear in F then no inference is required.)
Subsumption: Suppose that $\varphi \wedge D \dashv \varphi \wedge C$ is the inference where $C \subset D$. If D is contained in a clause of F we have $(\varphi \wedge D)^F = \varphi^F \wedge D \dashv \varphi^F \wedge C = (\varphi \wedge C)^F$ using Subsumption. Otherwise, $(\varphi \wedge D)^F = \varphi^F \dashv \varphi^F \wedge C^F = (\varphi \wedge C)^F$ using Weakening if C is contained in a clause of F or simply copying φ^F if C is not.
Restriction: Suppose that $\varphi \dashv \varphi|_x$ is the inference where x is a literal. By definition of restriction, each clause of $\varphi^F|_x$ is contained in a clause of φ^F and thus is contained in a clause of F. Furthermore each such clause is also a clause of $\varphi|_x$ so $(\varphi|_x)^F$ contains all clauses of $\varphi^F|_x$. In particular, this means that we can first derive $\varphi^F|_x$ by Restriction from φ^F and then derive $(\varphi|_x)^F$ by Weakening from $\varphi^F|_x$. This could possibly at most double the number of steps in a CC+R refutation so the sub-formula property for CC+R holds with $c = 2$. □

Therefore, given a CNF formula F with bounded clause size as input and a polynomial size CC, CC+S, or CC+S refutation of F, there is one with at most a polynomial number of symbols.

The following shows that the addition of the Restriction rule is sufficient to efficiently simulate the Subsumption rule which justifies eliminating separate consideration of CC+WRS.

Lemma 3.4. *If $C \subset D$ are clauses then there is a $CC + WR$ derivation of $\varphi \wedge C$ from $\varphi \wedge D$ of length $4|D - C|$.*

Proof. We prove this in the case that $D = C \vee x$. The result follows by repeating this process to reduce D to C one literal at a time. First apply Weakening to $\varphi \wedge D$ to yield $\varphi \wedge D \wedge C$. Then apply Restriction to yield $(\varphi \wedge D \wedge C)|_x = (\varphi \wedge C)|_x$ since $D|_x$ is satisfied and eliminated from the formula. We can also apply Restriction to $\varphi \wedge D \wedge C$ to derive $(\varphi \wedge D \wedge C)|_{\overline{x}} = (\varphi \wedge C)|_{\overline{x}}$ since $D|_{\overline{x}} = C = C|_{\overline{x}}$ and restriction distributes over conjunctions. (Here we use our view of CNF formulas as sets of clauses to eliminate duplicates.) Applying Branching to $(\varphi \wedge C)|_x$ and $(\varphi \wedge C)|_{\overline{x}}$ yields $\varphi \wedge C$ in a total of 4 steps as required. □

Corollary 3.5. *If F is an unsatisfiable CNF formula in n variables then $s_{CC+WR}(F) \leq 4n \cdot s_{CC+WS}(F)$.*

Theorem 3.6. *For any unsatisfiable CNF formula F, $s_{CC+R}(F) \leq n \cdot s_{Res}(F)$.*

Proof. Given a resolution derivation $\Pi = C_1, \ldots, C_s$ from a CNF formula F we show how to produce a CC+R derivation Π^* of length at most ns that contains $F|_{\overline{C_1}}, \ldots, F|_{\overline{C_s}}$. Observe that for $C = \Lambda$, $F|_{\overline{C}} = F$ and this implies the lemma since Λ is the final clause in a resolution refutation of F.

We have two cases to consider for our induction depending on the rule used in Π to derive the last clause.
(a): If C is a clause of F then $F|_{\overline{C}}$ contains the empty clause Λ. Therefore we can begin with $\dashv \Lambda$ and derive $F|_{\overline{C}}$ by one step of Limited Weakening to add all the remaining clauses of $F|_{\overline{C}}$.
(b): If C follows via resolving clauses $A \vee x$ and $B \vee \overline{x}$ (where x does not appear in A or B and $C = A \vee B$)

