Noname manuscript No.
(will be inserted by the editor)

Simulating Strong Practical Proof Systems with Extended Resolution

Benjamin Kiesl · Adrián Rebola-Pardo · Marijn J.H. Heule · Armin Biere

the date of receipt and acceptance should be inserted later

Abstract Proof systems for propositional logic provide the basis for decision procedures that determine the satisfiability status of logical formulas. While the well-known proof system of extended resolution—introduced by Tseitin in the sixties—allows for the compact representation of proofs, modern SAT solvers (i.e., tools for deciding propositional logic) are based on different proof systems that capture practical solving techniques in an elegant way. The most popular of these proof systems is likely DRAT, which is considered the de-facto standard in SAT solving. Moreover, just recently, the proof system DPR has been proposed as a generalization of DRAT that allows for short proofs without the need of new variables. Since every extended-resolution proof can be regarded as a DRAT proof and since every DRAT proof is also a DPR proof, it was clear that both DRAT and DPR generalize extended resolution. In this paper, we show that—from the viewpoint of proof complexity—these two systems are no stronger than extended resolution. We do so by showing that (1) extended resolution polynomially simulates DRAT and (2) DRAT polynomially simulates DPR. We implemented our simulations as proof-transformation tools and evaluated them to observe their behavior in practice. Finally, as a side note, we show how Kullmann's proof system based on blocked clauses (another generalization of extended resolution) is related to the other systems.

This work has been supported by the National Science Foundation under grant CCF-1910438, by the Austrian Science Fund (FWF) under project W1255-N23, by the Vienna Science and Technology Fund (WWTF) under projects VRG11-005 and ICT15-103, and by Microsoft Research through its PhD Scholarship Programme.

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany · TU Wien, Vienna, Austria · Carnegie Mellon University, Pittsburgh, USA · Johannes Kepler University, Linz, Austria

1 Introduction

When we look at proof systems for propositional logic, we observe an interesting peculiarity: Even though *extended resolution*, invented by Tseitin in the sixties [30], is known to be highly expressive, the practitioners in SAT solving have come up with different proof systems on which they base their solvers. The most important of these proof systems is likely DRAT [35], which can be considered the de-facto standard in SAT solving: Not only are the solvers in the annual SAT competitions required to produce DRAT proofs but also the proofs of long-standing mathematical problems, including the Boolean Erdős Discrepancy Conjecture [21] and the Boolean Pythagorean Triples Problem [11], were provided in DRAT.

One reason for the use of DRAT is that it can compactly represent many of the techniques used by modern SAT solvers. Moreover, due to its close relationship to unit propagation—which is a core part of modern SAT solvers—the correctness of DRAT proofs can be checked efficiently, leading to the development of formally verified DRAT proof checkers. Spinning the idea of propagation-based proof systems even further, the proof system DPR [13, 16] has been introduced as a generalization of DRAT. DPR allows for short proofs without the need for new variables, thus making it a strong candidate for practical SAT solving. In fact, the solver SaDiCaL [15], which implements the DPR-based *satisfaction-driven clause learning* (SDCL) paradigm [14], can automatically find short proofs of the pigeon-hole principle, Tseitin formulas over expander graphs [30], and mutilated chessboard problems [25]. All these problems are infamous in the proof-complexity literature for being extremely hard [9, 31, 1, 8], thus causing usual conflict-driven clause learning (CDCL) [24, 26] solvers some serious trouble.

While it seems clear that both DRAT and DPR provide practical advantages over extended resolution, it has long been unclear whether these advantages also manifest themselves in theory in the sense that they can lead to exponentially shorter proofs for some formulas. In this paper, we show that they do not. We do so by providing polynomial simulations between the mentioned proof systems. Specifically, we give two polynomial-time procedures—the first procedure takes as input a DRAT proof and returns as output an extended-resolution proof of the same formula; the second procedure takes as input a DPR proof and returns as output a DRAT proof. Together, the two procedures can be used to transform DPR proofs into extended-resolution proofs.

Our results confirm the expected proof-complexity landscape in which all top-tier proof systems—including extended resolution, DRAT, DPR, and extended Frege systems [32]—are essentially equivalent. Rounding off the picture, we show how blocked-clause addition [22]—a generalization of the extension rule from extended resolution—can be used to replace the addition of resolution asymmetric tautologies (RATs) in DRAT without introducing new variables. Our paper thus bridges the gap between proof systems from the present and from the past.

To evaluate the increase in size caused by our simulations in practice, we implemented them as proof-transformation tools and performed experiments on a range of DRAT and DPR proofs. The experiments show that the simulations incur a size increase that, though non-negligible, is relatively modest compared to the theoretical worst case. Our transformation tools thus allow practitioners to transform the output of SAT solvers into a format that might suit their applications better. Moreover, the transformation from DPR to DRAT enables the use of formally verified DRAT proof checkers for DPR proof checking.

The main contributions of this paper are as follows: (1) We prove that extended resolution polynomially simulates DRAT. (2) We prove that DRAT polynomially simulates DPR. (3) We implemented our simulations as tools. (4) We present an empirical evaluation of our simulation tools. (5) We show how blocked-clause addition can be used as an alternative for resolution-asymmetric-tautology addition in DRAT.

This paper is an extended version of our IJCAR 2018 best paper [20] and our TACAS 2018 paper [10].

2 Preliminaries

Here we present the background required for understanding this paper. We consider propositional formulas in *conjunctive normal form* (CNF), which are defined as follows. A *literal* is either a variable x (a *positive literal*) or the negation $\bar{x}$ of a variable x (a *negative literal*). The *complementary literal* $\bar{l}$ of a literal l is defined as $\bar{l} = \bar{x}$ if $l = x$ and $\bar{l} = x$ if $l = \bar{x}$. For a literal l, we denote the variable of l by $var(l)$. A *clause* is a disjunction of literals; we assume that clauses do not contain repeated literals. A *unit clause* is a clause that contains exactly one literal; a *tautology* contains complementary literals. A *formula* is a conjunction of clauses. We view clauses as sets of literals and formulas as sets of clauses. A clause C *subsumes* a clause D if $C \subseteq D$.

An *assignment* is a function from a set of variables to the truth values 1 (*true*) and 0 (*false*). An assignment is *total* with respect to a formula if it assigns a truth value to every variable occurring in the formula. We often denote assignments by the sequences of literals they satisfy. For instance, $x\,\bar{y}$ denotes the assignment that assigns 1 to x and 0 to y. A literal l is *satisfied* by an assignment α if l is positive and $\alpha(var(l)) = 1$ or if it is negative and $\alpha(var(l)) = 0$. A literal is *falsified* by an assignment if its complement is satisfied by the assignment. A clause is satisfied by an assignment α if it contains a literal that is satisfied by α. Finally, a formula is satisfied by an assignment α if all its clauses are satisfied by α. A formula is *satisfiable* if there exists an assignment that satisfies it. Two formulas are *logically equivalent* if they are satisfied by the same total assignments. Two formulas are *satisfiability-equivalent* if they are either both satisfiable or both unsatisfiable.

Given a clause C and an assignment α, we define $C\,|_\alpha$ as the clause obtained from C by removing all literals that are falsified by α. If F is a formula, we

define $F|_\alpha = \{C|_\alpha \mid C \in F \text{ and } \alpha \text{ does not satisfy } C\}$ also denoted as F under α. The result of applying the *unit-clause rule* to a formula F is the formula $F|_a$ (i.e., the formula $F|_\alpha$ with $\alpha = a$) where (a) is a unit clause in F. We also refer to applications of the unit-clause rule as *unit-propagation steps*. The iterated application of the unit-clause rule to a formula, until no unit clauses are left, is called *unit propagation*. If unit propagation on F yields the empty clause $\bot$, we say that it *derives a conflict* on F. For example, unit propagation derives a conflict on $F = (\bar{a} \vee b) \wedge (\bar{b}) \wedge (a)$ since $F|_a = (b) \wedge (\bar{b})$ and $F|_{ab} = \bot$.

For the rest of the paper, the notion of *implication via unit propagation* and the corresponding RUP clauses (short for *reverse unit propagation*) will be essential [34]:

Definition 1 *A clause $C = (c_1 \vee \cdots \vee c_k)$ is a* RUP *in a formula F if unit propagation derives a conflict on $F \wedge (\bar{c}_1) \wedge \cdots \wedge (\bar{c}_k)$. If C is a* RUP *in F, we say that F* implies *C* via unit propagation*, which we denote by $F \vdash_1 C$.*

For example, $(\bar{a} \vee c) \wedge (\bar{b} \vee \bar{c})$ implies $(\bar{a} \vee \bar{b})$ via unit propagation since unit propagation derives a conflict on $(\bar{a} \vee c) \wedge (\bar{b} \vee \bar{c}) \wedge (a) \wedge (b)$. Observe that if C is a resolvent of two clauses in a formula F, or if F contains a clause D that subsumes C, then C is a RUP in F. We also say that a formula F implies a formula G via unit propagation, denoted by $F \vdash_1 G$, if $F \vdash_1 C$ for every $C \in G$.

We define proof systems and polynomial simulations following Cook and Reckhow [6]:

Definition 2 *A* proof system *for propositional formulas in CNF is a surjective polynomial-time-computable function $f : \Sigma^* \to \mathcal{F}$ where Σ is some alphabet and $\mathcal{F}$ is the set of all unsatisfiable formulas.*

A proof system can thus be seen as a proof-checking function f that takes a *proof candidate* P (which is a string over Σ) together with an unsatisfiable formula F and checks in polynomial time if P is a correct proof of F. The requirement that f is surjective means that there must exist a proof for *every* unsatisfiable formula. We sometimes use the word *proof system* in a more colloquial way to denote the rules that define what constitutes a correct proof of a certain type. The *size* of a proof is the number of symbols occurring in it.

