What a Difference a Variable Makes*

Marijn J.H. Heule[1] and Armin Biere[2]

[1] Department of Computer Science, The University of Texas at Austin
[2] Institute for Formal Models and Verification, JKU Linz

Abstract. We present an algorithm and tool to convert derivations from the powerful recently proposed PR proof system into the widely used DRAT proof system. The PR proof system allows short proofs without new variables for some hard problems, while the DRAT proof system is supported by top-tier SAT solvers. Moreover, there exist efficient, formally verified checkers of DRAT proofs. Thus our tool can be used to validate PR proofs using these verified checkers. Our simulation algorithm uses only one new Boolean variable and the size increase is at most quadratic in the size of the propositional formula and the PR proof. The approach is evaluated on short PR proofs of hard problems, including the well-known pigeon-hole and Tseitin formulas. Applying our tool to PR proofs of pigeon-hole formulas results in short DRAT proofs, linear in size with respect to the size of the input formula, which have been certified by a formally verified proof checker.

1 Introduction

Satisfiability (SAT) solvers are powerful tools for many applications in formal methods and artificial intelligence [3,9]. Arguably the most effective new techniques in recent years are based on *inprocessing* [21,25]: Interleaving preprocessing techniques and conflict-driven clause learning (CDCL) [26]. Several powerful inprocessing techniques, such as symmetry breaking [6,1] and blocked clause addition [23], do not preserve logical equivalence and cannot be expressed compactly using classical resolution proofs [30]. The RAT proof system [14] was designed to express such techniques succinctly and facilitate efficient proof validation. All top-tier SAT solvers support proof logging in the DRAT proof system [12], which extends the RAT proof system with clause deletion.

More recently a ground-breaking paper [8] presented at TACAS'17 showed how to efficiently certify huge propositional proofs of unsatisfiability by proof checkers, which are formally verified by theorem provers, such as ACL2 [7], Coq [8,7], and Isabelle/HOL [24]. These developments are clearly a break-through in SAT solving. They allow us to have the same trust in the correctness of the results produced by a highly tuned state-of-the-art SAT solver as into those claims deduced with proof producing theorem provers. We can now use SAT solvers as part of such fully trusted proof generating systems.

* Supported by the National Science Foundation (NSF) under grant CCF-1526760 and by the Austrian Science Fund (FWF) under project S11409-N23 (RiSE).

On the other hand, with even more powerful proof systems we can produce even smaller proofs. The goal in increasing the power of proof systems is to cover additional not yet covered but existing reasoning techniques compactly, e.g., algebraic reasoning, but also to provide a framework for investigating new inprocessing techniques. If proofs are required, then this is a necessary condition for solving certain formulas faster. However it makes proof checking more challenging. The recently proposed PR proof system [17] (best paper at CADE'17) is such a generalization of the RAT proof system, actually an instance of the most general way of defining a clausal proof system based on clause redundancy.

There are short PR proofs without new variables for some hard formulas [17]. Some of them can be found automatically [18]. The PR proof system can therefore reveal new powerful inprocessing techniques. Short proofs for hard formulas in the RAT proof system likely require many new variables, making it difficult to find them automatically. The question whether PR proofs can efficiently be converted into proofs in the RAT and DRAT proof systems has been open. In this paper, we give a positive answer and present a conversion algorithm that in the worst case results in a quadratic blowup in size. Surprisingly only a single new Boolean variable is required to convert PR proofs into DRAT proofs.

At this point there exists only an unverified checker to validate PR proofs, written in C. In order to increase the trust in the correctness of PR proofs, we implemented a tool, called `PR2DRAT`, to convert PR proofs into DRAT proofs, which in turn can be validated using verified proof checkers. Thanks to various optimizations, the size increase during conversion is rather modest on available PR proofs, thereby making this a useful certification approach in practice.

Contributions

- We show that the RAT and DRAT proof systems are as strong as the recently introduced PR proof system by giving an efficient simulation algorithm of PR proofs by RAT and DRAT proofs.
- We implemented a proof conversion tool including various optimizations, which allow us to obtain linear size DRAT proofs from PR proofs for the well-known pigeon-hole formulas. These new DRAT proofs are significantly smaller than the most compact known DRAT proof for these formulas.
- We validated short PR proofs of hard formulas by converting them into DRAT proofs and certified these using a formally verified proof checker.

Structure

After preliminaries in Sect. 2 we elaborate on clausal proof systems in Sect. 3 also taking the idea of deletion steps into account. Then Sect. 4 describes and analyzes our simulation algorithm. In Sect. 5 we present how to optimize our new algorithm for special cases followed by alternative simulation algorithms in Sect. 6. Experiments are presented in Sect. 7 before we conclude with Sect. 8.

2 Preliminaries

Below we present the most important background concepts related to this paper.

Propositional Logic. Propositional formulas in *conjunctive normal form* (CNF) are the focus of this paper. A *literal* is either a variable x (a *positive literal*) or the negation $\overline{x}$ of a variable x (a *negative literal*). The *complementary literal* $\bar{l}$ of a literal l is defined as $\bar{l} = \overline{x}$ if $l = x$ and $\bar{l} = x$ if $l = \overline{x}$. A *clause* C is a disjunction of literals. A *formula* F is a conjunction of clauses. For a literal, clause, or formula ϕ, $var(\phi)$ denotes the variables in ϕ. We treat $var(\phi)$ as a variable if ϕ is a literal, and as a set of variables otherwise.