Proceedings of The 18th IEEE Annual Conference on Computational Complexity
ISBN 0-7695-1879-6/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

then by induction we have a CC+R refutation of length at most $n(s-1)$ that derives $G = F|_{\overline{A \vee x}} = F|_{\overline{A}\,\overline{x}}$ and $H = F|_{\overline{B \vee \overline{x}}} = F|_{\overline{B}\,x}$. We repeatedly apply Restriction using the elements of $\overline{B} - \overline{A}$ to derive $G|_{\overline{B}} = F|_{\overline{A \vee B \vee x}} = (F|_{\overline{A \vee B}})|_{\overline{x}}$ similarly we apply repeated Restriction using the elements of $\overline{A} - \overline{B}$ to derive $H|_{\overline{A}} = F|_{\overline{A \vee B \vee \overline{x}}} = (F|_{\overline{A \vee B}})|_{x}$. Now we can apply the Branching rule with $\varphi = F|_{\overline{C}} = F|_{\overline{A \vee B}}$ to obtain $F|_{\overline{C}}$. Since $A \vee B$ contains at most $n-1$ distinct variables there were at most $n-1$ Restriction steps and one Branching step to yield total refutation size of at most ns. □

Given a CNF formula F in variables $\{x_1, \ldots, x_n\}$ and an integer k, we can define a new formula $F^{(\wedge k)}$ in variables $\{z_{i,j} : i \in [n], j \in [k]\}$ by replacing every clause $C \in F$ by a conjunction of clauses corresponding to C with the substitution $x_i \leftarrow z_{i,1} \wedge \cdots \wedge z_{i,k}$ and distributing the result to form clauses. That is, if P and N are the indices of variables occurring positively and negatively in C then C is replaced by $\bigwedge_{(j_1,\ldots j_{|P|}) \in [k]^{|P|}} \left(\bigvee_{i \in P} z_{i,j_i} \vee \bigvee_{i \in N} \bigvee_{j \in [k]} \overline{z_{i,j}} \right)$. Note that if C has size b then it is replaced by at most k^b clauses each of size at most bk.

Lemma 3.7. *If U is any of the systems CC, CC+W, CC+R, CC+WR then for any unsatisfiable CNF formula F, $s_U(F^{(\wedge k)}) \leq k \cdot s_U(F)$.*

Proof. Given a formula caching refutation Π of F of length s we show how to derive all clauses of $\Pi^{(\wedge k)}$ using at most sk inference steps. Consider the rules used in the course of the refutation Π.
(1) Clearly $\Lambda^{(\wedge k)} = \Lambda$.
(2) If the inference rule in Π is Weakening $\varphi \dashv \varphi \wedge \psi$ and we have already $\varphi^{(\wedge k)}$ then we get $\varphi^{(\wedge k)} \dashv \varphi^{(\wedge k)} \wedge \psi^{(\wedge k)}$ also by Weakening and the latter is $(\varphi \wedge \psi)^{(\wedge k)}$ by definition. Further, if the Weakening inference in Π is Limited then the same will hold true in $\Pi^{(\wedge k)}$.
For the Restriction and Branching rules we will drop the variable subscript i for notational convenience.
(3) If the inference rule in Π is Restriction then we have two cases depending on whether the restriction literal is positive or negative. (i) If the rule applied is $\varphi \dashv \varphi|_x$ and x is positive (i.e. the substitution is $x = z_1 \wedge \ldots \wedge z_k$) then given $\varphi^{(\wedge k)}$ we apply a sequence of k restrictions:

$$\varphi^{(\wedge k)} \dashv \varphi^{(\wedge k)}|_{z_1} \dashv \varphi^{(\wedge k)}|_{z_1 z_2} \dashv \cdots \dashv \varphi^{(\wedge k)}|_{z_1 z_2 \ldots z_k}$$

and this is precisely $(\varphi|_x)^{(\wedge k)}$. (ii) If the rule applied is $\varphi \dashv \varphi|_{\overline{x}}$ the substitution is $x = z_1 \wedge \ldots \wedge z_k$ then given $\varphi^{(\wedge k)}$ we apply the single restriction $\varphi^{(\wedge k)} \dashv \varphi^{(\wedge k)}|_{\overline{z_j}}$ (for any $j \in [k]$) and claim that the latter is $(\varphi|_{\overline{x}})^{(\wedge k)}$. For any clause C of φ containing $\overline{x}$, every clause of $C^{(\wedge k)}$ will contain $\overline{z_j}$ and thus be eliminated. For any clause C of φ containing x, the set of clauses of $C^{(\wedge k)}$ containing z_j will be of precisely the right form when shortened.
(4) Suppose that clause $\varphi \in \Pi$ follows from $\varphi|_x$ and $\varphi|_{\overline{x}}$ using Branching and the substitution is $x = z_1 \wedge \ldots \wedge z_k$. For $j \in [k]$, let $F_j = \varphi^{(\wedge k)}|_{z_1 \ldots z_j}$ and $G_j = \varphi^{(\wedge k)}|_{z_1 \ldots z_{j-1}\overline{z_j}}$. As above, $F_k = \varphi^{(\wedge k)}|_{z_1 \ldots z_k} = (\varphi|_x)^{(\wedge k)}$. Furthermore, as above $G = (\varphi|_{\overline{x}})^{(\wedge k)} = \varphi^{(\wedge k)}|_{\overline{z_j}}$ for any $j \in [k]$. Since G contains no occurences of $z_1, \ldots, z_k$ for $j \in [k]$ we can also write $G = G|_{z_1 \ldots z_{j-1}} = \varphi^{(\wedge k)}|_{\overline{z_j} z_1 \ldots z_{j-1}} = G_j$. We wish to derive $\varphi^{(\wedge k)}$ from F_k and $G = G_1 = \ldots = G_k$. To do this we apply the Branching rule k times, deriving F_{k-1} from F_k and G_k using variable z_k, F_{k-2} from F_{k-1} and F_{k-1} using variable x_{k-1}, etc. until finally we obtain the desired clause using the branching rule applied to F_1 and G_1. □