Definition 3 *A proof system $f_1 : \Sigma_1^* \to \mathcal{F}$* polynomially simulates *a proof system $f_2 : \Sigma_2^* \to \mathcal{F}$ if there exists a polynomial-time-computable function $g : \Sigma_2^* \to \Sigma_1^*$ such that $f_1(g(x)) = f_2(x)$.*

In other words, f_1 polynomially simulates f_2 if there exists a polynomial-time-computable function that transforms f_2-proofs into f_1-proofs. We next present the proof systems *extended resolution*, DRAT, and DPR.

3 Extended Resolution (ER), DRAT, and DPR

We model proofs of a formula F as sequences $C_1, \dots, C_m, I_{m+1}, \dots, I_n$, where $C_1, \dots, C_m$ are clauses of F and $I_{m+1}, \dots, I_n$ are *instructions* as defined in the following. There are three different kinds of instructions: addition, deletion, and extension. An *addition* is either a pair $\langle \mathsf{a}, C\rangle$ or a triple $\langle \mathsf{a}, C, \omega\rangle$ where C is a clause and ω is an assignment; a *deletion* is a pair $\langle \mathsf{d}, C\rangle$ where C is a clause; and an *extension* (also called a *definition introduction*) is a pair $\langle \mathsf{e}, \varphi\rangle$ where φ is a definition of the form $x \leftrightarrow p \vee (c_1 \wedge \dots \wedge c_k)$ with x being a variable that does not occur in any earlier instructions of the proof and $p, c_1, \dots, c_k$ being literals whose variables are pairwise distinct. The CNF conversion of such a definition is the clause set $\mathsf{cnf}(\varphi) = \{(x \vee \bar{p}),\ (x \vee \bar{c}_1 \vee \dots \vee \bar{c}_k),\ (\bar{x} \vee p \vee c_1), \dots, (\bar{x} \vee p \vee c_k)\}$; in the particular case $k = 0$ we have $\mathsf{cnf}(\varphi) = \{(x \vee \bar{p}),\ (x)\}$. The sequence $C_1, \dots, C_m, I_{m+1}, \dots, I_n$ gives rise to formulas $F_0, F_1, \dots, F_n$ as follows:

$$F_i = \begin{cases} \{C_1, \dots, C_i\} & \text{if } i \le m \\ F_{i-1} \cup \{C\} & \text{if } i > m \text{ and } I_i \text{ is of the form } \langle \mathsf{a}, C\rangle \text{ or } \langle \mathsf{a}, C, \omega\rangle \\ F_{i-1} \setminus \{C\} & \text{if } i > m \text{ and } I_i \text{ is of the form } \langle \mathsf{d}, C\rangle \\ F_{i-1} \cup \mathsf{cnf}(\varphi) & \text{if } i > m \text{ and } I_i \text{ is of the form } \langle \mathsf{e}, \varphi\rangle \end{cases}$$

We call F_i the *accumulated formula* corresponding to the i-th instruction. Based on this, we can now define the details of extended resolution and DRAT. In both proof systems, a correct proof of a formula F must derive the empty clause $\bot$, i.e., $\bot \in F_n$. They differ only in the instructions they permit.

3.1 Extended Resolution

Extended resolution combines resolution with the *extension rule*: A sequence $C_1, \dots, C_m, I_{m+1}, \dots, I_n$ is a correct extended-resolution proof of a formula F if every instruction $I_i \in I_{m+1}, \dots, I_n$ is either (1) an addition $\langle \mathsf{a}, (C \vee D)\rangle$ where $(C \vee D)$ is the *resolvent* $(C \vee p) \otimes_p (D \vee \bar{p})$ of two clauses $(C \vee p)$ and $(D \vee \bar{p})$ occurring in F_{i-1}, or (2) an extension $\langle \mathsf{e}, \varphi\rangle$. When Tseitin originally introduced the extension rule [30], he only allowed definitions of the form $x \leftrightarrow (\bar{a} \vee \bar{b})$ where a and b are variables. These definitions correspond to the clauses $(x \vee a)$, $(x \vee b)$, and $(\bar{x} \vee \bar{a} \vee \bar{b})$. However, more general definitions can be derived from these basic definitions in a simple but tedious way. Because of this, more general extension rules are common in the literature, some even allowing definitions $x \leftrightarrow \psi$ where ψ is an arbitrary propositional formula over previous variables (cf. [5, 9, 29]).

3.2 DRAT

A sequence $C_1, \dots, C_m, I_{m+1}, \dots, I_n$ is a correct DRAT proof of a formula F if every instruction $I_i \in I_{m+1}, \dots, I_n$ is either (1) a deletion $\langle \mathsf{d}, C\rangle$ where C is an

arbitrary clause, or (2) an addition $\langle \mathsf{a}, C \rangle$ where C is a RAT or a RUP in F_{i-1}; we have already introduced RUPs in Definition 1 on page 4. A RAT is then simply a clause for which all resolvents upon one of its literals are RUPs [18]:

Definition 4 *A clause $(C \vee p)$ is a* resolution asymmetric tautology (RAT) *on p in a formula F if for every clause $(D \vee \bar{p}) \in F$, the resolvent $(C \vee D)$ is implied by F via unit propagation.*

Example 1 Consider the formula $F = (\bar{p} \vee \bar{a}) \wedge (\bar{p} \vee b) \wedge (b \vee c) \wedge (\bar{c} \vee a)$ and the clause $C = (a \vee p)$. There are two resolvents of C upon p: The resolvent $(a \vee \bar{a})$, obtained by resolving with $(\bar{p} \vee \bar{a})$, is a tautology and thus trivially a RUP in F; the resolvent $(a \vee b)$, obtained by resolving with $(\bar{p} \vee b)$, is a RUP in F since unit propagation derives a conflict on $F \wedge (\bar{a}) \wedge (\bar{b})$. It follows that C is a RAT on p in F. □

As shown in [18], if a clause C is a RAT on p in a formula F, then F and $F \wedge C$ are satisfiability-equivalent. The idea behind the proof is that every satisfying assignment of F that does not satisfy C can be turned into a satisfying assignment of $F \wedge C$ by making the literal p true.

Observe that if C is a non-empty RUP in F, it is a RAT in F on any literal $p \in C$ (the empty clause $\bot$ cannot be a RAT as it contains no literals). In the rest of the paper, we thus call a clause a *proper* RAT if it is a RAT on some literal p but not a RUP. The addition of definition clauses, as with the extension rule, is a special case of blocked-clause addition [17] (see Section 5), which itself is a particular case of RAT addition. We thus regard DRAT as a generalization of extended resolution.

3.3 DPR

A sequence $C_1, \ldots, C_m, I_{m+1}, \ldots, I_n$ is a correct DPR proof of a formula F if every instruction $I_i \in I_{m+1}, \ldots, I_n$ is either (1) a deletion $\langle \mathsf{d}, C \rangle$ where C is an arbitrary clause, or (2) an addition $\langle \mathsf{a}, C \rangle$ where C is a RUP in F_{i-1}, or (3) an addition $\langle \mathsf{a}, C, \omega \rangle$ where C is propagation-redundant with respect to F_{i-1} and ω; we define propagation-redundancy in the following, it is based on the notion of *precluded* assignments:

Definition 5 *Given an assignment $\alpha = a_1 \ldots a_k$, the clause $(\bar{a}_1 \vee \cdots \vee \bar{a}_k)$ is the clause that* precludes *α.*

With this we can now define propagation-redundancy [16]:

Definition 6 *Let F be a formula, C a clause, α the assignment precluded by C, and ω an assignment that satisfies C. Then, C is* propagation redundant (PR) *with respect to F and ω if $F|_\alpha \vdash_1 F|_\omega$.*

We call ω the witness for the propagation-redundancy of C.

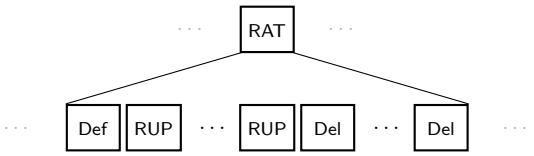


Fig. 1. We transform a RAT addition into a definition introduction (Def), followed by RUP additions and clause deletions (Del).

Example 2 Let $F = (x \vee y) \wedge (\bar{x} \vee y) \wedge (\bar{x} \vee z)$, $C = (x)$, and let $\omega = x\,z$. Then, C precludes the assignment $\alpha = \bar{x}$, and ω satisfies C. Now, consider the formulas $F|_\alpha = (y)$ and $F|_\omega = (y)$. Clearly, $F|_\alpha \vdash_1 F|_\omega$, and so C is propagation redundant with respect to F and witness ω.

If C is a RAT on p in a formula F, then C is propagation-redundant with respect to F and some witness ω [16]. Hence, DPR can be seen as a generalization of DRAT. Moreover, if a DPR proof contains no deletions, we call it a PR proof. Likewise, DRAT proofs without deletions are RAT proofs.

We now proceed to showing that extended resolution polynomially simulates DRAT. After this, we show that DRAT polynomially simulates DPR.

4 Simulating DRAT with Extended Resolution

We perform the transformation of a DRAT proof into an extended-resolution proof in four stages. In the first stage, we use the extension rule together with RUP addition and clause deletion to eliminate all additions of proper RATs. In the second stage, we get rid of all clause deletions. In the third stage, we then replace all RUP additions by resolution inferences and subsumed-clause additions. Finally, in the fourth stage, we also eliminate the subsumed-clause additions to obtain a correct extended-resolution proof.