Satisfiability. An *assignment* is a (partial) function from a set of variables to the truth values 1 (*true*) and 0 (*false*). An assignment is *total* w.r.t. a formula if it assigns a truth value to all variables occurring in the formula. We extend a given α to an assignment over literals, clauses and formulas in the natural way. Let ϕ be either a literal, clause or formula ϕ. Then ϕ is *satisfied* if $\alpha(\phi) = 1$ and *falsified* if $\alpha(\phi) = 0$. Otherwise ϕ is *unassigned*. In particular, we have $\overline{x}$ is satisfied if x is falsified by α and vice versa. A clause is satisfied by α if it contains a literal that is satisfied by α and falsified if all its literals are falsified. Finally a formula is satisfied by α if all its clauses are satisfied by α. We often denote assignments by sequences of literals they satisfy. For instance, $x\,\overline{y}$ denotes the assignment that assigns 1 to x and 0 to y. For an assignment α, $var(\alpha)$ denotes the variables assigned by α. Further, α_l denotes the assignment obtained from α by flipping the truth value of literal l assuming it is assigned. A formula is *satisfiable* if there exists an assignment that satisfies it and *unsatisfiable* otherwise.

Formula Simplification. We denote the empty clause by $\bot$ and by $\top$ the valid and always satisfied clause. A clause is a *tautology* if it contains a literal l and its negation $\bar{l}$. Given assignment α and clause C, we define $C|_\alpha = \top$ if α satisfies C; otherwise, $C|_\alpha$ denotes the result of removing from C all the literals falsified by α. For a formula F, we define $F|_\alpha = \{C|_\alpha \mid C \in F \text{ and } C|_\alpha \neq \top\}$. We say that an assignment α *touches* a clause C if $var(\alpha) \cap var(C) \neq \emptyset$. A *unit clause* is a clause with only one literal. The result of applying the *unit clause rule* to a formula F is the formula $F|_l$ where (l) is a unit clause in F. The iterated application of the unit clause rule to a formula, until no unit clauses are left, is called *unit propagation*. If unit propagation yields the empty clause $\bot$, we say that it derived a *conflict*. Given two clauses $(l \vee C)$ and $(\bar{l} \vee D)$ their *resolvent* is $C \vee D$. If further $D \subseteq C$, *self-subsuming literal elimination* (SSLE) allows removing l from $(l \vee C)$. Notice that C is the resolvent of $(l \vee C)$ and $(\bar{l} \vee D)$. So an SSLE step can be seen as two operations, learning the resolvent C followed by the removal of $(l \vee C)$, which is subsumed by C. The reverse of SSLE is *self-subsuming literal addition* (SSLA), which can add a literal l to a clause C in the presence of a clause $(\bar{l} \vee D)$ with $D \subseteq C$. The notion of SSLE first appeared in [10] and is a special case of *asymmetric literal elimination* (ALE), which in turn is the inverse of *asymmetric literal addition* (ALA) [16].

Clause C is *blocked* on literal $l \in C$ w.r.t. a formula F, if all resolvents of C and $D \in F$ with $\bar{l} \in D$ are tautologies. If a clause $C \in F$ is blocked w.r.t. F, C can be removed from F while preserving satisfiability. If a clause $C \notin F$ is blocked w.r.t. F, then C can be added to F while preserving satisfiability.

Formula Relations. Two formulas are *logically equivalent* if they are satisfied by the same assignments. Two formulas are *satisfiability equivalent* if they are either both satisfiable or both unsatisfiable. Given two formulas F and F', we denote by $F \vDash F'$ that F implies F', i.e., all assignments satisfying F also satisfy F'. Furthermore, by $F \vdash_1 F'$ we denote that for every clause $(l_1 \vee \cdots \vee l_n) \in F'$, unit propagation on $F \wedge (\bar{l}_1) \wedge \cdots \wedge (\bar{l}_n)$ derives a conflict. If $F \vdash_1 F'$, we say that F implies F' through unit propagation. For example, $(x) \wedge (y) \vdash_1 (x \vee z) \wedge (y)$, since unit propagation of the unit clauses $(\overline{x})$ and $(\overline{z})$ derives a conflict with (x), and unit propagation of $(\overline{y})$ derives a conflict with (y).

3 Clausal Proof Systems

In this section, we introduce a formal notion of clause redundancy and demonstrate how it provides the basis for clausal proof systems. We start by introducing clause redundancy [22]:

Definition 1. *A clause C is* redundant *w.r.t. a formula F if F and $F \cup \{C\}$ are satisfiability equivalent.*

For instance, the clause $C = (x \vee y)$ is redundant w.r.t. $F = (\overline{x} \vee \overline{y})$ since F and $F \cup \{C\}$ are satisfiability equivalent (although they are not logically equivalent). Since this notion of redundancy allows us to add redundant clauses to a formula without affecting its satisfiability, it gives rise to clausal proof systems.

Definition 2. *For $n \in \mathbb{N}$ a* derivation *of a formula F_n from a formula F_0 is a sequence of n triples $(d_1, C_1, \omega_1), \ldots, (d_n, C_n, \omega_n)$, where each clause C_i for $1 \leq i \leq n$ is redundant w.r.t. $F_{i-1} \setminus \{C_i\}$ with $F_i = F_{i-1} \cup \{C_i\}$ if $d_i = 0$ and $F_i = F_{i-1} \setminus \{C_i\}$ if $d_i = 1$. The assignment ω_i acts as (arbitrary)* witness *of the redundancy of C_i w.r.t. F_{i-1} and we call the number n of* steps *also the* length *of the derivation. A derivation is a* refutation *of F_0 if $d_n = 0$ and $C_n = \bot$. A derivation is a* proof of satisfaction *of F_0 if F_n equals the empty formula.*

If there exists such a derivation of a formula F' from a formula F, then F and F' are satisfiability equivalent. Further a refutation of a formula F, as defined above, obviously certifies the unsatisfiability of F since any F' containing the empty clause is unsatisfiable. Note that at this point these ω_i are still place-holders used in refinements, i.e., in the RAT and PR proof systems defined below, where these ω_i are witnesses for the redundancy of C_i w.r.t. F_{i-1}. In these specialized proof systems this redundancy can be *checked efficiently*, i.e., in polynomial time w.r.t. the size of C_i, F_{i-1} and ω_i.

