Extended Resolution Simulates DRAT [⋆]

Benjamin Kiesl[1], Adrián Rebola-Pardo[1], and Marijn J.H. Heule[2]

[1] Institute of Logic and Computation, TU Wien
[2] Department of Computer Science, The University of Texas at Austin

Abstract. We prove that extended resolution—a well-known proof system introduced by Tseitin—polynomially simulates DRAT, the standard proof system in modern SAT solving. Our simulation procedure takes as input a DRAT proof and transforms it into an extended-resolution proof whose size is only polynomial with respect to the original proof. Based on our simulation, we implemented a tool that transforms DRAT proofs into extended-resolution proofs. We ran our tool on several benchmark formulas to estimate the increase in size caused by our simulation in practice. Finally, as a side note, we show how blocked-clause addition—a generalization of the extension rule from extended resolution—can be used to replace the addition of resolution asymmetric tautologies in DRAT without introducing new variables.

1 Introduction

Propositional logic presents us with an intricate problem: Does there exist a polynomially-bounded proof system for the unsatisfiable propositional formulas? In other words, can the unsatisfiability of formulas be certified in a compact way? Although we still don't know the answer, the attempts to solve this problem have led to a variety of interesting results in the area of proof complexity (for an excellent survey, see [18]). Many of these results had, and continue to have, a direct impact on automated reasoning.

Already in 1985, Haken [6] proved that the resolution proof system, which is well-suited for mechanization, does not admit polynomial-size proofs for all unsatisfiable formulas. However, by adding a simple rule that allows the introduction of definitions over new variables, Tseitin [17] turned resolution into an exponentially stronger proof system known as *extended resolution.* Up to this day, there are no known exponential lower-bounds on the size of extended-resolution proofs and so it is seen as one of the most powerful proof systems.

While this might convince a theoretician, it seemingly hasn't impressed the practitioners in SAT solving. These practitioners aim at developing tools that can decide the satisfiability of propositional formulas as efficiently as possible, and for this, they need proof systems that succinctly express the techniques used by their tools. Skeptical that extended resolution could meet their needs, they

[⋆] This work has been supported by the National Science Foundation under grant CCF-1618574, by the Austrian Science Fund (FWF) under project W1255-N23, and by Microsoft Research through its PhD Scholarship Programme.

came up with several proof systems of which DRAT [21] has become their defacto standard. For instance, participants in the annual SAT competition must produce DRAT proofs and also recent proofs of open mathematical problems, including the Erdős Discrepancy Conjecture [12], were provided in DRAT.

The DRAT proof system generalizes extended resolution insofar as every extended-resolution proof can be seen as a DRAT proof, but beyond that, DRAT allows additional techniques. While in extended resolution we show the unsatisfiability of a formula by successively deriving more and more consequences, in DRAT we iteratively modify a formula in satisfiability-preserving ways. To keep proof checking practical, DRAT allows only the derivation of specific facts that fulfill an efficiently-checkable syntactic criterion—so-called *resolution asymmetric tautologies* [10] (see Definition 4 on page 5).

Although its additional features make DRAT suitable for SAT solving, it remained unclear whether these features can indeed cause exponential gains in expressivity. In this paper, we show that they do *not*. To this end, we prove in a constructive way that extended resolution simulates DRAT polynomially, i.e., we show how every DRAT proof can be feasibly transformed into an extended-resolution proof. This confirms the expected proof-complexity landscape where all top-tier proof systems—including extended resolution, DRAT, and extended Frege systems [18]—are essentially equivalent.

Rounding off the picture, we show how blocked-clause addition [13]—a generalization of the extension rule from extended resolution—can be used to replace the addition of resolution asymmetric tautologies in DRAT without introducing new variables. In combination with recent simulation results regarding DRAT and newer proof systems [7,8], our paper thus bridges the gap between proof systems from the present and from the past.