4.1 Eliminating Additions of Proper RATs

Given a DRAT proof $C_1, \ldots, C_m, I_{m+1}, \ldots, I_n$, we iterate over the instructions $I_{m+1}, \ldots, I_n$ and replace every addition $I_i = \langle \mathsf{a}, (p \vee C) \rangle$ of a clause $(p \vee C)$ that is a proper RAT on p in the accumulated formula F_{i-1} by a sequence π_i of instructions. As illustrated in Fig. 1, such a sequence π_i consists of a single definition introduction followed first by several RUP additions and then by several clause deletions. In the case where I_i is not the addition of a proper RAT, we simply let π_i be I_i. At the end of this iterative process, we obtain a sequence $C_1, \ldots, C_m, \pi_{m+1}, \ldots, \pi_n$, where every π_i is a sequence of instructions corresponding to the instruction I_i from the original proof. The sequence $C_1, \ldots, C_m, \pi_{m+1}, \ldots, \pi_n$ contains no additions of proper RATs, but instead contains definition introductions.

Each iteration of this process performs the following transformation, where I_i is an addition instruction of a clause $C = (p \vee c_1 \vee \cdots \vee c_k)$ which is a RAT on literal p in the accumulated formula F_{i-1} before I_i.

$$\begin{array}{c} C_1, \ldots, C_m, \pi_{m+1}, \ldots, \pi_{i-1}, I_i, I_{i+1}, \ldots, I_n \\ \wr \\ C_1, \ldots, C_m, \pi_{m+1}, \ldots, \pi_{i-1}, \pi_i, I'_{i+1}, \ldots, I'_n \end{array}$$

We first use the extension rule to introduce a clause $(x \vee c_1 \vee \cdots \vee c_k)$ as well as some other definition clauses, where x is a *new* variable in the sense that it is not used anywhere else in the proof. Note that $(x \vee c_1 \vee \cdots \vee c_k)$ differs from C only on the literal p, which is replaced by the variable x. We then use RUP additions and clause deletions to replace all occurrences of p in F_{i-1} by x. Our procedure guarantees that the formula accumulated after π_i in the resulting sequence is exactly $F_i[x/p]$, obtained from $F_i = F_{i-1} \cup \{C\}$ (the accumulated formula after I_i in the original proof) by simultaneously replacing occurrences of p by x and occurrences of $\bar{p}$ by $\bar{x}$.

As a consequence, the correctness of the whole proof is preserved by simply renaming p to x, and $\bar{p}$ to $\bar{x}$, in all later instructions, resulting in the instructions $I'_{i+1}, \ldots, I'_n$. It is thus clear that the size of the accumulated formula after π_i in the new proof is the same as that of F_i in the original proof; this property will be crucial for the complexity analysis in Section 4.5. We now explain in detail how the sequence π_i is obtained, and provide an example to illustrate the procedure.

(1) Use the extension rule to introduce the definition $x \leftrightarrow p \vee (\bar{c}_1 \wedge \cdots \wedge \bar{c}_k)$. This adds the clause set $\{(x \vee c_1 \vee \cdots \vee c_k),\ (x \vee \bar{p}),\ (\bar{x} \vee p \vee \bar{c}_1), \ldots, (\bar{x} \vee p \vee \bar{c}_k)\}$. The first clause will be our replacement of the RAT $(p \vee c_1 \vee \cdots \vee c_k)$. This is similar to the definitions introduced to express conditional overwrites in propositional logic in [28], and intuitively follows the correctness proof of RAT clause addition from [18]: given any interpretation satisfying F_{i-1}, we can construct another interpretation satisfying F_i by conditionally changing the truth value of p, precisely as given by the definition of x. The rest of the transformation simply replaces occurrences of p by x.

(2) Replace the literal p in all clauses of F_{i-1} by the new variable x:

   (a) Add for every clause $(D \vee p) \in F_{i-1}$ the clause $(D \vee x)$. This is a RUP addition since $(D \vee x)$ is a resolvent of $(D \vee p)$ and $(x \vee \bar{p})$.

   (b) Add for every clause $(D \vee \bar{p}) \in F_{i-1}$ the clause $(D \vee \bar{x})$. To show that this is a correct RUP addition, we show that unit propagation derives a conflict on $F_{i-1} \wedge \bar{D} \wedge (x)$, where $\bar{D}$ is the conjunction of the negated literals of D. As C is a RAT on p in F_{i-1}, we know that the resolvent $(c_1 \vee \cdots \vee c_k \vee D)$ of C and $(D \vee \bar{p})$ is a RUP in F_{i-1}. Now, by propagating the unit clauses of $\bar{D}$, we derive $(\bar{p})$ because the clause $(D \vee \bar{p})$ is in F_{i-1}. After this, we propagate x and $\bar{p}$ to derive all the unit clauses $(\bar{c}_1), \ldots, (\bar{c}_k)$ from the clauses $(\bar{x} \vee p \vee \bar{c}_j)$ with

$j \in 1, \ldots, k$. But then we have derived the negations of all literals in the resolvent $(c_1 \vee \cdots \vee c_k \vee D)$, and since this resolvent is a RUP in F_{i-1}, unit propagation must eventually derive a conflict.

(c) Delete all clauses containing p or $\bar{p}$, including those added in step 1. Note that this does not delete the clause $(x \vee c_1 \vee \cdots \vee c_k)$.

Example 3 Say we are given a proof $C_1, \ldots, C_m, I_{m+1}, \ldots, I_i, \ldots, I_n$ and we want to eliminate the addition $I_i = \langle \mathsf{a}, C \rangle$ where $C = (p \vee a)$ is a proper RAT on p in the accumulated formula $F_{i-1} = \{(\bar{p} \vee b),\ (a \vee b \vee c),\ (\bar{c} \vee d),\ (\bar{d}),\ (\bar{a} \vee p)\}$. Observe that C is a RAT on p because the resolvent $(a \vee b)$, obtained by resolving C with $(\bar{p} \vee b)$ upon p, is a RUP in F_{i-1}.

We first use the extension rule to add the definition $x \leftrightarrow (p \vee \bar{a})$. This adds the clauses $(x \vee a)$, $(x \vee \bar{p})$, and $(\bar{x} \vee p \vee \bar{a})$. Next, we need to replace the literal p in F_{i-1} by x. To do so, we first resolve $(x \vee \bar{p})$ with $(\bar{a} \vee p)$ to derive $(\bar{a} \vee x)$. Then, we introduce the RUP $(\bar{x} \vee b)$ for the existing clause $(\bar{p} \vee b)$. (It can be easily seen that the clause $(\bar{x} \vee b)$ is a RUP in the accumulated formula $F_{i-1} \cup \{(\bar{x} \vee p \vee \bar{a}),\ (x \vee \bar{p}),\ (x \vee a)\}$: By propagating $(\bar{b})$, we derive $(\bar{p})$ from $(\bar{p} \vee b)$. After this, the propagation of (x) and $(\bar{p})$ derives $(\bar{a})$ from $(\bar{x} \vee p \vee \bar{a})$. But then further propagation will eventually lead to a conflict because $(a \vee b)$, which is the resolvent of $(p \vee a)$ and $(\bar{p} \vee b)$, is a RUP in F_{i-1}.) Finally, we delete all clauses containing p or $\bar{p}$. We thus obtain the proof $C_1, \ldots, C_m, I_{m+1}, \ldots, I_{i-1}, \pi_i, \ldots, I_n$ where π_i is the instruction sequence $\langle \mathsf{e}, x \leftrightarrow (p \vee \bar{a}) \rangle$, $\langle \mathsf{a}, (\bar{a} \vee x) \rangle$, $\langle \mathsf{a}, (\bar{x} \vee b) \rangle$, $\langle \mathsf{d}, (\bar{p} \vee b) \rangle$, $\langle \mathsf{d}, (\bar{a} \vee p) \rangle$, $\langle \mathsf{d}, (\bar{x} \vee p \vee \bar{a}) \rangle$, $\langle \mathsf{d}, (x \vee \bar{p}) \rangle$. After the last instruction of π_i, we get the accumulated formula $\{(\bar{x} \vee b),\ (a \vee b \vee c),\ (\bar{c} \vee d),\ (\bar{d}),\ (\bar{a} \vee x),\ (x \vee a)\}$, which is precisely $F_i[x/p]$. We then just need to replace p by x and $\bar{p}$ by $\bar{x}$ in $I_{i+1}, \ldots, I_n$ to obtain a correct proof $C_1, \ldots, C_m, I_{m+1}, \ldots, I_{i-1}, \pi_i, I'_{i+1}, \ldots, I'_n$. □

4.2 Eliminating Clause Deletions

At this point, our proof is a sequence of (1) clauses from the original formula, (2) definition introductions, (3) RUP additions, and (4) clause deletions. Since no additions of proper RATs remain in the proof, the elimination of a deletion instruction does not affect the correctness of other proof instructions: The addition of RUPs depends only on the existence of clauses in the accumulated formula but not on their non-existence (if C is a RUP in F, it is a RUP in every superset of F). Also the extension rule is not affected by additional clauses. By simply eliminating all deletions, we thus end up with a correct proof. Note that this would not work if *proper* RAT additions were still present, because they depend on the non-existence of certain clauses (a clause C is a RAT in a formula F only if F contains *no* resolvents with C that are not RUPs).

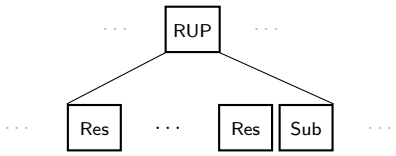


Fig. 2. We transform a RUP addition into a sequence of resolution steps (Res) followed by a single subsumed-clause addition (Sub).

```
1  C_{n+1} ← D_{n+1}
2  for i = n, ..., 1 do
3    if ā_i ∈ C_{i+1} then C_i ← D_i ⊗_{a_i} C_{i+1}
4    else C_i ← C_{i+1}
```

Algorithm 1. Given a RUP C, the algorithm derives a clause $C_1 \subseteq C$.