3.1 The RAT proof system

The RAT proof system allows the addition of a redundant clause, which is is a so-called *resolution asymmetric tautology* [21] (RAT, defined below). It can be efficiently checked whether a clause is a RAT.The following definition of RAT is equivalent to the original one in [21] based on resolvents using results from [17].

Definition 3. *Let F be a formula, C a clause, and α the smallest assignment that falsifies C. Then, C is a* resolution asymmetric tautology (RAT) *with respect to F if there exists a literal $l \in C$ such that $F|_\alpha \vdash_1 F|_{\alpha_l}$. We say that C is a* RAT *on l w.r.t. F. The empty clause $\bot$ is a* RAT *w.r.t. F iff $F \vdash_1 \bot$.*

Informally, $F|_\alpha \vdash_1 F|_{\alpha_l}$ means that $F|_{\alpha_l}$ is at least as satisfiable compared to $F|_\alpha$. We know that α_l satisfies C as $l \in C$, thus $F|_{\alpha_l} = (F \wedge C)|_{\alpha_l}$. Hence, if F has a satisfying assignment β that falsifies C, which necessarily is an extension of α, then it also satisfies $(F \wedge C)|_{\alpha_l}$, and thus there exists a satisfying assignment of F that satisfies C, obtained from β by flipping the assigned value of l.

Example 1. Let $F = (x \vee y) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee z)$ and $C = (x \vee \overline{z})$. Then, $\alpha = \overline{x}\, z$ is the smallest assignment that falsifies C. Observe that C is a RAT clause on literal x w.r.t. F. First, $\alpha_x = x\, z$. Now, consider $F|_\alpha = (y)$ and $F|_{\alpha_x} = (y)$. Clearly, unit propagation on $F|_\alpha \wedge (\overline{y})$ derives a conflict, thus $F|_\alpha \vdash_1 F|_{\alpha_x}$. □

In a RAT derivation $(d_1, C_1, \omega_1), \ldots, (d_n, C_n, \omega_n)$ all d_i's are zero (additions). Let α_i denote the smallest assignment that falsifies C_i and let $l_i \in C_i$ be a literal on which C_i is a RAT on l_i w.r.t F_{i-1}. Each witness ω_i in a RAT derivation equals $(\alpha_i)_{l_i}$, which is obtained from α_i by flipping the value of l_i.

3.2 The PR proof system

As discussed, addition of PR clauses (short for *propagation-redundant clauses*) to a formula can lead to short proofs for hard formulas without the introduction of new variables. Although PR as well as RAT clauses are not necessarily implied by the formula, their addition preserves satisfiability [17]. The intuitive reason for this is that the addition of a PR clause prunes the search space of possible assignments in such a way that there still remain assignments under which the formula is as satisfiable as under the pruned assignments.

Definition 4. *Let F be a formula, C a non-empty clause, and α the smallest assignment that falsifies C. Then, C is* propagation redundant (PR) *with respect to F if there exists an assignment ω which satisfies C, such that $F|_\alpha \vdash_1 F|_\omega$.*

The clause C can be seen as a constraint that "prunes" from the search space all assignments that extend α. Note again, that in our setting assignments are in general partial functions. Since $F|_\alpha$ implies $F|_\omega$, every assignment that satisfies $F|_\alpha$ also satisfies $F|_\omega$, meaning that F is at least as satisfiable under ω as it is under α. Moreover, since ω satisfies C, it must disagree with α on at least one variable. We refer to ω as the *witness*, since it witnesses the propagation-redundancy of the clause. Consider the following example from [17].

Example 2. Let $F = (x \vee y) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee z)$, $C = (x)$, and let $\omega = x\, z$ be an assignment. Then, $\alpha = \overline{x}$ is the smallest assignment that falsifies C. Now, consider $F|_\alpha = (y)$ and $F|_\omega = (y)$. Clearly, unit propagation on $F|_\alpha \wedge (\overline{y})$ derives a conflict. Thus, $F|_\alpha \vdash_1 F|_\omega$ and C is propagation redundant w.r.t. F. Notice that C is not RAT w.r.t F as $(y) = F|_\alpha \nvdash_1 F|_{\alpha_x} = (y)(z)$. □

Most known types of redundant clauses are PR clauses [17], including *blocked clauses* [23], *set-blocked clauses* [22], *resolution asymmetric tautologies*, etc.

3.3 The power of deletion

The clausal proof system DRAT [29] is the de-facto standard for proofs of unsatisfiability (refutations) in practice. It extends RAT by allowing the deletion of clauses. The main purpose of clause deletion is to reduce computation cost to validate proofs of unsatisfiability. Note, that SAT solvers not only learn clauses, but also aggressively delete clauses to speed up reasoning. Integrating deletion information in proofs is crucial to speed up proof checking.

In principle, while deleted clause information has to be taken into account to update the formula after a deletion step, one does not need to check the validity of clause deletion steps in order to refute a propositional formula. Simply removing deleted clauses during proof checking trivially preserves unsatisfiability.

Proofs of satisfiability only exist in proof systems that allow and enforce valid deletion steps, because they are required to reduce a formula to the empty formula. In case of propositional formulas, the notion of proofs of satisfiability is probably not useful as a satisfying assignment can be used to certify satisfiability. However, for richer logics, such as quantified Boolean formulas, the proof of satisfiability can be exponentially smaller compared to alternatives [19,20].

4 Conversion algorithm

This section presents our main algorithm, which describes how to convert a PR derivation $(0, C_1, \omega_1), \ldots, (0, C_n, \omega_n)$ of a formula F_n from a formula F_0 into a DRAT derivation $(d_1, D_1, \omega_1'), \ldots, (d_m, D_m, \omega_m')$ of $G_m = F_n$ from $G_0 = F_0$. Each PR proof step adds a clause to the formula. Let G_0 be a copy of F_0 and $F_i := F_{i-1} \wedge C_i$ for $1 \leq i \leq n$. Each proof step in a DRAT proof either deletes or adds a clause depending on whether d_i is 1 or 0 (respectively). For $1 \leq j \leq m$ we either have $G_i := G_{i-1} \setminus \{D_i\}$ if d_i is 1 or $G_i := G_{i-1} \wedge D_i$ if d_i is 0.