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

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* $\bar{a}$ of a literal a is defined as $\bar{a} = \bar{x}$ if $a = x$ and $\bar{a} = x$ if $a = \bar{x}$. 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, otherwise it is *partial.* 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 $F\,|\,\alpha = \{C\,|\,\alpha \mid C \in F \text{ and } \alpha \text{ does not satisfy } C\}$. The result of applying the *unit-clause rule* to a formula F is the formula $F\,|\,a$ with (a) being 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 $F\,|\,ab = \bot$.

We define proof systems and polynomial simulations following Cook and Reckhow [5]:

Definition 1. *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 2. *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* and DRAT.

3 Extended Resolution (ER) and DRAT

An extended-resolution proof as well as a DRAT proof of a formula F are sequences of the form $C_1, \dots, C_m, I_{m+1}, \dots, I_n$ 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 a pair $\langle \mathsf{a}, C\rangle$ where C is a clause; 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 propositional definition of the form $x \leftrightarrow p \vee (c_1 \wedge \dots \wedge c_k)$ where x is a variable not occurring in any earlier instructions of the proof and $p, c_1, \dots, c_k$ are literals where $var(x), var(p), var(c_1), \dots, var(c_k)$ are pairwise distinct. Converting such a definition to CNF yields 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:

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

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 [17], 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. [4,6,16]).

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 now proceed to introduce these notions. We start by defining RUPs (short for *reverse unit propagation*) [20]:

Definition 3. *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.

As an example, $F = (\bar{a} \vee c) \wedge (\bar{b} \vee \bar{c})$ implies $(\bar{a} \vee \bar{b})$ via unit propagation since unit propagation on $F \wedge (a) \wedge (b)$ derives both (c) and $(\bar{c})$, which leads to a conflict. 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. Now, a RAT is a clause for which all resolvents upon one of its literals are RUPs [10]:

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. □

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 $\perp$ 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 [9] (see Section 6), which itself is a particular case of RAT addition. We thus regard DRAT as a generalization of extended resolution.

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

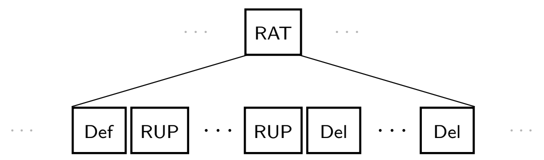

Fig. 1. We transform a RAT addition into a definition introduction (Def), followed by RUP additions and clause deletions (Del).

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.

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

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 $p \vee c_1 \vee \cdots \vee c_k$. Intuitively, this definition follows the correctness proof of RAT clause addition from [10]: 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 correct 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 2. Suppose 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 $\bar{x} \vee b$ is a RUP in $F_{i-1} \cup \{\bar{x} \vee p \vee \bar{a},\ x \vee \bar{p},\ x \vee a\}$: 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 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 also a RUP in every

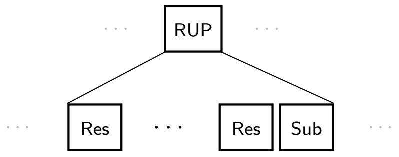

Fig. 2. We transform a RUP addition into a sequence of resolution steps (Res) followed by a single subsumed-clause addition (Sub).

superset of F). Likewise, 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).

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 now 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 [19,15].

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 $a \lor \bar{a} \lor D$, 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 $F \land \bar{C}$ 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 $a_1, \dots, a_n$ such that the unit clause (a_i) occurs in $F\,|\,\bar{C}a_1 \dots a_{i-1}$ for each $1 \leq i \leq n$, and the empty clause $\bot$ occurs in $F\,|\,\bar{C}a_1 \dots a_n$. 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 $D_1, \dots, D_{n+1} \in F$ with the following properties: (I) the clause $D_i\,|\,\bar{C}a_1 \dots a_{i-1}$ is the unit clause (a_i) for $1 \leq i \leq n$, (II) D_i is not satisfied by $\bar{C}a_1 \dots a_{i-1}$ for $1 \leq i \leq n+1$, and (III) the clause $D_{n+1}\,|\,\bar{C}a_1 \dots a_n$ is the empty clause.