4.3 Eliminating RUP Additions

Similar to the first stage of our simulation, we again iterate over the proof from the beginning. In this stage, we replace all additions of RUPs that are neither resolvents nor subsumed clauses. In the following, we show how the addition of such a RUP can be transformed into a sequence of resolution steps followed by a single subsumed-clause addition. This is illustrated in Fig. 2. We note that this has already been explained on a high level in the literature [33, 27].

Let us first observe that, given a correct proof containing only RUP additions and definition introductions, the RUP additions of tautological clauses can be directly eliminated. To see this, simply observe that definition introductions are never affected by the presence of tautologies. Furthermore, if a clause C is a RUP in F, and F contains a tautology $(a \vee \bar{a} \vee D)$, the latter never becomes a unit clause in $F|_\alpha$ under any assignment α; therefore, C is also a RUP in the formula resulting from removing tautologies from F. In the following, we thus consider only proofs without tautological clauses.

If a non-tautological clause C is a RUP in a formula F, we know that unit propagation derives a conflict on $F \wedge \bar{C}$ where $\bar{C}$ is the conjunction of the negated literals in C. This is equivalent to saying that unit propagation derives a conflict on $F|\bar{C}$, viewing $\bar{C}$ as the assignment that satisfies $\bar{C}$. Hence, there exists a (possibly empty) sequence of literals $a_1, \ldots, a_n$ such that the unit clause (a_i) occurs in $F|\bar{C}a_1 \ldots a_{i-1}$ for each $1 \leq i \leq n$, and the empty clause $\bot$ occurs in $F|\bar{C}a_1 \ldots a_n$. Intuitively, (a_i) is the unit clause propagated at the i-th propagation step after all unit clauses in $\bar{C}$ have been propagated. These unit clauses and the empty clause stem from clauses $D_1, \ldots, D_{n+1} \in F$ with the following properties: (I) the clause $D_i|\bar{C}a_1 \ldots a_{i-1}$ is the unit clause (a_i) for $1 \leq i \leq n$, (II) D_i is not satisfied by $\bar{C}a_1 \ldots a_{i-1}$ for $1 \leq i \leq n+1$, and (III) the clause $D_{n+1}|\bar{C}a_1 \ldots a_n$ is the empty clause.

Algorithm 1 uses the clauses $D_1, \ldots, D_{n+1}$ as follows: It starts with the last clause, D_{n+1}, and step-by-step resolves it with the clauses $D_n, \ldots, D_1$ until it

obtains a clause C_1 that subsumes C. Using C_1, we can then derive C with a subsumed-clause addition. Note that the algorithm performs n iterations and that the clauses $D_1, \ldots, D_{n+1} \in F$ can be found by performing unit propagation on $F \wedge (\bar{c}_1) \wedge \cdots \wedge (\bar{c}_k)$, where $C = (c_1 \vee \cdots \vee c_k)$. As unit propagation is known to run in polynomial time, Algorithm 1 thus also runs in polynomial time with respect to $F \wedge C$. Example 4 illustrates the algorithm.

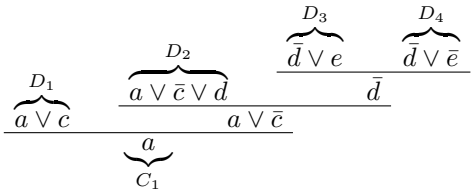
Example 4 Consider the clause $C = (a \vee b)$ and $F = D_1 \wedge D_2 \wedge D_3 \wedge D_4$ where:

$$D_1 = (a \vee c) \qquad D_2 = (a \vee \bar{c} \vee d) \qquad D_3 = (\bar{d} \vee e) \qquad D_4 = (\bar{d} \vee \bar{e})$$

The clause C is a RUP in F because unit propagation derives a conflict on $F \wedge (\bar{a}) \wedge (\bar{b})$, or equivalently, it derives a conflict on $F|\bar{a}\bar{b}$. To illustrate this, we perform the unit propagation:

$$D_1|\bar{a}\bar{b} = (c) \qquad D_2|\bar{a}\bar{b}c = (d) \qquad D_3|\bar{a}\bar{b}cd = (e) \qquad D_4|\bar{a}\bar{b}cde = \bot$$

Our algorithm now performs resolution steps as follows:



As we can see, the resulting clause $C_1 = (a)$ subsumes $C = (a \vee b)$. □

Lemma 1 *If a formula F implies a non-tautological clause C via unit propagation, then the clause C_1, computed by Algorithm 1, subsumes C.*

Proof We show by induction that, for every $1 \leq i \leq n+1$, the clause C_i computed by Algorithm 1 satisfies $C_i|\bar{C}a_1 \ldots a_{i-1} = \bot$. The claim then follows from $C_1|\bar{C} = \bot$, which is equivalent to $C_1 \subseteq C$.

Base case ($i = n+1$): Follows from $C_{n+1} = D_{n+1}$ and property (III).

Induction step ($1 \leq i \leq n$): Assume the claim holds for $i+1$. Then, we have $C_{i+1}|\bar{C}a_1 \ldots a_i = \bot$, and from property (I) we know $D_i|\bar{C}a_1 \ldots a_{i-1} = (a_i)$. Now, if $\bar{a}_i \notin C_{i+1}$, then $C_{i+1}|\bar{C}a_1 \ldots a_{i-1} = \bot$. In this case, the algorithm sets $C_i = C_{i+1}$ and so the claim holds for i. In contrast, if C_{i+1} contains $\bar{a}_i$, then the algorithm sets $C_i = D_i \otimes_{a_i} C_{i+1}$. But then, as C_i contains only literals of D_i and C_{i+1} except for a_i and $\bar{a}_i$, the claim also follows for i. □

The following statement, which is a variant of Theorem 2 in [27] as well as of the Theorem of Lee [23], is a consequence of Lemma 1; it allows us to repeatedly eliminate all additions of RUPs that are not resolvents or subsumed clauses.

Theorem 2 *If a formula F implies a non-tautological clause C via unit propagation using n propagation steps, then we can derive C from F via at most n resolution steps followed by one subsumed-clause addition.*

4.4 Eliminating Subsumed-Clause Additions

At this point, every instruction is either a definition introduction or it adds a resolvent or a subsumed clause. Since the extension rule does not depend on previous clauses, we can reorder the instructions of our proof so that all definition introductions occur before all addition instructions.

Now, by a well-known method (e.g., [2]) we can eliminate all subsumed-clause additions from the latter part of our proof. The procedure works by recursively labeling every clause in the proof with a subclause. These labels give a resolution proof, possibly with unnecessary inferences. The labeling proceeds as follows:

1. We label every leaf clause by itself.
2. For each resolvent of two clauses $(C_1 \vee x)$ and $(C_2 \vee \bar{x})$, which are labeled by D_1 and D_2 respectively, we label the resolvent by D_1 if $x \notin D_1$; by D_2 if $\bar{x} \notin D_2$; and by the resolvent of D_1 and D_2 upon x if $x \in D_1$ and $\bar{x} \in D_2$.
3. For each subsumption inference from a clause C that is labeled by D, we label the subsumed clause by D.

It is straightforward to check that the labels define a resolution derivation without subsumed-clause additions; in fact, a refutation, as the only subclause of $\bot$ is $\bot$ itself. The resulting derivation may contain redundant parts such as unused subderivations, but these do not affect our analysis and can be removed easily. After eliminating all subsumed-clause additions, we finally obtain an extended-resolution proof.

Example 5 The proof tree below includes subsumed-clause additions 1 and 2. The clauses in the proof that are strict supersets of their labels are afterwards dropped from the proof (for instance, $(a \vee \bar{b})$ is dropped because $\{\bar{b}\} \subset \{a, \bar{b}\}$):

$$\dfrac{\dfrac{a \vee b \quad [a \vee b] \qquad \overset{1}{\dfrac{\bar{b} \quad [\bar{b}]}{a \vee \bar{b} \quad [\bar{b}]}}}{a \quad [a]} \qquad \dfrac{\bar{a} \vee c \quad [\bar{a} \vee c] \qquad \overset{2}{\dfrac{d \quad [d]}{\bar{c} \vee d \quad [d]}}}{\bar{a} \vee d \quad [d]} \qquad \bar{a} \vee \bar{d} \quad [\bar{a} \vee \bar{d}]}{\bar{a} \quad [\bar{a}]}}{\bot \quad [\bot]}$$

After dropping clauses, the result is the following proof:

$$\dfrac{\dfrac{a \vee b \qquad \bar{b}}{a} \qquad \dfrac{d \qquad \bar{a} \vee \bar{d}}{\bar{a}}}{\bot}$$

4.5 Complexity of the Simulation

We show now that our simulation only involves a polynomial blow-up. To simplify the presentation, we use the number of literals (with repetitions) in a proof P as the measure for its size, denoted by $\|P\|$. After we have shown that

the size of the resulting extended-resolution proof is polynomial compared to the original DRAT proof, it should be clear that the computation of the simulation is also polynomial, given the simplicity of the used techniques (the only stage where this might not be straightforward is stage 2, for which we discussed in Section 4.3 why it runs in polynomial time). Let the original DRAT proof be $P = C_1, \ldots, C_m, I_{m+1}, \ldots, I_n$. Note first that for every $m+1 \leq i \leq n$, the size $\|I_i\|$ of the instruction I_i, and the size $\|F_i\|$ of the accumulated formula F_i are both bounded by $\mathcal{O}(\|P\|)$. Note also that the elimination of clause deletions and subsumed-clause additions shrinks the proof. Hence, out of the four stages in the simulation, we only need to consider the first stage (elimination of RAT additions) and the third stage (elimination of RUP additions) to obtain an upper bound on the proof size.