We can state an even more general result that allows arbitrary disjoint substitutions of variables. For F a formula in $\{x_1, \ldots, x_n\}$ and $\sigma = (\sigma_1, \ldots, \sigma_n)$ be a sequence of Booleans functions with $\sigma_i : \{0,1\}^{k_i} \to \{0,1\}$. We can define a new formula $F^{(\sigma)}$ in variables $\{z_j^i \mid i \in [n],\ j \in [k_i]\}$ in which each σ_i is represented optimally in CNF and DNF (depending on whether x_i appears positively or negatively in a clause of F) and the result expanded canonically into clauses in some fashion. It is not hard to show that the number of lines in a proof of $F^{(\sigma)}$ is at most the number of lines in the original proof multiplied the maximum over all i of the optimal read-once branching program size of σ_i.

Note: We did not include Subsumption in Lemma 3.7 but it is also possible to simulate subsumption proofs with a somewhat weaker bound. The issue is that for a single clause D, $D^{(\wedge k)}$ may produce a large number of clauses so the reduction of these clauses to the clauses of $C^{(\wedge k)}$ for $C \subset D$ may involve a large number of individual Subsumption inferences. (In total, by the subformula property the number of such clauses is bounded in terms of the size of the target formula so if the original proof is polynomial size the new proof will be as well.)

Corollary 3.8. *Let $\{F\}$ be a family of unsatisfiable CNF formulas.*

- *If $\{F\}$ has polynomial-size DPLL proofs then $\{F^{(\wedge k)}\}$ has polynomial-size CC proofs.*
- *If $\{F\}$ has polynomial-size regular resolution proofs then $\{F^{(\wedge k)}\}$ has polynomial-size CC+W proofs.*
- *If $\{F\}$ has polynomial-size resolution proofs then $\{F^{(\wedge k)}\}$ has polynomial-size CC+R proofs.*

We first use this corollary to show that CC can efficiently prove the pebbling tautologies mentioned in the previous section.

Proceedings of The 18th IEEE Annual Conference on Computational Complexity
ISBN 0-7695-1879-6/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

Lemma 3.9. *Given a directed acyclic graph G of in-degree 2 with m edges and subsets S and T of its vertices, if $Peb_{G,S,T}$ is unsatisfiable then $s_{CC}(Peb_{G,S,T}) = O(m)$.*

Proof. (Sketch) The contradiction $Peb_{G,S,T}$ has two variables per vertex v, z_v^0 and z_v^1 and the statement that v can be pebbled is represented by the clause $(z_v^1 \vee z_v^1)$. The tautology represents the statement that (1) all nodes in S can be pebbled (2) if both of a nodes' predecessors can be pebbled then so can the node itself (3) no node in T can be pebbled. We observe that this formula is derived by disjoint substitution $x_v = z_v^0 \vee z_v^1$ from a simpler formula $P_{G,S,T}$ where we represent the ability to pebble node v by a single variable x_v. That is $Peb_{G,S,T} = P_{G,S,T}^{(\vee 2)}$. The formula $P_{G,S,T}$ can be proved unsatisfiable in a linear number of steps by unit propagation following a topological sort from S to T. Therefore it follows immediately in CC. By the closure property of CC under disjoint substitution (this is $\vee$ instead of $\wedge$ but the same result follows either by negating variables and using the result for $(\wedge k)$ or directly from the read-once branching program generalization of Lemma 3.7) $Peb_{G,S,T}$ also has a linear size proof in CC. □

We now use Corollary 3.8 together with results of [15] to separate the CC+R proof system from $Res(k)$ for any constant k.