Algorithm 1 uses the clauses $D_1, \dots, D_{n+1}$ as follows: It starts with the last clause, D_{n+1}, and step-by-step resolves it with the clauses $D_n, \dots, D_1$ until it obtains a clause C_1 that subsumes C. Using C_1, we can then derive C with a subsumed-clause addition. Example 3 illustrates the execution of the algorithm.

```
$C_{n+1} \leftarrow D_{n+1}$
**for** $i = n, \ldots, 1$ **do**
  **if** $\bar{a}_i \in C_{i+1}$ **then** $C_i \leftarrow D_i \otimes_{a_i} C_{i+1}$
  **else** $C_i \leftarrow C_{i+1}$
```

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

Example 3. Consider the clause $C = a \lor b$ and $F = D_1 \land D_2 \land D_3 \land D_4$ where:

$$D_1 = a \lor c \qquad D_2 = a \lor \bar{c} \lor d \qquad D_3 = \bar{d} \lor e \qquad D_4 = \bar{d} \lor \bar{e}$$

The clause C is a RUP in F because unit propagation derives a conflict on $F \land (\bar{a}) \land (\bar{b})$, 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 (* marks unit literals):

$$\dfrac{\overbrace{a \lor c^*}^{D_1} \qquad \dfrac{\overbrace{a \lor \bar{c} \lor d^*}^{D_2} \qquad \dfrac{\overbrace{\bar{d} \lor e^*}^{D_3} \qquad \overbrace{\bar{d} \lor \bar{e}}^{D_4}}{\bar{d}}}{a \lor \bar{c}}}{\underbrace{a}_{C_1}}$$

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

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 C_{i+1} does not contain $\bar{a}_i$, 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. $\square$

The following statement, which is a variant of Theorem 2 in [15] as well as of the Theorem of Lee [14], 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, 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., [1]) 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 $C_1 \lor x$ and $C_2 \lor \bar{x}$, 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 $D_1 \lor D_2$ 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. This is polynomial, and can only reduce the size of the input. The resulting derivation may contain redundant parts such as unused subderivations, but these do not affect our analysis and can be easily removed. After eliminating all subsumed-clause additions, we finally obtain an extended-resolution proof.

Example 4. The following proof tree includes the subsumed-clause additions 1 and 2. A subclause labeling as above is given in brackets; it can be read off as a resolution proof. Clauses in red are redundant and can be skipped from the resulting proof.

$$\dfrac{\dfrac{a \lor b \quad [a \lor b] \qquad {}^{1}\dfrac{\bar{b} \quad [\bar{b}]}{a \lor \bar{b} \quad [\bar{b}]^*}}{a \quad [a]} \qquad \dfrac{\dfrac{\bar{a} \lor c \quad [\bar{a} \lor c]^* \qquad {}^{2}\dfrac{d \quad [d]}{\bar{c} \lor d \quad [d]^*}}{\bar{a} \lor d \quad [d]^*} \qquad \bar{a} \lor \bar{d} \quad [\bar{a} \lor \bar{d}]}{\bar{a} \quad [\bar{a}]}}{\bot \quad [\bot]}$$

After dropping the marked (*) clauses, the result is the following proof:

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

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. Let the original DRAT proof be $P = C_1, \dots, C_m, I_{m+1}, \dots, I_n$. 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 $i \in m+1, \dots, n$, 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 6). For the elimination of a single RAT addition of a clause $p \vee c_1 \vee \dots \vee c_k$, we first add the definition $x \leftrightarrow p \vee (\bar{c}_1 \wedge \dots \wedge \bar{c}_k)$. 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 representing them explicitly shrinks the resulting extended-resolution proof significantly. As we will see in Section 5, 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) [7], we obtain the complexity landscape depicted in Fig. 3.