Elimination of RAT *additions.* For the following, remark that for $i \in m+1, \ldots, n$, the size of the accumulated formula after the i-th proof fragment π_i (obtained by transforming the instruction I_i) in the new proof is the same as that of F_i in the original DRAT proof (we explained this on page 8). For the elimination of a single RAT addition of a clause $(p \vee c_1 \vee \cdots \vee c_k)$, we first add the definition $x \leftrightarrow p \vee (\bar{c}_1 \wedge \cdots \wedge \bar{c}_k)$. This step is clearly $\mathcal{O}(\|P\|)$. After this, we add for each clause $(D \vee p) \in F_{i-1}$ the clause $(D \vee x)$, and we add for each clause $(D \vee \bar{p}) \in F_{i-1}$ the clause $(D \vee \bar{x})$. This leads to at most $\mathcal{O}(\|F_{i-1}\|) = \mathcal{O}(\|P\|)$ new literals. Finally, we delete all clauses containing p or $\bar{p}$. These deletions together are again of size at most $\mathcal{O}(\|F_{i-1}\|) = \mathcal{O}(\|P\|)$. Overall, the size of the proof generated by eliminating a single RAT addition is thus bounded by $3 \times \mathcal{O}(\|P\|) = \mathcal{O}(\|P\|)$. Finally, as we perform at most n such RAT eliminations and since $n = \mathcal{O}(\|P\|)$, the size of the resulting proof after eliminating all RATs is bounded by $\mathcal{O}(\|P\|^2)$.

Elimination of RUP *additions.* Before we eliminate RUPs, we have a proof whose size is $\mathcal{O}(\|P\|^2)$. We thus eliminate at most $\mathcal{O}(\|P\|^2)$ RUP additions. It remains to determine a bound for the size of the proof instructions obtained by eliminating a single RUP addition. Theorem 2 tells us that if C is a RUP that is implied via unit propagation using k propagation steps, we can derive C with at most k resolution steps followed by a single subsumed-clause addition. Clearly, the number of unit-propagation steps is bounded by the number of variables occurring in the proof (every variable can be propagated at most once). Now, the number of variables in the original proof P is clearly bounded by $\|P\|$ and since the elimination of RAT additions has introduced at most one new variable for every RAT, we have $\mathcal{O}(\|P\|)$ variables. Hence, a single RUP elimination leads to at most $\mathcal{O}(\|P\|)$ instructions. As the size of a single instruction is bounded by $\mathcal{O}(\|P\|)$ (a clause can contain at most two literals per variable), every RUP elimination results in a proof of size $\mathcal{O}(\|P\|^2)$. We conclude that the size of the resulting extended-resolution proof is $\mathcal{O}(\|P\|^4)$.

Note that our analysis is very conservative. For instance, representing resolvents implicitly (just pointing to their two parent clauses) instead of repre-

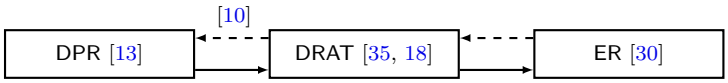


Fig. 3. A dashed line from X to Y means that X simulates Y polynomially. A solid line from X to Y means that every Y proof can be regarded as an X proof.

senting them explicitly shrinks the resulting extended-resolution proof significantly. As we will see in Section 7, the increase in size on practical DRAT proofs is way smaller than the theoretical bound we obtain here. Combining our result with the recent result that DRAT polynomially simulates DPR (a generalization of DRAT) [10], we obtain the complexity landscape depicted in Fig. 3.

5 Replacing RAT Addition With Blocked-Clause Addition

In our polynomial simulation, we needed to introduce a new variable for every proper RAT addition. This cannot be avoided because extended resolution without new variables is just ordinary resolution, and ordinary resolution is exponentially weaker than both DRAT and extended resolution [9]. We now show how *blocked-clause addition*, introduced by Kullmann [22] as a generalization of the extension rule from extended resolution, can be used to replace RAT addition *without* introducing new variables. This shows that a simple generalization of the extension rule is essentially as powerful as RAT addition, even when no new variables are introduced. Informally, a clause is blocked if all resolvents upon one of its literals are tautologies [22]:

Definition 7 *A clause C is* blocked *by a literal $p \in C$ in a formula F if all resolvents of C upon p with clauses in F are tautologies.*

Example 6 Consider the formula $F = (\bar{p} \vee \bar{b}) \wedge (\bar{p} \vee \bar{a}) \wedge (p \vee c) \wedge (a \vee c)$ and the clause $(a \vee b \vee p)$. There are two resolvents of $(a \vee b \vee p)$ upon p: The clause $(a \vee b \vee \bar{b})$, obtained by resolving with $(\bar{p} \vee \bar{b})$, and the clause $(a \vee b \vee \bar{a})$, obtained by resolving with $(\bar{p} \vee \bar{a})$. As both resolvents are tautologies, $(a \vee b \vee p)$ is blocked by p in F. □

Blocked clauses are thus more restricted than RATs: While the RAT property only requires all the resolvents to be implied via unit propagation, blocked clauses require them to be tautologies, which are trivially implied via unit propagation. Hence, every blocked clause is also a RAT but not vice versa.

We follow an iterative procedure similar to the one presented in Section 4. Suppose $C = (c_1 \vee \cdots \vee c_k \vee p)$ is a proper RAT on p in a formula F. To replace the addition of C to F, we first turn C into a blocked clause by replacing the resolution partners that do not lead to tautological resolvents. We then add the clause with blocked-clause addition and afterwards derive all the original

resolution partners again. As illustrated in Fig. 4, this leads to a sequence consisting of RUP additions, clause deletions, and a single blocked-clause addition. Specifically, we perform the following steps:

(1) For every $(D \vee \bar{p}) \in F_{i-1}$ whose resolvent $R = (c_1 \vee \cdots \vee c_k \vee D)$ with C upon p is not a tautology, add R with RUP addition. The resolvent R is guaranteed to be a RUP because C is a RAT on p in F_{i-1}.

(2) For every $(D \vee \bar{p}) \in F_{i-1}$ whose resolvent with C upon p is not a tautology, replace $(D \vee \bar{p})$ by the set $D_p = \{(\bar{c}_j \vee D \vee \bar{p}) \mid 1 \leq j \leq k\}$. Since all the clauses in D_p are subsumed by $(D \vee \bar{p})$, this replacement results in a sequence of deletions and RUP additions. Note that in case C is a unit clause, the set D_p is empty and so all resolution partners are deleted.

(3) Add C with blocked-clause addition. This is a correct addition because after step 2, every clause that contains $\bar{p}$ contains a literal $\bar{c}_j$ with $c_j \in C$. Hence, by resolving such a clause with C we obtain a tautology.

(4) Use RUP addition to add all the clauses $(D \vee \bar{p})$, which we replaced in step 2, again. The addition of such a clause $(D \vee \bar{p})$ is a correct RUP addition: If C is a unit clause, we have added $R = D$, which subsumes $(D \vee \bar{p})$, in step 1. If C is not a unit clause, then $D_p \cup \{R\}$ implies $(D \vee \bar{p})$ via unit propagation: By propagating p and the negated literals of D, we derive the unit clauses $(\bar{c}_1), \ldots, (\bar{c}_k)$ from the clauses in $D_p = \{(\bar{c}_j \vee D \vee \bar{p}) \mid 1 \leq j \leq k\}$. But these unit clauses lead to a conflict with the clause $(c_1 \vee \cdots \vee c_k)$, which we derive by propagating the negated literals of D on $R = (c_1 \vee \cdots \vee c_k \vee D)$.

(5) Delete all the RUPs added in step 1 and the clause sets D_p added in step 2.

Example 7 Consider $F = \{(\bar{p}),\ (a \vee b \vee c),\ (\bar{c} \vee d),\ (\bar{d}),\ (\bar{a} \vee e),\ (\bar{b} \vee e)\}$ and the clause $C = (a \vee b \vee p)$. The clause C is not blocked but it is a RAT on p in F, meaning that F implies the resolvent $(a \vee b)$ of C and $(\bar{p})$ via unit propagation. To turn C into a blocked clause, we first add $(a \vee b)$ with RUP addition. We next replace $(\bar{p})$ by the clauses $(\bar{p} \vee \bar{a})$ and $(\bar{p} \vee \bar{b})$ (both clauses are subsumed by $(\bar{p})$ and thus they are RUPs). Now $(\bar{p} \vee \bar{a})$ and $(\bar{p} \vee \bar{b})$ contain literals whose complements occur in C. We can thus add C with blocked-clause addition.

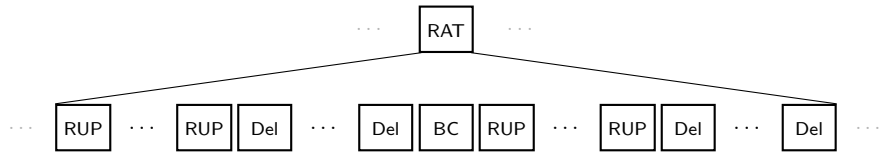


Fig. 4. We transform a RAT addition into a sequence consisting of RUP additions, clause deletions (Del), and a single blocked-clause addition (BC).