Each single PR derivation step $(0, C_i, \omega_i)$ is also a PR derivation of F_i from F_{i-1} and our conversion algorithm simply translates each such PR derivation step separately into a DRAT derivation of F_i from F_{i-1}. The conversion of the whole PR derivation is then obtained as concatenation of these individual DRAT derivations, which gives a DRAT derivation of F_n from F_0. We will first offer an informal top-down description of converting a single PR derivation step into a sequence of DRAT steps.

4.1 Top-Down

Consider a formula F and a clause C which has PR w.r.t. F with witness ω, i.e., a single PR derivation step. The central question addressed in this paper is how to construct a DRAT derivation of $F \wedge C$ from F. The constructed DRAT derivation $(d_1, C_1, \omega_1), \ldots, (d_q, C_q, \omega_q), (d_{q+1}, C_{q+1}, \omega_{q+1}), \ldots, (d_p, C_p, \omega_p)$ of $F \wedge C$ from F consists of three parts. It also requires to introduce a (new) Boolean variable x that does not occur in F.

1. Construct a DRAT derivation $(d_1, C_1, \omega_1), \ldots, (d_q, C_q, \omega_q)$ of F' from F where
   a. the clause $(x \vee C)$ is a RAT on x w.r.t. F' and
   b. there exists a DRAT derivation from $F' \wedge (x \vee C)$ to $F \wedge C$.
2. In step $q+1$, clause $C_{q+1} = (x \vee C)$ is added to F'.
3. The steps after step $q+1$ transform $F' \wedge (x \vee C)$ into $F \wedge C$.

Notice that $(x \vee C)$ is blocked w.r.t. F and could therefore be added to F as a first step. However, it is very hard to eliminate literal x from $F \wedge (x \vee C)$. Instead, we transform F into F', before the addition and reverse the transformation afterwards. Below we describe the details of our simulation algorithm in five phases of which phase (I) and (II) correspond to the transformation (part 1.) and phase (IV) and (V) corresponds to the reverse transformation (part 3.).

4.2 Five Phases

We will show a transformation of how F_{i+1} is derived from F_i using PR step $(0, C_{i+1}, \omega_{i+1})$ into a sequence of p DRAT proof steps from G_j to G_{j+p} such that $G_j = F_i$ and $G_{j+p} = F_{i+1}$. In the description below, F refers to F_i, C refers to C_{i+1}, and ω refers to ω_{i+1}. Further let x be a new Boolean variable, i.e., x does not occur in F. We can assume that $var(C) \subseteq var(F)$. Otherwise there exists a literal $l \in C$ and $var(l) \notin var(F)$. Thus C is blocked on l w.r.t. F and can be added to F using a single RAT step.

(I) *Add shortened copies of clauses that are reduced, but not satisfied by ω.*
The first phase of the conversion algorithm extends F by adding the clauses $(\overline{x} \vee D)$ with $D \in F$ such that $D|_\omega \subset D$. The literal x does not occur in F. All clauses $(\overline{x} \vee D)$ are blocked on $\overline{x}$ w.r.t. F as no resolution on x is possible. We denote with $G^{(1)}$ the formula F after these clause additions.

(II) *Weaken the clauses that are reduced and satisfied by ω.*
A clause $E \in F$ is called *involved* if it is both reduced by ω as well as satisfied by ω. The second phase weakens all involved clauses by replacing E with $(x \vee E)$ as follows. First, we add the implication $x \Rightarrow \omega$, or in clauses $(\overline{x} \vee l)$ with $l \in \omega$. These clauses are blocked because $G^{(1)}$ does not contain clauses with literal x. Second, we weaken the involved clauses using self-subsuming literal addition (SSLA), since they all contain at least one $l \in \omega$. Third, we remove the implication $x \Rightarrow \omega$. When this implication was added, the clauses $(\overline{x} \vee l)$ with $l \in \omega$ were blocked on x. Now we can remove them, because they have RAT on l, which can be seen as follows. Consider a clause

containing $\bar{l}$. If it is a weakened clauses $(x \vee E)$ of E where $E \in F$ is satisfied by ω, then x occurs in opposite phase and the resolvent is a tautology (same condition as for blocked clauses). Otherwise the resolvent on l of $(\overline{x} \vee l)$ with the clause containing $\bar{l}$ is subsumed by a clause $(\overline{x} \vee D)$ with $D \in F|_\omega \setminus F$ added in first step above. The resulting formula, where all involved clauses in $G^{(\mathrm{I})}$ are weakened, is denoted by $G^{(\mathrm{II})}$.

(III) *Add the weakened* PR *clause.*
Add the clause $(x \vee C)$ to $G^{(\mathrm{II})}$, resulting in $G^{(\mathrm{III})}$. The key observation related to this phase is that $(x \vee C)$ has RAT on x w.r.t. $G^{(\mathrm{II})}$: The only clauses in $G^{(\mathrm{II})}$ that contain literal $\overline{x}$ are the ones that were added in the first phase. We need to show that $G^{(\mathrm{II})}$ implies every clause $(x \vee C \vee D)$ with $D \in F|_\omega \setminus F$ by unit propagation. Let α be the smallest assignment that falsifies C. Since C has PR w.r.t. F using witness ω, we know that $F|_\alpha \vdash_1 D$ with $D \in F|_\omega \setminus F$. This is equivalent to $F \vdash_1 (C \vee D)$ with $D \in F|_\omega \setminus F$. Furthermore $G^{(\mathrm{II})}|_{\overline{x}} \supseteq F$. Hence, $G^{(\mathrm{II})}|_{\overline{x}} \vdash_1 (C \vee D)$ or equivalently, $G^{(\mathrm{II})} \vdash_1 (x \vee C \vee D)$.