In order to separate $Res(k+1)$ from $Res(k)$, Segerlind, Buss, and Impagliazzo [15] define an unsatisfiable CNF formula $GOP(G)$ for any undirected graph G (describing for the *graph ordering principle* on G) and prove that, although $GOP(G)$ always has polynomial-size resolution refutations, there is an infinite family of graphs G such that for any constant k, $GOP(G)^{(\wedge k+1)}$ requires exponential-size $Res(k)$ refutations. More precisely, given $G = (V, E)$ with $|V| = n$ define $n(n-1)$ variables $x_{u,v}$ for all $u \neq v$ which are intended to represent a transitive, irreflexive, antisymmetric relation on the vertices of G. Thus we have clauses $(\overline{x_{u,v}} \vee \overline{x_{v,u}})$ for antisymmetry and $(\overline{x_{u,v}} \vee \overline{x_{v,w}} \vee x_{u,w})$ for transitivity for all distinct $u, v, w \in V$. The graph ordering principle for G states that any such relation must have an element that is locally minimal in G. Thus to represent the negation of this principle for each $v \in V$ we add the clause $\bigvee_{(u,v)\in E} x_{u,v}$.

Theorem 3.10 ([15]). *For any positive integer k, there are constants $c > 0$ and $\epsilon_k > 0$, and an infinite family of graphs $\{G\}$ such that $GOP(G)$ has resolution refutations of size $O(n^c)$ where $n = n(G)$, $GOP(G)^{(\wedge k)}$ has $Res(k)$ refutations of size $O(n^c)$, $GOP(G)^{(\wedge k+1)}$ has size $O(n^c)$ but requires $Res(k)$ refutations of size $2^{\Omega(n^{\epsilon_k})}$.*

Theorem 3.11. *For any positive integer k, there are formulas with polynomial-size CC+R refutations that require exponential-size $Res(k)$ refutations.*

Proof. Consider the family of polynomial-size formulas $GOP(G)^{(\wedge k+1)}$ given by Theorem 3.10. Since the formulas $GOP(G)$ have polynomial size resolution refutations, by Theorem 3.6 and Lemma 3.7 the formulas $GOP(G)^{(\wedge k+1)}$ have polynomial-size CC+R refutations. On the other hand by Theorem 3.10 they require exponential-size $Res(k)$ refutations. □

We can observe that not only does $GOP(G)$ have polynomial-size resolution refutations but these refutations are in fact regular resolution refutations to derive the following:

Theorem 3.12. *For any positive integer k, there are formulas with polynomial-size CC+W refutations that require exponential-size $Res(k)$ refutations.*

4 Formula caching search and contradiction caching inference systems

Theorem 4.1. F C *and CC are polynomially equivalent refutation systems.*

Proof. To show that CC can efficiently simulate F C, observe that in an execution of F C, each recursive call adds precisely one formula to L and each such formula F is derivable either because it contains the empty clause Λ and therefore follows from the Axiom of CC via one step of Limited Weakening, or as the result of $F|_x$ and $F|_{\overline{x}}$ being in L and therefore follows via one Branching step.

For the reverse direction, let F be the goal formula for CC (and input for F C). For simplicity we will take the result of every Limited Weakening rule as an additional axiom in the CC proof so we have a proof whose only inference rule is Branching. By the sub-formula property of CC, w.l.o.g. every formula in the CC proof is a sub-formula of F and thus every subformula is a restriction of F by some partial assignment ρ and each non-axiom node is associated with a variable involved in the branching. Draw the DAG of inferences in this simplified CC proof directed from the goal formula back to the leaves. The F C algorithm will follow a depth-first traversal of this proof and choose its branch variable according to the variables labeling the nodes in the DAG it encounters. Whenever it traverses a forward edge or a cross edge with respect to the DFS tree, by construction the associated formula will already be in the cache L. The number of recursive calls is equal to the number of edges in this proof DAG. □

Corollary 4.2. *For any in-degree 2 pebbling graph G and sets S and T there is a polynomial-time* F C *refutation of $Peb_{G,S,T}$ if this is contradictory.*

Proceedings of The 18th IEEE Annual Conference on Computational Complexity
ISBN 0-7695-1879-6/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

However, this simulation does not extend to all of regular resolution. In particular, consider the family of GT formulas, defined in [6], which separate regular resolution from tree resolution. These were the inspiration for the GOP formulas defined above. In particular for any n the GT_n formula includes all clauses of $GOP(K_n)$ where K_n is the complete graph on $V = \{1, ..., n\}$ together with *totality* clauses $(x_{i,j} \vee x_{j,i})$ for each $i \neq j$. As shown in [6], like the formulas $Peb_{G,S,T}$ above, these formulas have polynomial-size regular resolution refutations but require exponential-size tree resolution refutations.