After this, we use RUP addition to add the original resolution partner ($\bar{p}$) again: This is a correct RUP addition because $(a \vee b)$, $(\bar{p} \vee \bar{a})$, and $(\bar{p} \vee \bar{b})$ together imply $(\bar{p})$ via unit propagation (to see this, observe that making p true forces $\bar{a}$ and $\bar{b}$ to be true which leads to a conflict with $(a \vee b)$). This step is actually the reason why we derived $(a \vee b)$ in the beginning. Finally, we delete the intermediate clauses $(a \vee b)$, $(\bar{p} \vee \bar{a})$, and $(\bar{p} \vee \bar{b})$ to obtain the formula $F \cup \{C\}$. □

6 Simulating DPR by DRAT

We show how a DPR proof $C_1, \ldots, C_m, I_{m+1}, \ldots, I_n$ can be transformed into a DRAT proof. The main idea is similar to the idea behind eliminating proper RAT additions in the simulation of DRAT with extended resolution: We iterate over the instructions $I_{m+1}, \ldots, I_n$ and replace every instruction I_i by a sequence π_i of instructions that are allowed in DRAT, i.e., additions of RATs or RUPs and deletions:

$$\begin{array}{l} C_1, \ldots, C_m, \pi_{m+1}, \ldots, \pi_{i-1}, I_i, I_{i+1}, \ldots, I_n \\ \qquad\qquad\qquad\qquad\qquad\qquad \wr \\ C_1, \ldots, C_m, \pi_{m+1}, \ldots, \pi_{i-1}, \pi_i, I'_{i+1}, \ldots, I'_n \end{array}$$

We define the transformation in such a way that the accumulated formula F_i after proof step I_i in the original PR proof is the same as the accumulated formula after the proof steps π_i in the resulting DRAT proof. Because of this, deletions in the DPR proof can simply be copied to the DRAT proof. The interesting remaining case is to transform addition instructions for PR clauses.

6.1 Transforming a Single PR Addition

We want to transform a PR addition step of the form $\langle \mathsf{a}, C_i, \omega_i \rangle$, where C_i is propagation-redundant with respect to F_{i-1} (the accumulated formula before the addition of C_i) and ω_i. On a high level, we do the following: We first take a fresh variable x (i.e., a variable not occurring in the rest of the proof). Then, we perform a sequence of DRAT steps that turn the accumulated formula F_{i-1} into a formula F'_{i-1} such that $(C_i \vee x)$ becomes a RAT in F'_{i-1}. After adding $(C_i \vee x)$ to F'_{i-1}, we then perform a sequence of DRAT steps that transform F'_{i-1} to F_{i-1} and that replace $(C_i \vee x)$ by C_i. We thus end up with the accumulated formula F_i. Note that since x is fresh, we could immediately add $(C_i \vee x)$ as a RAT (as there are no resolvents upon x), but this would make it hard to turn $(C_i \vee x)$ into C_i.

To simplify notation, we denote F_{i-1} by F, C_i by C, and ω_i by ω. We also assume that $var(C) \subseteq var(F)$, otherwise we could simply add C by a single RAT addition since there would be no resolvents upon the literals whose variables are in $var(C) \setminus var(F)$. We perform the transformation in five phases:

(1) *Add extended copies of clauses that are touched but not satisfied by ω.* We say that an assignment α *touches* a clause D if $var(\alpha) \cap var(D) \neq \emptyset$. We extend F by adding the clauses $\{(\bar{x} \vee D|_\omega) \mid D \in F \text{ and } D|_\omega \subset D\}$. Notice that $D|_\omega \subset D$ is false if ω satisfies D. As the literal x does not occur in F, all clauses $(\bar{x} \vee D|_\omega)$ are RATs on $\bar{x}$ in F (since there are no resolvents upon $\bar{x}$). We denote the resulting formula by $G^{(1)}$.

(2) *Weaken involved clauses.* We call a clause *involved* if it contains literals that are falsified by ω as well as literals that are satisfied by ω. We weaken every involved clause $E \in F$ by replacing it with $(x \vee E)$. Since every weakening step can be performed by a RUP addition (subsumed clauses are RUPs) followed by a deletion, this leads to valid DRAT steps. We denote the resulting formula by $G^{(2)}$.
This phase only preserves satisfiability as the formula is weakened, while the other phases preserve both satisfiability and unsatisfiability. In order to preserve both satisfiability and unsatisfiability, one can add the clauses corresponding to the implication $x \to \omega$ before weakening the clauses from E to $(x \vee E)$ and after the weakening remove the clauses corresponding to the implication. Adding the implication clauses can be achieved by RAT addition as these clauses have RAT on literal $\bar{x}$. The removal of these clauses can be achieved by RAT deletion as they have RAT on the other literal after the weakening.

(3) *Add the weakened propagation-redundant clause.* We add the clause $(C \vee x)$ to $G^{(2)}$, resulting in $G^{(3)}$. To prove that $(C \vee x)$ is a RAT on x in $G^{(2)}$, we need to show that for every clause $(\bar{x} \vee D) \in G^{(2)}$, the resolvent $(C \vee D)$ of $(C \vee x)$ and $(\bar{x} \vee D)$ is implied by $G^{(2)}$ via unit propagation: The only clauses in $G^{(2)}$ that contain the literal $\bar{x}$ are the ones we added in the first phase, which are of the form $(\bar{x} \vee D|_\omega)$ where $D \in F$; hence the corresponding resolvent that we must show from $G^{(2)}$ via unit propagation is $(C \vee D|_\omega)$. Let α be the assignment precluded by C. Since C is propagation-redundant with respect to F and witness ω, we know that $F|_\alpha \vdash_1 D|_\omega$ since $D \in F$. This is equivalent to $F \vdash_1 (C \vee D|_\omega)$. Now, all clauses of F are also contained in $G^{(2)}$, except for clauses of the form $(x \vee E)$—added in phase (2)—for which F contains the corresponding clauses of the form E. However, since propagation of the negated literals of D on $G^{(2)}$ can derive the unit clause $\bar{x}$, we have that $F \vdash_1 (C \vee D|_\omega)$ implies $G^{(2)} \vdash_1 (C \vee D|_\omega)$.

(4) *Strengthen all weakened clauses.* We remove all occurrences of the literal x from clauses in $G^{(3)}$. With this we reverse the second phase by strengthening the clauses $(E \vee x)$ to E *and* strengthen $(C \vee x)$ to C. First, we introduce the clauses corresponding to the implication $x \to \omega$, i.e., the clauses $\{(\bar{x} \vee l) \mid \omega(l) = 1\}$. Let us show that these clauses are RATs on l in $G^{(3)}$ after making a couple of observations. First, any clause $(\bar{x} \vee D|_\omega)$ introduced in step 1 does not contain the literal $\bar{l}$ with $l \in \omega$. Second, any

clause $E \in F$ containing $\bar{l}$ and satisfied by ω is replaced by the clause $(x \vee E)$ in step 2. Hence, the only clauses that can contain $\bar{l}$ are (i) clauses $D \in F$ which are not satisfied by ω, (ii) clauses of the form $(x \vee E)$ with $\bar{l} \in E$ (as introduced in step 2), and (iii) the clause $(C \vee x)$ introduced in step 3. For a clause D as in the first case, the clause $(\bar{x} \vee D|_\omega)$ is introduced in step 1; the resolvent $(\bar{x} \vee l) \otimes D$ is then subsumed by $(\bar{x} \vee D|_\omega)$, which occurs in $G^{(3)}$ because of step 1. Resolvents with clauses of either the form $(x \vee E)$ in the second case, or the form $(C \vee x)$ in the third case, are tautologies as they contain x and $\bar{x}$. After this, we strengthen all clauses $(x \vee E) \in G^{(3)}$, including $(C \vee x)$, to E as follows: Observe that all clauses $(x \vee E) \in G^{(3)}$ are satisfied by ω and therefore there exists a clause $(\bar{x} \vee l)$ with $l \in E$. We can thus derive E from $(x \vee E)$ by resolving with this clause. We denote the resulting formula by $G^{(4)}$.

(5) *Remove clauses that contain $\bar{x}$.* We remove the clauses $\{(\bar{x} \vee l) \mid \omega(l) = 1\}$ of the implication $x \to \omega$ as well as the clauses of the form $(\bar{x} \vee D)$ that were added in the first phase. We denote the resulting formula by $G^{(5)}$.

6.2 Complexity of the Simulation

We now analyze the worst-case complexity of transforming a PR proof of the form $C_1, \ldots, C_m, \langle \mathsf{a}, C_{m+1}, \omega_{m+1} \rangle, \ldots, \langle \mathsf{a}, C_n, \omega_n \rangle$ into a DRAT proof of the form $C_1, \ldots, C_m, \pi_{m+1}, \ldots, \pi_n$. The number of DRAT steps that are required to simulate a single PR addition step depends on the size of the accumulated formula at the respective proof step. For what follows, $|F_n| = n$ denoted the number of clauses in F_n and let $V = |var(F_n)|$ be the number of variables in F_n. Since a PR proof does not delete clauses, we have $|F_i| = |F_{i-1}| + 1$ and $|var(F_i)| \geq |var(F_{i-1})|$ and thus $|F_i| \leq n$ and $|var(F_i)| \leq V$ for $1 \leq i \leq n$. In the analysis, we ignore clause deletion since the number of clause deletions is bounded by the number of added clauses. Notice that since PR proofs don't contain deletion steps, no original clauses are deleted in the resulting DRAT proof.

In phase (1) of the conversion algorithm, copies of clauses that are reduced but not satisfied by ω_i are added, while in phase (2) clauses are weakened which are reduced and satisfied by ω_i. Since a clause is either satisfied, not satisfied, or untouched by ω_i, the sum of the number of copies and weakened clauses is at most $|F_i| \leq n$. Phase (3) adds a single clause. Phase (4) adds the clauses for the implication $x \to \omega_i$ (at most V steps) and strengthens all weakened clauses (at most n steps). Phase (5) only deletes clauses. Thus the total number of clause additions for all phases in the conversion of a single PR step is bounded by $V + 2n + 1$.