(IV) *Strengthen all weakened clauses.*
The fourth phase removes all occurrences of the literal x from clauses in $G^{(\mathrm{III})}$, thereby reversing the second phase *and* strengthening $(x \vee C)$ to C. This phase consists of three parts. First, we reintroduce the implication $x \Rightarrow \omega$, or in clauses $(\overline{x} \vee l)$ with $l \in \omega$. These clauses have RAT on l w.r.t. $G^{(\mathrm{III})}$ by the same reasoning used to remove them in the second phase above and in case $(\overline{x} \vee l)$ can be resolved on l with the only clause $(x \vee C)$ added in the third phase, thus $\bar{l} \in C$, the resolvent is a tautology (contains x and $\overline{x}$). Afterwards, we strengthen all clauses $(x \vee E) \in G^{(\mathrm{III})}$ to E as follows. Note that this also strengthens clause $(x \vee C)$ to C. Observe that all clauses $(x \vee E) \in G^{(\mathrm{III})}$ including $(x \vee C)$ are satisfied by ω and therefore there exists a clause $(\overline{x} \vee l)$ with $l \in E$. Self-subsuming literal elimination (SSLE) can now eliminate all literals x. Finally, the implication $x \Rightarrow \omega$ is no longer required. The clauses $(\overline{x} \vee l)$ with $l \in \omega$ added twice already can be removed again since literal $\overline{x}$ has become pure due to the strengthening of all clauses containing literal x. The resulting formula obtained from $G^{(\mathrm{III})}$ by removing all occurrences of literal x is denoted by $G^{(\mathrm{IV})}$.

(V) *Remove the shortened copies.*
The fifth phase reverses the first phase, and actually uses the same argument as the fourth phase. All clauses in $G^{(\mathrm{III})}$ that contained a literal x were strengthened by removing these literals in phase four. As a consequence, the literal $\overline{x}$ is (still) pure in $G^{(\mathrm{IV})}$. The only clauses that still contain literal $\overline{x}$ are exactly the clauses that have been added in the first phase. Since they are all blocked on $\overline{x}$ w.r.t. $G^{(\mathrm{IV})}$, they can be eliminated, while preserving satisfiability. After removing these clauses we obtain $G^{(\mathrm{V})}$ which equals $F \wedge C$.

4.3 Complexity

In this section we analyze the worst case complexity of converting a PR derivation $(0, C_1, \omega_1), \ldots, (0, C_n, \omega_n)$ of a formula F_n from a formula F_0 into a DRAT

derivation $(d_1, D_1, \omega'_1), \ldots, (d_1, D_m, \omega'_m)$ of $G_m = F_n$ from $G_0 = F_0$ using the presented simulation algorithm. The number of DRAT steps that are required to simulate a single PR addition step depends on the size of the formula. Let $N = |F_n|$ be the number of clauses in the last F_n and $V = |var(F_n)|$ the number of its variables. Since a PR derivation does not remove clauses, we have $|F_i| = |F_{i-1}| + 1$ and $|var(F_i)| \geq |var(F_{i-1})|$. Therefore for $i \in \{1..n\}$, $|F_i| \leq N$ and $|var(F_i)| \leq V$. In the analysis we ignore clause deletion, since the number of clause deletions is bounded by the number of added clauses.

In phase (I) of the conversion algorithm, copies of clauses that are reduced but not satisfied by ω_i are added, while phase (II) 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 $|F_i| \leq N$. Also the implication $x \Rightarrow \omega_i$ is added in phase (II), meaning at most $|var(\omega_i)| \leq |var(F_i)| \leq V$ clause addition steps. Phase (III) adds a single clause. Phase (IV) adds again the implication $x \Rightarrow \omega_i$ (at most V steps) and strengthens all weakened clauses (at most N steps). Phase (V) only deletes clauses. Thus the total number of clause additions for all phases in the conversion of a single PR step is bounded by $2V + 2N + 1$.

There are $n \leq N$ 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 derivation is at most $2NV + 2N^2 + N$. Since $V \leq N$ for any interesting PR derivation, the number of steps in the resulting DRAT derivation is in $\mathcal{O}(N^2)$.

5 Optimizations

The simulation algorithm described in the prior section was designed to result in compact DRAT derivations using a single new variable, while focussing on converting any PR derivation into a DRAT derivation. The algorithm can be further optimized to reduce the size of the resulting DRAT derivations.

5.1 Refutations

In practice, most PR derivations are refutations, i.e., they include adding the empty clause. When converting PR refutations, one can ignore the justification of any weakening steps as such steps trivially preserve unsatisfiability. The only weakening steps in the simulation algorithm are performed in phase (II). The purpose of the addition of the implication $x \Rightarrow \omega$ in phase (II) is to allow the weakening via self-subsuming literal addition (SSLA). This justification is no longer required for PR refutations. Without the addition of $x \Rightarrow \omega$, one can also discard its removal. So both the first and third part of phase (II) can be omitted.

5.2 Witness minimization

In some situations, only a subset of the involved clauses needs to be weakened (phase (II)) and later strengthened (phase (IV)). Weakening of involved clauses

is required to make sure that the clauses $(\overline{x} \vee l)$ with $l \in \omega$ are RAT on l w.r.t. $G^{(\text{III})}$ in phase (IV) of the simulation algorithm. However, some of the clauses $(\overline{x} \vee l)$ may be unit implied by others (and do not require to be a RAT on l). This situation occurs when a subset of the witness implies the full witness via unit propagation. We minimize the witness by searching for the smallest witness $\omega' \subseteq \omega$ such that ω' implies ω via unit propagation. Only clauses reduced by ω' and satisfied by ω need to be weakened in phase (II) and strengthened in (IV).