Write $G \dashv_{WS} H$ iff H follows from G solely via Weakening and Subsumption. We observe the following simple properties of $\dashv_{WS}$.

Proposition 4.3. *(a) $\dashv_{WS}$ is transitive, i.e. if $F \dashv_{WS} G$ and $G \dashv_{WS} H$ then $F \dashv_{WS} H$.*

(b) If $F \dashv_{WS} H$ and $G \dashv_{WS} H$ then $F \wedge G \dashv_{WS} H$.

(c) For any literal x, if $G \dashv_{WS} H$ then $G|_x \dashv_{WS} H|_x$.

(d) For any literal x, if $G \dashv_{WS} H|_x$ then $G' \dashv_{WS} H$ where $G' = \bigwedge_{C \in G}(\overline{x} \vee C)$.

Proof. Parts (a) and (b) follow immediately from the definition. Suppose that x is a literal and $G \dashv_{WS} H$. If $C \in G|_x$ then neither x nor $\overline{x}$ appears in C and either C or $(C \vee \overline{x})$ appears in G. If $C \in G$ then there is some $D \in H$ with $D \subseteq C$ $D \in H|_x$. If $(C \vee \overline{x}) \in G$ then there is some $D \in H$ with $D \subseteq (C \vee \overline{x})$ and thus $D|_x \subseteq C$ and $D|_x \in H|_x$. Thus (c) follows. For part (d), consider a clause $(\overline{x} \vee C)$ in G'. Since $C \in G$ there is a $D \in H|_x$ with $D \subseteq C$. Then $D \subseteq (\overline{x} \vee D) \subseteq (\overline{x} \vee C)$, and since either $D \in H$ or $(\overline{x} \vee D) \in H$ there is clause of H contained in $(\overline{x} \vee C)$. Thus $G' \dashv_{WS} H$. □

Let $unitprop(H)$ be the formula obtained from H after applying unit propagations to H.

Lemma 4.4. *If $G \dashv_{WS} H$ then there is a restriction π such that $G|_\pi \dashv_{WS} unitprop(H)$ and $G|_\pi$ has no unit clauses.*

Proof. Assume that $\Lambda \notin unitprop(H)$ for otherwise the lemma follows immediately with $G|_\pi = unitprop(G)$. Otherwise let π be the set of assignments that are made during unit propagation on H. By the proposition above we have $G|_\pi \dashv_{WS} unitprop(H)$. If x is a unit clause in $G|_\pi$ then, since $\Lambda \notin unitprop(H)$, $unitprop(H)$ must contain x as a unit clause which is a contradiction. □

We will be interested in formulas $G = GT_n|_\sigma$ and $H = GT_n|_\tau$ such that $G \dashv_{WS} H$. Using Lemma 4.4 we will only need to study this when G and H have no unit clauses and H does not contain the empty clause.

Observe that if $G = GT_n|_\sigma$ has no unit clauses and does not contain the empty clause then σ must be transitively closed and so we can identify σ with a partial order $<_\sigma$ on V.

Given a partial order $<_\sigma$ on V define

- $\sigma^i = \{j \in V \mid i <_\sigma j\}$.
- $minimal(\sigma) = \{i \mid \nexists j \in V, j <_\sigma i\}$,
- $tops(\sigma) = \{k \mid \forall i \in minimal(\sigma), i <_\sigma k\}$, and
- $prune(\sigma)$ to be $<_\sigma$ restricted to $V - tops(\sigma)$.

Lemma 4.5. *If $G = GT_n|_\sigma \dashv_{WS} H = GT_n|_\tau$ and G and H do not contain Λ or any unit clause then $prune(\sigma) = prune(\tau)$.*

Proof. For any pair $j, k \in V$, if j and k are incomparable in $<_\sigma$ then G contains the clause $(x_{j,k} \bigvee x_{k,j})$ which must also appear in H since H does not contain Λ or a unit clause. Therefore j and k are incomparable in $<_\tau$.