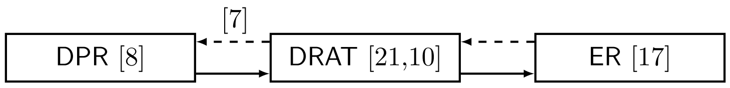

Fig. 3. A dashed line from X to Y means that X simulates Y polynomially. A solid line from X to Y means that every Y proof can be regarded as an X proof.

5 Experimental Evaluation

We implemented our simulation procedure as an extension of the proof checker `DRAT-trim`.[3] We then evaluated the simulation tool on existing DRAT proofs for the *pigeon-hole formulas*, *two-pigeons-per-hole formulas* [2], and *Tseitin formulas* [17,3]. 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 DRAT proofs of these formulas for three reasons. First, out of all DRAT proofs we are aware of, they have the highest ratio of proper-RAT-to-RUP-instructions and so the transformation from DRAT to extended resolution can offer insight into a worst-case scenario regarding existing proofs. Second, the proofs originate from a transformation of DPR proofs to DRAT proofs [7]. We thus also see what happens when we transform the more general DPR proofs, and not only DRAT proofs, to extended resolution. Third, all three formula families are hard for resolution, meaning that they admit only resolution proofs whose size is exponential with respect to the formula [6,18].

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 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 [4] whereas the proofs of the Tseitin formulas were produced using the tool EBDDRES 1.2 [11]. 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 [16]. It generated an extended-resolution proof with 2 638 385 definitions and 18 848 004 resolution steps for the formula `tph8`.

[3] The simulation tool, checkers, formulas, and proofs discussed in this section are available on `http://www.cs.utexas.edu/~marijn/drat2er`.

Table 1. A size comparison of DPR, DRAT, and ER proofs of formulas that are hard for resolution. We generated the ER proofs from existing DRAT proofs [7]. 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 [4]			ER by EBDDRES [11]	
formula	#def	#res	*formula*	#def	#res
`hole20`	2660	160 151	`Urquhart-s5-b1`	11 054	39 702
`hole30`	8990	810 161	`Urquhart-s5-b2`	12 684	45 389
`hole40`	21 320	2 560 171	`Urquhart-s5-b3`	28 358	100 585
`hole50`	41 650	6 250 181	`Urquhart-s5-b4`	16 295	58 552

6 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 [6]. We now show how *blocked-clause addition*, introduced by Kullmann [13] 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 [13]:

Definition 5. *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 5. 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 clause $D \vee \bar{p} \in F_{i-1}$ such that the 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 clause $D \vee \bar{p} \in F_{i-1}$ such that the resolvent with C upon p is not a tautology, replace $D \vee \bar{p}$ by the clause 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

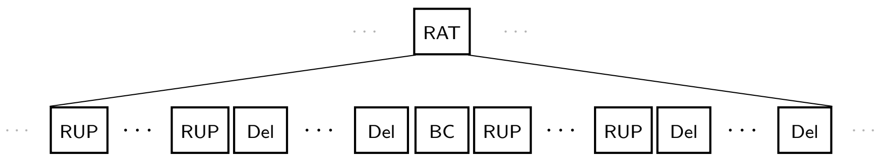

Fig. 4. We transform a RAT addition into a sequence consisting of RUP additions, clause deletions (Del), and a single blocked-clause addition (BC).

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 6. Consider the formula $F = \{\bar{p},\ a \vee b \vee c,\ \bar{c} \vee d,\ \bar{d},\ \bar{a} \vee e,\ \bar{b} \vee e\}$ 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.

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 $F \cup \{C\}$. □

7 Conclusion

We showed how every DRAT proof can be feasibly transformed into an extended-resolution proof. To evaluate the increase in size caused by our simulation, we implemented it and performed experiments on existing DRAT proofs for hard formulas. The experiments revealed that the obtained proofs are far smaller than the theoretical worst case and that they are also not much larger than existing extended-resolution proofs of the same formulas. We imagine that the size of the proofs could be reduced even further by performing additional compression steps, which is a direction for future work.