5.3 Avoiding copying

In some quite specific case, one can avoid copying the clauses that are reduced, but not satisfied by the witness altogether. In other words skip phase (I) and (V) of the simulation algorithm. This case, however, occurred frequently in our PR proofs. Let α denote the smallest assignment that falsifies the PR clause C to be added. Furthermore, let ω be the witness and ω' the minimized witness as discussed above. The condition for avoiding clause copying consists of two parts. First, there is no literal $l \in \alpha$ such that $\bar{l} \in \omega'$. Recall that there always exists a literal $l \in \alpha$ such that $\bar{l} \in \omega$. So witness minimization is necessary. Second, for each literal $l \in \omega'$, the unit clause (l) should be a RAT on l w.r.t. the current formula without the involved clauses under α. Although both conditions are very restrictive, they apply often in the PR proofs used in the evaluation.

Basically, this optimization removes phases (I) and (V), and modifies (II), (III), and (IV). The modified phases are named phase (i), (ii), and (iii), resp.

(i) *Weaken the clauses that are reduced by ω' and satisfied by ω.*
Clause $E \in F$ is called *involved* if it is reduced by the reduced witness ω' and satisfied by the original ω. The first phase weakens all involved clauses E to $(x \vee E)$ as follows. First, we add the implication $x \Rightarrow \omega' \cup \alpha$, or in clauses $(\overline{x} \vee l)$ with $l \in \omega' \cup \alpha$. These clauses are blocked because G does not contain clauses with literal x. Now we can weaken the involved clauses using SSLA. Then we remove the implication part $x \Rightarrow \omega'$, but keep $x \Rightarrow \alpha$. When adding this implication, the clauses $(\overline{x} \vee l)$ with $l \in \omega'$ were blocked on x. Now we can remove them, because they have RAT on l as all clauses containing $\bar{l}$ have been either weakened (if they were satisfied by ω) or are implied by α by the second condition. The resulting formula, G in which all involved clauses are weakened and includes $x \Rightarrow \alpha$, is denoted by $G^{(\text{i})}$.

(ii) *Add the weakened* PR *clause.*
Add the clause $(x \vee C)$, which is equivalent to the implication $x \Leftarrow \alpha$, to $G^{(\text{i})}$, resulting in $G^{(\text{ii})}$. The only clauses containing literal $\overline{x}$ are the ones that originate from $x \Rightarrow \alpha$. As a consequence, $(x \vee C)$ is blocked on x w.r.t. $G^{(\text{i})}$.

(iii) *Strengthen all weakened clauses.*
The third phase removes all occurrences of the literal x from clauses in $G^{(\text{ii})}$, thereby reversing the second phase *and* strengthening $(x \vee C)$ to C. This phase consists of four parts. First, we reintroduce the implication part $x \Rightarrow \omega'$, or in clauses $(\overline{x} \vee l)$ with $l \in \omega'$. Again, these clauses have RAT on l w.r.t. $G^{(\text{ii})}$. Second, we remove the implication part $x \Rightarrow \alpha$, i.e. the clauses

$(\overline{x} \vee l)$ with $l \in \alpha$. Afterwards, we strengthen $(x \vee C)$ to C and all clauses $(x \vee E) \in G^{(\text{ii})}$ to E. Observe that all clauses $(x \vee E) \in G^{(\text{ii})}$ including $(x \vee C)$ are satisfied by ω and therefore there exists a clause $(\overline{x} \vee l)$ with $l \in E$. SSLE can therefore remove all literals x. Finally, the implication $x \Rightarrow \omega'$ is no longer required. The clauses $(\overline{x} \vee l)$ with $l \in \omega'$ can be eliminated because literal $\overline{x}$ has become pure due to the strengthening of all clauses containing literal x. The resulting formula, i.e., $G^{(\text{ii})}$ after removing all occurrences of literal x, is denoted by $G^{(\text{iii})}$ and equals $G \wedge C$.

In case the PR derivation is a refutation, we can further optimize this case, by changing phase (i) as follows: Instead of adding the implication $x \Rightarrow \omega' \cup \alpha$, the implication $x \Rightarrow \alpha$ is added. Without the addition of the implication part $x \Rightarrow \omega'$, we can also discard removing that part at the end of phase (i).

6 Alternative Simulation Algorithms

Even though the conversion from PR derivations to DRAT derivations is arguably the most useful one in practice, one can also consider the following alternatives.

6.1 Limiting the number of RAT steps

Most steps in the simulation algorithm are "basic" steps, i.e., self-subsuming literal addition or elimination and blocked clause addition or elimination. There are only few "full" RAT addition steps: The removal of the implication in phase (II), the addition of the weakened PR clause in phase (III) and the addition of the implication in phase (IV). It is interesting to explore the option to reduce the number of these "full" RAT addition steps. Eliminating "full" RAT addition steps brings us close to a simulation algorithm with only basic steps.

It is easy to eliminate all but one "full" RAT addition step. In order to eliminate the RAT steps in phase (II), one can weaken the clauses (i.e., add a literal x using SSLA) that are reduced but not satisfied by the witness using the shortened copies of clauses that are reduced, but not satisfied by ω. After the weakening, we can remove the implication $x \Rightarrow \omega$ using blocked clause elimination (instead of RAT), because now all clauses that are touched by ω have a literal x. Therefore all clauses $(\overline{x} \vee l)$ with $l \in \omega$ are blocked on l. The weakening also allows adding the implication $x \Rightarrow \omega$ in phase (IV) using blocked clause addition steps (instead of RAT). The strengthening of the newly weakened clause can be performed in phase (IV) using SSLE (after adding the implication). It is not obvious how to replace the only remaining RAT addition in phase (III) using basic steps.