Since G does not contain Λ or a unit clause, G contains a non-minimality clause $C_i = \bigvee_{j \in V - \sigma^i} x_{j,i}$ of size at least 2 for each $i \in minimal(\sigma)$. Therefore H must contain a clause $D_i \subseteq C_i$ with at least two positive literals whose last coordinate is i. This can only be the non-minimality clause $D_i = \bigvee_{j \in V - \tau^i} x_{j,i}$ and thus $i \in minimal(\tau)$ and $\sigma^i \subseteq \tau^i$. Since any $j \notin \sigma^i$ is incomparable to i in $<_\sigma$, it must be incomparable to i in $<_\tau$ so $j \notin \tau^i$. Therefore $minimal(\sigma) = minimal(\tau)$ and each such minimal element has $\sigma^i = \tau^i$. Furthermore by definition $tops(\sigma) = tops(\tau)$.

If $j <_\sigma k$ and $j, k \in V - tops(\sigma)$ then there is some $i \in minimal(\sigma)$ such that $i \not<_\sigma k$. Therefore i is incomparable to both j and k in $<_\sigma$. Therefore G will contain two clauses of size 2 that are the restrictions of the transitivity clauses for the triple (i, j, k), namely $(\neg x_{i,j} \vee x_{i,k})$ and $(\neg x_{k,i} \vee x_{j,i})$. These clauses must also appear in H and the only possible sources for them are the same transitivity clauses in GT_n. Therefore $j <_\tau k$.

Therefore for all $j, k \in V - tops(\sigma) = V - tops(\tau)$, $j <_\sigma k$ if and only if $j <_\tau k$ and thus $prune(\sigma) = prune(\tau)$. □

Theorem 4.6. *Any* $\mathrm{F\,C^{WSR}}$ *refutation of GT_n requires at least 2^{n-2} nodes.*

Proof. We show that there are at least 2^{n-2} distinct residual formulas in any such refutation, with the property that no two of them can be inferred using Weakening, Subsumption and Restriction from the same residual subformula.

For any restriction ρ such that $GT_n|_\rho$ does not infer Λ via unit propagation, the transitive closure, ρ^*, of the relation defined by ρ forms a partial order $<_{\rho^*}$. Call a branch point in an $\mathrm{F\,C^{WSR}}$ execution *novel* if (1) the residual formula $GT_n|_\rho$ at the branch point does not infer Λ by unit propagation and (2) it branches on a variable $x_{i,j}$ such that i and

Proceedings of The 18th IEEE Annual Conference on Computational Complexity
ISBN 0-7695-1879-6/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

j are in different connected components of the Hasse diagram associated with $<_{\rho^*}$. Observe that if only $n-2$ novel branch points have been made on a path then $tops(\sigma^*) = \emptyset$. Furthermore, every consistent branch can be extended until it contains at least $n-2$ novel branch points and the restrictions ρ defining these branches are inconsistent with each other. Therefore there are at least 2^{n-2} of them at the novelty level $n-2$ and their transitive closures σ all have $prune(\sigma) =<_\sigma$ and disagree about the relative order of some pair of elements.

Let $H = GT_n|_\rho$ be the residual formula at a novel branch point and assume that $G = GT_n|_\pi$ infers H using Weakening, Subsumption, and Restriction. Therefore $G' \dashv_{WS} H$ for $G' = G$ or $G|_x$ for some literal x. Applying Lemma 4.4 we obtain a formula $G'' = G'|_{\pi'}$ for the restriction π' such that $unitprop(H) = H|_{\pi'}$, $G'' \dashv_{WS} unitprop(H)$, and G'' does not have an empty or unit clause. Let σ be the restriction that is the combination of π, π', and x and let τ be the restriction that is the combination of ρ and π'. By construction σ and τ correspond to partial orders on $\{1, \ldots, n\}$. By Lemma 4.5 we must have $<_\tau = prune(\tau) = prune(\sigma) =<_\sigma$.

Now if G is added to L before H in the execution of F $\mathrm{C}^{\mathrm{WSR}}$ then either G is in the subtree below H or there is some variable $x_{i,j}$ on which π and ρ disagree. If the latter were to occur, the corresponding extended restrictions σ and τ would retain this disagreement, and $<_\sigma$ and $<_\tau$ would disagree about the relative order of i and j. This would contradict the requirement that $<_\tau =<_\sigma$. Therefore any such clause G would have to be in the subtree below H. Since these subtees are disjoint for every pair H and H' of our set of clauses at novelty level $n-2$, the theorem follows. □

Corollary 4.7. *CC does not polynomially simulate regular resolution.*

Thus even the strongest of the basic formula caching systems is not strong enough to efficiently simulate regular resolution. In fact, these systems cannot efficiently simulate the *ordered* regular resolution method defined in the original paper of Davis and Putnam [8] since, as shown in [6], the formulas GT_n are provable in ordered regular resolution.