There are fewer than n additions in the PR proof and for each addition we apply the conversion algorithm. Hence the total number of clause addition steps in the DRAT proof is at most $nV + 2n^2 + n$. Since $V \leq n$ for any

interesting PR proof[1], the number of steps in the resulting DRAT proof is in $\mathcal{O}(n^2)$. From this it should be clear that the simulation can be performed in polynomial time.

6.3 Optimizations

Our simulation procedure was designed to result in compact DRAT proofs using only one new variable, while focusing on converting any DPR derivation into a DRAT derivation. The procedure can be further optimized to reduce the size of the resulting DRAT proof.

Witness minimization. In some situations, only a subset of the involved clauses needs to be weakened (phase 2) and strengthened (phase 4). Weakening of involved clauses is required to make sure that the clauses of the form $(\bar{x} \vee l)$, where ω satisfies l, are RATs on l in $G^{(3)}$ in phase (4) of the simulation algorithm. However, some of the clauses $(\bar{x} \vee l)$ may be implied via unit propagation by others (and do not require to be a RATs on l). This situation occurs when a subset ω' of the witness ω implies ω via unit propagation. We thus minimize ω by searching for the smallest witness $\omega' \subseteq \omega$ such that ω' implies ω via unit propagation, i.e., $F|_{\omega'} \vdash_1 (l_1) \wedge \cdots \wedge (l_n)$ for $\omega = l_1 \dots l_n$. Only clauses reduced by ω' and satisfied by ω need to be weakened in phase (2) and strengthened in (4).

Avoiding copying. In some cases, which we describe in the following, we can avoid copying the clauses that are touched but not satisfied by the witness, meaning that we can skip phase (1) and (5) of the simulation algorithm: Let α be the assignment precluded by the PR clause C to be added, let ω be the witness, and let ω' be the minimized witness as discussed above. If the following two conditions hold, we can avoid clause copying: First, there is no literal l that is satisfied by α but falsified by ω'.[2] Second, for each literal l that is satisfied by ω', the unit clause (l) should be a RAT on l in the current formula without the involved clauses under α. Although these conditions seem very restrictive, they apply often in the PR proofs used in our empirical evaluation. This optimization removes phases (1) and (5), and modifies (2), (3), and (4). We denote the modified phases by phase (i), (ii), and (iii), respectively:

(i) *Weaken involved clauses.* We now call a clause *involved* if it contains literals that are falsified by the minimized witness ω' as well as literals that are satisfied by the original witness ω. In this phase, we replace each involved clause E by the clause $(x \vee E)$, which is a valid RUP addition. We denote the resulting formula by $G^{(i)}$.

[1] A formula with more variables than clauses can be reduced to a smaller satisfiability-equivalent formula with more clauses than variables in polynomial time.

[2] Recall that, by definition, there always exists a literal that is satisfied by α but falsified by ω, hence this condition can only be true if ω was minimized to ω'.

(ii) *Add the weakened propagation-redundant clause.* Add the clause $(C \vee x)$. Since no clause contains the literal $\bar{x}$, this is a valid RAT addition. We denote the resulting formula by $G^{(ii)}$.

(iii) *Strengthen all weakened clauses.* We now remove all occurrences of the literal x from clauses in $G^{(ii)}$. With this, we reverse the second phase (i) by strengthening $(E \vee x)$ to E *and* strengthen $(C \vee x)$ to C. First, we introduce the clauses corresponding to the implication $x \to \omega'$, i.e., the clauses $\{(\bar{x} \vee l) \mid \omega'(l) = 1\}$. These clauses are RATs on l in $G^{(ii)}$: Assume a clause D contains $\bar{l}$, i.e., D contains a literal that is falsified by ω'. Then, D is either satisfied by ω or not. If D is satisfied by ω, then it is of the form $(x \vee E)$ (introduced in phase i), and thus the resolvent with $(\bar{x} \vee l)$ is a tautology. If D is not satisfied by ω, then it is implied by unit propagation under the current formula under α because of the second condition above. After adding these clauses, we strengthen $(C \vee x)$ to C and all clauses $(x \vee E) \in G^{(ii)}$ to E. Observe that all clauses $(x \vee E) \in G^{(ii)}$, including $(C \vee x)$, are satisfied by ω and therefore there exists a clause $(\bar{x} \vee l)$ with $l \in E$. Resolving with the clauses $(\bar{x} \vee l)$, we can therefore remove all literals x. We denote the resulting formula, which is equal to $G \wedge C$, by $G^{(iii)}$.

7 Experimental Evaluation

We implemented our simulation procedures as dedicated tools, called `drat2er` (for the transformation from DRAT to extended resolution) and `pr2drat` (for the transformation from DPR to DRAT).[3] We then evaluated the simulation tools on existing DPR proofs for the *pigeon-hole formulas*, *two-pigeons-per-hole formulas* [3], and *Tseitin formulas* [30, 4]. The pigeon-hole formulas (`hole*`) ask whether $n+1$ pigeons can be placed into n holes such that each hole contains at most one pigeon. Similarly, the two-pigeons-per-hole formulas (`tph*`) ask whether $2n+1$ pigeons can be placed into n holes with at most two pigeons per hole. Finally, the Tseitin formulas (`Urquhart*`) encode a parity problem on graphs.

We selected the proofs of these formulas for two reasons. First, all three formula families are hard for resolution, meaning that they admit only resolution proofs whose size is exponential with respect to the formula [9, 32]. Second, out of all DRAT proofs we are aware of, the DRAT proofs in our experiments have the highest ratio of proper-RAT-to-RUP-instructions, meaning that the transformation from DRAT to extended resolution can offer insight into a worst-case scenario regarding existing proofs. Table 1 shows the results of our experiments. Although the extended-resolution proofs are clearly larger than the corresponding DRAT proofs, the blow-up is far from the theoretical

[3] The tool `drat2er` is available at https://github.com/benjaminkiesl/drat2er and the tool `pr2drat` is available at http://www.cs.cmu.edu/~mheule/pr2drat/. The formulas and proofs are available at https://github.com/marijnheule/drat2er-proofs.

Table 1. A size comparison of DPR, DRAT, and ER proofs of formulas that are hard for resolution. Column headers refer to the numbers of variables (#var), clauses (#cls), clause additions (#add), added definitions (#def), and resolution steps (#res).

	input		DPR	DRAT	ER	
formula	#var	#cls	#add	#add	#def	#res
`hole20`	420	4221	2870	26 547	18 162	282 471
`hole30`	930	13 981	9455	89 827	61 962	1 393 411
`hole40`	1640	32 841	22 140	213 107	147 562	4 344 126
`hole50`	2550	63 801	42 925	416 387	288 962	10 517 116
`tph8`	136	5457	1156	25 204	13 931	1 093 959
`tph12`	300	27 625	3950	127 296	68 645	11 688 956
`tph16`	528	87 329	9416	401 004	212 847	63 391 635
`tph20`	820	213 241	18 450	976 376	512 841	236 415 141
`Urquhart-s5-b1`	106	714	620	28 189	8320	102 293
`Urquhart-s5-b2`	107	742	606	32 574	9020	123 943
`Urquhart-s5-b3`	121	1116	692	41 230	11 404	188 875
`Urquhart-s5-b4`	114	888	636	37 978	10 497	171 576

Table 2. Small existing ER proofs of pigeon-hole formulas and Tseitin formulas.

	ER by Cook [5]	
formula	#def	#res
`hole20`	2660	160 151
`hole30`	8990	810 161
`hole40`	21 320	2 560 171
`hole50`	41 650	6 250 181

	ER by EBDDRES [19]	
formula	#def	#res
`Urquhart-s5-b1`	11 054	39 702
`Urquhart-s5-b2`	12 684	45 389
`Urquhart-s5-b3`	28 358	100 585
`Urquhart-s5-b4`	16 295	58 552

worst case. As we already selected proofs with many proper RAT instructions, we imagine that the growth is even smaller on proofs with a modest number of RAT instructions. For a pigeon-hole formula `holeX`, the increase in size is roughly the factor `X`. For the two-pigeons-per-hole formulas, the growth is larger. This can be explained by the high clauses-to-variables ratio. Finally, for the Tseitin formulas, the growth lies between a factor of 20 and 30.

As a comparison, Table 2 shows the smallest extended-resolution proofs of the pigeon-hole formulas and of the Tseitin formulas known to us. The proofs of the pigeon-hole formulas were manually constructed by Cook [5] whereas the proofs of the Tseitin formulas were produced using the tool EBDDRES 1.2 [19]. To the best of our knowledge, there is only one tool supporting extended resolution that was able to solve one of the selected two-pigeons-per-hole formulas: EBDDRES 1.1 [29]. It generated an extended-resolution proof with 2 638 385 definitions and 18 848 004 resolution steps for the formula `tph8`.

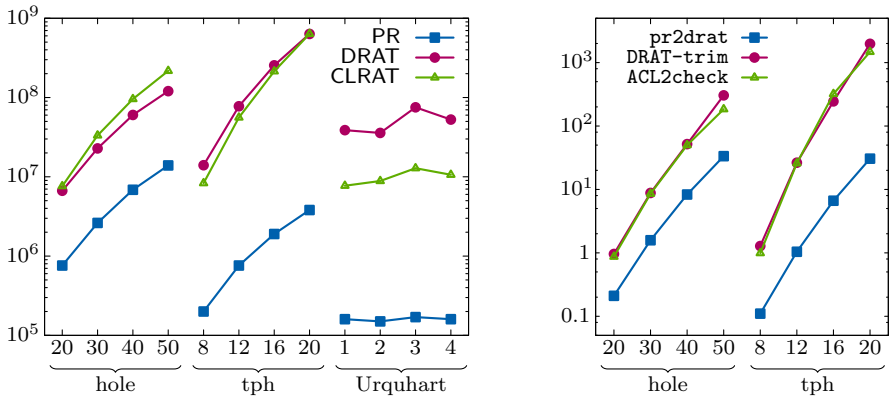


Fig. 5. Certification of PR proofs using `pr2drat`, `DRAT-trim`, and the formally verified checker `ACL2check`. Left the sizes of proofs in the PR, DRAT, and CLRAT formats are shown in bytes and right the proof conversion and checking times are in seconds. No times are shown for the Urquhart instances as all times were less than a second.