6.2 Converting DPR proofs into DRAT proofs

So far we only considered converting a PR clause addition as a sequence of DRAT steps and ignored deletion of PR clauses from a formula. In most cases, clause deletion steps in a proof facilitate more efficient checking of a proof of

unsatisfiability and can therefore be deleted without any checking. However, there are situations in which one wants to check the validity of clause deletion steps. In particular for proofs of satisfiability, i.e., a sequence of proof steps that show that a given formula is equivalent to the empty formula and thus satisfiable.

The DPR proof system is a clausal proof system that allows the addition and deletion of PR clauses. Conversion of a PR clause addition step into DRAT proof steps is equivalent to the conversion of such a step in the PR proof system. The conversion of a PR clause deletion step is slightly different. Given a formula F and a clause $C \in F$, which is a PR clause w.r.t. F with witness ω. The first phase of the conversion is exactly the same as phase (I) of the PR clause addition conversion. The second phase of the conversion is slightly different compared to phase (II) of the PR clause addition conversion: Instead of weakening all clauses reduced and satisfied by ω, we weaken all clauses satisfied by ω. Notice that this includes weakening C to $(x \vee C)$. The third phase consists of deleting $(x \vee C)$ from the current formula. Recall that phase (III) of the PR clause addition conversion added $(x \vee C)$. The final phase corresponds to phases (IV) and (V).

6.3 Converting PR refutations into RAT refutations

The presented simulation algorithm converts PR derivations into DRAT derivations. We selected the DRAT proof system as target, because it is the most widely-supported proof system by top-tier SAT solvers and it allows step-wise simulation using deletion steps. The question arises whether deletion steps are required when converting a PR refutation. In short, the answer is *no* when allowing the introduction of arbitrary many new Boolean variables. Converting a deletion step can be realized as follows. Let C be the clause that is deleted from a formula F. For each $x \in var(C)$, add to F the equivalence $x' \Leftrightarrow x$ with x' being a new variable. Afterwards, copy all clauses in F —apart from C— that contain at least one literal l with $var(l) \in var(C)$ using the new x' variables instead of the old x variables. Finally replace all occurrences of old literals x and $\overline{x}$ in the remaining proof by literals x' and $\overline{x}'$, respectively.

In order to limit the number of copy operations, one can group (consecutive) deletion steps and use the same variables x' for the group. The simulation algorithm can be partitioned into two groups of (consecutive) clause addition steps that are followed each by groups of consecutive clause deletion steps: The first group of addition steps consists of phase (I) and the first half of phase (II), i.e., adding the implication $x \Rightarrow \omega$ and the weakened involved clauses. The first group of deletion steps consists of the remaining part of phase (II), i.e., deletion of the involved clauses and deletion of the implication $x \Rightarrow \omega$. The second group of consecutive addition steps consists of phase (III) and the first half of phase (IV), i.e, adding the implication $x \Rightarrow \omega$ and adding back the involved clauses. The second group of consecutive deletion steps consists of the remaining part of phase (IV), i.e., removal of the weakened involved clauses and the implication $x \Rightarrow \omega$, and phase (V). By grouping the deletion steps, one can convert PR refutations into RAT refutations with at most a quadratic blowup, so the same worst case complexity as converting PR derivations into DRAT derivations.

7 Evaluation

We implemented a tool, called `PR2DRAT`, to convert PR proofs into DRAT proofs[3] and evaluated the tool on short PR proofs for hard formulas from three families:

(1) pigeon-hole, (2) two-pigeons-per-hole [2], and (3) Tseitin formulas [27,4].

Every resolution proof of a formula in these families is exponential in the size of the formula [11,28]. As a consequence, any CDCL solver without dedicated special reasoning techniques, such as cardinality or XOR reasoning, is unable to solve these benchmarks in reasonable time. In contrast, our PR proofs are smaller than the formulas, so linear in size. The PR proofs of the pigeon-hole formulas and two-pigeons-per-hole formulas have been constructed manually in earlier work [17]. The proofs of the Tseitin formulas have been manually constructed by expressing Gaussian elimination in the PR system. Applying Gaussian elimination —after syntactically extracting XOR constraints from the CNF formulas— is enough to solve these formulas. We will first evaluate the size of the conversion. Afterwards we certify for the first time the short PR proofs by converting them into DRAT proofs which are checked by a formally verified checker.

7.1 Proof Simulation and Optimization

We will compare three kinds of DRAT proofs for the benchmarks used in the experiments: the most compact existing ones [14,15], the proofs obtained from using our plain conversion algorithm, and the proofs obtained from our optimized algorithm. The most compact existing ones originate from expressing symmetry breaking as DRAT proof steps. Table 1 shows the comparison. All proofs have been trimmed using the `DRAT-trim` tool [12] once. Applying `DRAT-trim` multiple rounds (using the output proof as input proof for the next round) allows further reduction of the proof size, but typically these extra reductions are small.

For pigeon-hole formulas over n pigeons, the most compact existing proofs have $\mathcal{O}(n^4)$ proof steps. This is also the case for the DRAT proofs obtained through our basic conversion algorithm as well as for the extended resolution proofs by Cook [5]. However, DRAT proofs obtained with our optimized algorithm have only $\mathcal{O}(n^3)$ proof steps. Notice that the size of pigeon-hole formulas as well as the size of PR proofs are both in $\mathcal{O}(n^3)$. In other words, our optimized conversion algorithm cannot only produce DRAT proofs, but for pigeon-hole formulas it generates the first DRAT proofs of linear size.

The results for the two-pigeons-per-hole formulas are similar, but more pronounced: There exist only DRAT proofs of the formulas up to 12 holes and 25 pigeons (tph12) [15]. Our plain simulation algorithm can produce DRAT proofs of the formulas up to 20 holes and 41 pigeons (tph20). Moreover, our optimized simulation algorithm is able to produce proofs that are linear in size of the formulas, although not linear in the size of the PR proofs.