However, when we augment formula caching by having it return the reason for unsatisfiability as well as the mere fact of unsatisfiability, we can efficiently simulate regular resolution (and much more).

Theorem 4.8. F $\mathrm{C}^{\mathrm{WS}}_{reason}$ *linearly simulates regular resolution.*

Proof. We follow the general pattern of the proof of Theorem 3.2. See the proof of that lemma for the notation we use here. The sequence of branches followed by the F $\mathrm{C}^{\mathrm{WS}}_{reason}$ algorithm will follow a depth-first search of the regular resolution d.a.g. refuting F. We prove by induction that if a node in this d.a.g. labelled by clause C is reached for the first time then F $\mathrm{C}^{\mathrm{WS}}_{reason}(F|_\pi, L)$ returns a formula J whose variables are in $V'(C)$ and such that $J \dashv_{WS} F\#_C$. (On subsequent visits, J will suffice to prevent the algorithm from descending below this node.)

The induction starts at the leaves. If C labels a leaf in the proof then $C \in F$, $F\#_C = \Lambda$, and the algorithm returns Λ. If $C = (A \vee B)$ is the resolvent of $(A \vee x)$ and $(B \vee \overline{x})$ which label its children then by the induction hypothesis we have that F $\mathrm{C}^{\mathrm{WS}}_{reason}$ has returned a formula H defined on $V'(A \vee x)$ such that $H \dashv_{WS} F\#_{(A \vee x)}$ and G defined on $V'(B \vee \overline{x})$ such that $G \dashv_{WS} F\#_{(B \vee \overline{x})}$.

By the argument in the proof of Lemma 3.2, $F\#_{(B \vee \overline{x})} \dashv_{WS} (F|_{\overline{C}})|_x$ and thus by transitivity $G \dashv_{WS} (F|_{\overline{C}})|_x$. Similarly, $F\#_{(A \vee x)} \dashv_{WS} (F|_{\overline{C}})|_{\overline{x}}$ and thus $H \dashv_{WS} (F|_{\overline{C}})|_{\overline{x}}$.

Applying Proposition 4.3(d), we have $\bigwedge_{D \in G}(\overline{x} \vee D) \dashv_{WS} F|_{\overline{C}}$ and $\bigwedge_{E \in H}(x \vee E) \dashv_{WS} F|_{\overline{C}}$. Proposition 4.3(b) then implies that the clause J returned by F $\mathrm{C}^{\mathrm{WS}}_{reason}$ satisfies

$$J = \bigwedge_{D \in G} (\overline{x} \vee D) \wedge \bigwedge_{E \in H} (x \vee E) \dashv_{WS} F|_{\overline{C}}.$$

Now since $V'(A \vee x)$ and $V'(B \vee \overline{x})$ are subsets of $V'(C)$ and $x \in V'(C)$, every variable in J is a subset of $V'(C)$. The inference $J \dashv_{WS} F|_{\overline{C}}$ depends only on the clauses of $F|_{\overline{C}}$ that contain variables appearing in J so we can remove all clauses of $F|_{\overline{C}}$ that have variables outside $V'(C)$ while maintaining the inference. Thus we have $J \dashv_{WS} F\#_C$ as required. The theorem follows immediately. □

Thus, in particular, for any graph G F $\mathrm{C}^{\mathrm{WS}}_{reason}$ can efficiently refute $GOP(G)$. By a similar idea to that used in Lemma 3.7, given a refutation of a formula F in F $\mathrm{C}^{\mathrm{WS}}_{reason}$, we can obtain an F $\mathrm{C}^{\mathrm{WS}}_{reason}$ refutation of $F^{(\wedge k)}$ of size at most $O(k)$ times that of F by replacing each branch on a variable x_i of F by a sequence of branches on the variables $z_{i,j}$ such that x_i is replaced by $\bigwedge_{j=1}^{k} z_{i,j}$. Applying this to the $GOP(G)^{(\wedge k+1)}$ formulas defined in Theorem 3.10, we obtain:

Theorem 4.9. *For any positive integer k, there are formulas with polynomial-size* F $\mathrm{C}^{\mathrm{WS}}_{reason}$ *refutations that require exponential-size $Res(k)$ refutations.*

We conclude by showing the equivalences between the more powerful nondeterministic formula caching systems and the contradiction caching proof systems.