7.1 Verified PR Proof Checking

Our tool `pr2drat` can be used to validate PR proofs with formally verified tools and thereby increase the confidence in their correctness. The tool chain works as follows: Given a formula F and an alleged PR proof P_{PR} of F, `pr2drat` converts P_{PR} into a DRAT proof P_{DRAT}. Afterwards, we use the `DRAT-trim` tool to convert P_{DRAT} into a CLRAT (compressed linear RAT) proof P_{CLRAT}. CLRAT proofs can be efficiently checked using formally verified checkers [7]. We used the verified checker `ACL2check` [12] to certify that P_{CLRAT} is a valid proof of unsatisfiability of F. Notice that the correctness of the tools `pr2drat` and `DRAT-trim` has not been formally verified and thus they could possibly turn an invalid proof into a valid proof or vice versa.

Figure 5 shows the results of applying this tool chain on the benchmark suite. The `pr2drat` tool was able to convert each PR proof into a DRAT proof in less than a minute, and half of the proofs in even less than a second. The runtimes of `DRAT-trim` and `ACL2check` are one to two orders of magnitude higher than for `pr2drat`. Thus, `pr2drat` adds little overhead to the tool chain. The sizes of the DRAT and CLRAT proofs are comparable. However, these proofs are different since `DRAT-trim` (1) removes redundant clause additions, (2) includes hints to speedup verified checking, and (3) compresses proofs. The effect of 1 depends on proof quality, 2 increases the size of proofs of small hard problems by roughly a factor of four, and 3 reduces size to 30% of the uncompressed proofs. The difference between the DRAT and CLRAT proofs therefore indicates how much redundancy was removed: For the pigeon-hole proofs, there is hardly any redundancy added. For the two-pigeons-per-hole proofs, only a modest amount is added, and for the Tseitin proofs a lot of

redundancy is added. Notice that runtimes of the verified checker `ACL2check` are comparable to the C-based checker `DRAT-trim`.

8 Conclusion

We showed different simulations between propositional proof systems. The first simulation transforms DRAT proofs into extended-resolution proofs whereas the second simulation transforms DPR proofs into DRAT proofs. Together, these two simulations show how extended resolution is related to modern propagation-based proof systems used in practical SAT solving. In addition, we showed how blocked-clause addition can be used to simulate the addition of RATs without the introduction of new variables. Our results provide us with a better understanding of DRAT and DPR as well as of extended resolution. We now know how extended resolution can mimic the reasoning steps of these modern proof systems.

To evaluate the increase in size caused by our simulations, we implemented them and performed experiments on existing DRAT and DPR proofs of hard formulas. Even though the size increase could be considerable in theory, in practice it is still feasible. Especially our simulation tool of DPR by DRAT allows to certify the correctness of DPR proofs by first transforming them to DRAT and then using formally verified proof checkers.

References

1. Alekhnovich M (2004) Mutilated chessboard problem is exponentially hard for resolution. Theoretical Computer Science 310(1-3):513–525
2. Baaz M, Leitsch A (2011) Methods of Cut-Elimination. No. 3 in Trends in Logic, Springer Netherlands
3. Biere A (2013) Two pigeons per hole problem. In: Proc. of SAT Competition 2013: Solver and Benchmark Descriptions, p 103
4. Chatalic P, Simon L (2000) Multi-resolution on compressed sets of clauses. In: Proc. of the 12th IEEE Int. Conference on Tools with Artificial Intelligence (ICTAI 2000), pp 2–10
5. Cook SA (1976) A short proof of the pigeon hole principle using extended resolution. SIGACT News 8(4):28–32
6. Cook SA, Reckhow RA (1979) The relative efficiency of propositional proof systems. The Journal of Symbolic Logic 44(1):pp. 36–50
7. Cruz-Filipe L, Heule MJH, Jr WAH, Kaufmann M, Schneider-Kamp P (2017) Efficient certified RAT verification. In: Proc. of the 26th Int. Conference on Automated Deduction (CADE-26), Springer, LNCS, vol 10395, pp 220–236
8. Dantchev SS, Riis S (2001) "Planar" tautologies hard for resolution. In: Proc. of the 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001), IEEE Computer Society, pp 220–229

9. Haken A (1985) The intractability of resolution. Theoretical Computer Science 39:297–308
10. Heule MJH, Biere A (2018) What a difference a variable makes. In: Proc. of the 24th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2018), Springer, LNCS, vol 10806, pp 75–92
11. Heule MJH, Kullmann O, Marek VW (2016) Solving and verifying the Boolean Pythagorean Triples problem via Cube-and-Conquer. In: Proc. of the 19th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2016), Springer, Cham, LNCS, vol 9710, pp 228–245
12. Heule MJH, Hunt Jr WA, Kaufmann M, Wetzler ND (2017) Efficient, verified checking of propositional proofs. In: Proc. of the 8th Int. Conference on Interactive Theorem Proving (ITP 2017), Springer, LNCS, vol 10499, pp 269–284
13. Heule MJH, Kiesl B, Biere A (2017) Short proofs without new variables. In: Proc. of the 26th Int. Conference on Automated Deduction (CADE-26), Springer, Cham, LNCS, vol 10395, pp 130–147
14. Heule MJH, Kiesl B, Seidl M, Biere A (2017) PRuning through satisfaction. In: Proc. of the 13th Haifa Verification Conference (HVC 2017), Springer, LNCS, vol 10629, pp 179–194
15. Heule MJH, Kiesl B, Biere A (2019) Encoding redundancy for satisfaction-driven clause learning. In: Proc. of the 25th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2019), Springer, LNCS, vol 11427, pp 41–58
16. Heule MJH, Kiesl B, Biere A (2019) Strong extension-free proof systems. Journal of Automated Reasoning
17. Järvisalo M, Biere A, Heule MJH (2010) Blocked clause elimination. In: Proc. of the 16th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010), Springer, Heidelberg, LNCS, vol 6015, pp 129–144
18. Järvisalo M, Heule MJH, Biere A (2012) Inprocessing rules. In: Proc. of the 6th Int. Joint Conference on Automated Reasoning (IJCAR 2012), Springer, Heidelberg, LNCS, vol 7364, pp 355–370
19. Jussila T, Sinz C, Biere A (2006) Extended resolution proofs for symbolic SAT solving with quantification. In: Proc. of the 9th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2006), Springer, LNCS, vol 4121, pp 54–60
20. Kiesl B, Rebola-Pardo A, Heule MJH (2018) Extended resolution simulates DRAT. In: Proc. of the 9th Int. Joint Conference on Automated Reasoning (IJCAR 2018), Springer, LNCS, vol 10900, pp 516–531
21. Konev B, Lisitsa A (2015) Computer-aided proof of Erdős discrepancy properties. Artificial Intelligence 224(C):103–118
22. Kullmann O (1999) On a generalization of extended resolution. Discrete Applied Mathematics 96-97:149–176
23. Lee CT (1967) A completeness theorem and a computer program for finding theorems derivable from given axioms. PhD thesis, University of Cal-

ifornia, Berkeley
24. Marques-Silva JP, Sakallah KA (1999) GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers 48(5):506–521
25. McCarthy J (1964) A tough nut for proof procedures. Memo 16, Stanford Artificial Intelligence Project
26. Moskewicz MW, Madigan CF, Zhao Y, Zhang L, Malik S (2001) Chaff: Engineering an efficient SAT solver. In: Proc. of the 38th Design Automation Conference (DAC 2001), ACM, pp 530–535
27. Philipp T, Rebola-Pardo A (2017) Towards a semantics of unsatisfiability proofs with inprocessing. In: Proc. of the 21st Int. Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-21), EasyChair, EPiC Series in Computing, vol 46, pp 65–84
28. Rebola-Pardo A, Suda M (2018) A theory of satisfiability-preserving proofs in SAT solving. In: Proc. of the 22nd Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-22), EasyChair, EPiC Series in Computing, vol 57, pp 583–603
29. Sinz C, Biere A (2006) Extended resolution proofs for conjoining BDDs. In: Proc. of the 1st Int. Computer Science Symposium in Russia (CSR 2006), Springer, Heidelberg, LNCS, vol 3967, pp 600–611
30. Tseitin GS (1968) On the complexity of derivation in propositional calculus. Studies in Mathematics and Mathematical Logic 2:115–125
31. Urquhart A (1987) Hard examples for resolution. Journal of the ACM 34(1):209–219
32. Urquhart A (1995) The complexity of propositional proofs. The Bulletin of Symbolic Logic 1(4):425–467
33. Van Gelder A (2008) Verifying RUP proofs of propositional unsatisfiability. In: Proc. of the 10th Int. Symposium on Artificial Intelligence and Mathematics (ISAIM 2008)
34. Van Gelder A (2012) Producing and verifying extremely large propositional refutations. Annals of Mathematics and Artificial Intelligence 65(4):329–372
35. Wetzler ND, Heule MJH, Hunt Jr WA (2014) DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In: Proc. of the 17th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2014), Springer, Cham, LNCS, vol 8561, pp 422–429