In addition, we showed how blocked-clause addition can be used to simulate the addition of resolution asymmetric tautologies (RATs) without the introduction of new variables. Our results provide us with a better understanding of both DRAT and extended resolution. We now know how extended resolution can mimic the reasoning steps of DRAT. Moreover, our transformations illustrate that the addition of RATs in DRAT combines in an elegant way the benefits of resolution, subsumption, and blocked-clause addition. We thus believe that DRAT is still the preferable proof system for practical SAT solving, even though it offers no exponential gains in expressivity compared to extended resolution.

References

1. Baaz, M., Leitsch, A.: Methods of Cut-Elimination. No. 3 in Trends in Logic, Springer Netherlands (2011)
2. Biere, A.: Two pigeons per hole problem. In: Proc. of SAT Competition 2013: Solver and Benchmark Descriptions. p. 103 (2013)

3. Chatalic, P., Simon, L.: 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 (2000)
4. Cook, S.A.: A short proof of the pigeon hole principle using extended resolution. SIGACT News 8(4), 28–32 (Oct 1976)
5. Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. Journal of Symbolic Logic 44(1), 3650 (1979)
6. Haken, A.: The intractability of resolution. Theoretical Computer Science 39, 297–308 (1985)
7. Heule, M.J.H., Biere, A.: What a difference a variable makes. To appear in: Proc. of the 24th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2018) (2018)
8. Heule, M.J.H., Kiesl, B., Biere, A.: Short proofs without new variables. In: Proc. of the 26th Int. Conference on Automated Deduction (CADE-26). LNCS, vol. 10395, pp. 130–147. Springer, Cham (2017)
9. Järvisalo, M., Biere, A., Heule, M.J.H.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) Proc. of the 16th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010). LNCS, vol. 6015, pp. 129–144. Springer, Heidelberg (2010)
10. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Proc. of the 6th Int. Joint Conference on Automated Reasoning (IJCAR 2012). LNCS, vol. 7364, pp. 355–370. Springer, Heidelberg (2012)
11. Jussila, T., Sinz, C., Biere, A.: Extended resolution proofs for symbolic SAT solving with quantification. In: Proc. of the 9th Int. Conference on Theory and Applications of Satisfiability Testing. LNCS, vol. 4121, pp. 54–60. Springer (2006)
12. Konev, B., Lisitsa, A.: Computer-aided proof of Erdős discrepancy properties. Artificial Intelligence 224(C), 103–118 (Jul 2015)
13. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics 96-97, 149–176 (1999)
14. Lee, C.T.: A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms. Ph.D. thesis (1967)
15. Philipp, T., Rebola-Pardo, A.: 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). EPiC Series in Computing, vol. 46, pp. 65–84. EasyChair (2017)
16. Sinz, C., Biere, A.: Extended resolution proofs for conjoining BDDs. In: Proc. of the 1st Int. Computer Science Symposium in Russia (CSR 2006). LNCS, vol. 3967, pp. 600–611. Springer, Heidelberg (2006)
17. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Mathematics and Mathematical Logic 2, 115–125 (1968)
18. Urquhart, A.: The complexity of propositional proofs. The Bulletin of Symbolic Logic 1(4), 425–467 (1995)
19. Van Gelder, A.: Verifying RUP proofs of propositional unsatisfiability. In: Proc. of the 10th Int. Symposium on Artificial Intelligence and Mathematics (ISAIM 2008) (2008)
20. Van Gelder, A.: Producing and verifying extremely large propositional refutations. Annals of Mathematics and Artificial Intelligence 65(4), 329–372 (2012)
21. Wetzler, N.D., Heule, M.J.H., Hunt Jr., W.A.: 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). LNCS, vol. 8561, pp. 422–429. Springer, Cham (2014)