Theorem 4.10. *The following pairs of refutation systems are polynomially equivalent:*

- *CC+W and* F $\mathrm{C}^{\mathrm{W}}_{nondet}$

IEEE COMPUTER SOCIETY

- *CC+WS and* $\mathrm{F\,C}^{\mathrm{WS}}_{nondet}$
- *CC+WSR and* $\mathrm{F\,C}^{\mathrm{WSR}}_{nondet}$ *(which is polynomially equivalent to* $\mathrm{F\,C}^{\mathrm{WR}}_{nondet}$*)*

Proof. In each case one can observe that the CC proof rules of Weakening, Subsumption and Restriction can reverse the result of the corresponding nondeterministic tinkering with the residual formula F in the $\mathrm{F\,C}$ system. Thus it is easy to see that the systems involving CC can efficiently simulate the corresponding systems involving $\mathrm{F\,C}$.

The reverse simulation is a little trickier. As in the previous theorem we prune the proof DAG involving CC by taking the results of Limited Weakening as leaves. We again follow a DFS of the proof DAG directed from the goal formula to the leaves. Observe that in this DAG all nodes have out-degree 1 except the Branching nodes. Whenever we reach the result of a Branching inference we choose the associated variable and make the recursive call as we would in plain CC. Otherwise we observe that we can follow the path of out-degree 1 inferences back either to an axiom or to a Branching inference. It is easy to check that the nondeterministic tinkering with F allowed in the $\mathrm{F\,C}$-based system can simulate this path. That is, Weakening and Subsumption and Restriction can be simulated since each of the manipulations of F allowed in the $\mathrm{F\,C}$ extension permits one to reverse the corresponding inference rule. □

References

[1] F. Bacchus, S. Dalmao, and T. Pitassi. DPLL with Caching: A new algorithm for #SAT and Bayesian inference. Technical Report TR03-003, Electronic Colloquium in Computation Complexity, `http://www.eccc.uni-trier.de/eccc/`, 2003.

[2] Paul Beame, Henry Kautz, and Ashish Sabharwal. On the power of clause learning. In *Proceedings of the 18th IJCAI*, 2003. To appear.

[3] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near-optimal separation of treelike and general resolution. Technical Report TR00-005, Electronic Colloquium in Computation Complexity, `http://www.eccc.uni-trier.de/eccc/`, 2000.

[4] E. Ben-Sasson and A. Wigderson. Short proofs are narrow – resolution made simple. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 517–526, Atlanta, GA, May 1999.

[5] M. L. Bonet, J. L. Esteban, N. Galesi, and J. Johansen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM Journal on Computing*, 30(5):1462–1484, 2000.

[6] M. L. Bonet and N. Galesi. A study of proof search algorithms for resolution and polynomial calculus. In *Proceedings 40th Annual Symposium on Foundations of Computer Science*, New York,NY, October 1999. IEEE.

[7] V. Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, 1988.

[8] M. Davis and H. Putnam. A computing procedure for quantification theory. *Communications of the ACM*, 7:201–215, 1960.

[9] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–305, 1985.

[10] S. M. Majercik and M. L. Littman. Using caching to solve larger probabilistic planning problems. In *Proceedings of the 14th AAAI*, pages 954–959, 1998.

[11] João P. Marques-Silva and Karem A. Sakallah. Grasp – a new search algorithm for satisfiability. In *Proceedings of the International Conference on Computer-Aided Design*, pages 220–227, San Jose, CA, November 1996. ACM/IEEE.

[12] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, pages 287–295, 1985.

[13] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, pages 530–535, Las Vegas, NV, June 2001. ACM/IEEE.

[14] J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, 1986.

[15] N. Segerlind, S. Buss, and R. Impagliazzo. A switching lemma for small restrictions and lower bounds for k-DNF resolution. In *Proceedings 43nd Annual Symposium on Foundations of Computer Science*, Vancouver, BC, November 2002. IEEE.

[16] Hantao Zhang. Sato: An efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction, LNAI*, volume 1249, pages 272–275, July 1997.

[17] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the International Conference on Computer-Aided Design*, pages 279–285, San Jose, CA, November 2001. ACM/IEEE.

IEEE COMPUTER SOCIETY