[3] The tool, checkers, formulas, and proofs discussed in this section are available at `http://www.cs.utexas.edu/~marijn/pr2drat/`.

Table 1. Comparison of the size of trimmed, generated DRAT proofs for hard formulas. The size of proofs is measured in the number of clause addition steps (#add). We denote with "—" that no DRAT proof is available. Bold is used for the smallest DRAT proofs.

	input		PR proofs	DRAT proofs (#add)		
formula	#var	#cls	#add	existing [14,15]	plain	optimized
hole20	420	4,221	2,870	49,410	94,901	**26,547**
hole30	930	13,981	9,455	234,195	422,101	**89,827**
hole40	1,640	32,841	22,140	715,030	1,241,126	**213,107**
hole50	2,550	63,801	42,925	1,708,915	2,893,476	**416,387**
tph8	136	5,457	1,156	253,958	86,216	**25,204**
tph12	300	27,625	3,950	1,966,472	612,108	**127,296**
tph16	528	87,329	9,416	—	2,490,672	**401,004**
tph20	820	213,241	18,450	—	7,440,692	**976,376**
Urquhart-s5-b1	106	714	620	—	30,235	**28,189**
Urquhart-s5-b2	107	742	606	—	34,535	**32,574**
Urquhart-s5-b3	121	1,116	692	—	44,117	**41,230**
Urquhart-s5-b4	114	888	636	—	40,598	**37,978**

We are unaware of any DRAT proofs of hard Tseitin formulas, e.g., from the Urquhart-s5-b* family [4], nor of any tool able to produce such DRAT proofs. However, we succeeded to manually produce short PR proofs without new variables for these formulas and convert them into DRAT proofs. The resulting DRAT proofs, with and without optimizations, are relatively large compared to the PR proofs. The blowup is close to the quadratic worse case. We observed that `DRAT-trim` was able to remove many (around 70%) of clause additions, which suggests that there could be an optimization to generate shorter DRAT proofs.

7.2 Verified PR Proof Checking

Our proof simulation approach can be used to validate PR proofs with formally verified tools and thereby increasing the confidence in their correctness. The tool chain works as follows: Given a formula F and an alleged PR proof P_{PR} of F, our tool `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` [13] to certify that P_{CLRAT} is a valid proof of unsatisfiability of F. Notice that the tools `PR2DRAT` and `DRAT-trim` are unverified and thus may turn an invalid proof into a valid proof or vice versa.

Figure 1 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 less than a second. The runtimes of `DRAT-trim` and `ACL2check` are one to two orders of magnitude higher than for `PR2DRAT`. Thus our tool adds little overhead to the tool chain. The sizes of the DRAT and CLRAT proofs are comparable. However, these proofs are different: `DRAT-trim` (A) removes redundant clause additions; (B) includes hints to

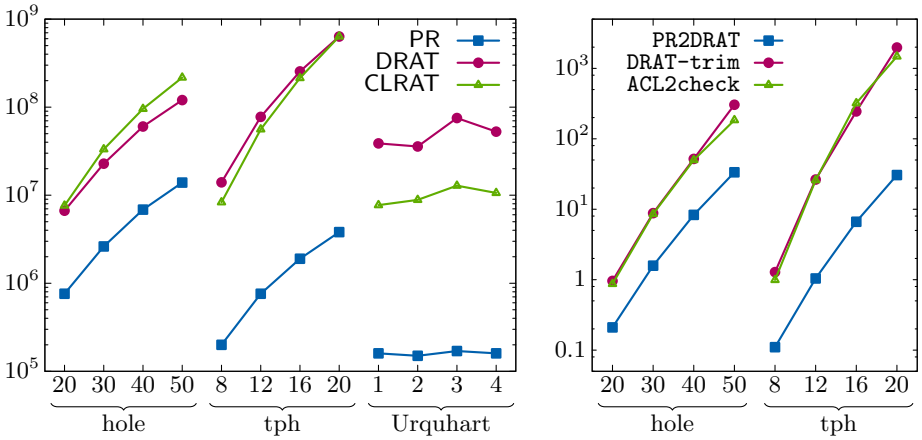

Fig. 1. Certification of PR proofs using `PR2DRAT`, `DRAT-trim`, and the formally verified checker `ACL2check`. Left the sizes of proofs in the PR, DRAT, and CLRAT formats are shown in bytes and right the proof conversion and checking times are in seconds. No times are shown for the Urquhart instances as all times were less than a second.

speedup verified checking; (C) compresses proofs. The effect of (A) depends on proof quality; (B) increases the size of proofs of small hard problems by roughly a factor of four; (C) reduces size to 30% of the uncompressed proofs. The difference between the DRAT and CLRAT proofs therefore indicate how much redundancy was removed: for pigeon-hole proofs hardly anything, for two-pigeons-per-hole proofs a modest amount, and for Tseitin proofs a lot. Notice that runtimes of the verified checker `ACL2check` are comparable to the C-based checker `DRAT-trim`.

8 Conclusions and Future Work

We showed how to convert PR proofs into DRAT proofs using only a single new variable with an at most quadratic blowup in proof size. This result suggests that it might also be possible to construct DRAT proofs without new variables using one variable elimination step and reusing the eliminated variable. The optimizations implemented in our conversion tool `PR2DRAT` made it possible to produce DRAT proofs for hard problems that are significantly smaller compared to existing DRAT proofs of those problems. The main open question is whether PR proofs can be converted into RAT proofs (i.e., not allowing the deletion steps) with a small number of new variables. Without deletion steps, it seems that copying the formula using new variables is required.

Our new tool chain for certifying SAT solving results using PR proofs consists of four steps: proof production (solving), conversion from PR to DRAT, conversion from DRAT to CLRAT, and validation of the CLRAT proof using a formally verified checker. In order to fasten adaptation of the approach, we are exploring elimination of the second step, by integrating the conversion algorithm in either SAT solvers or in DRAT proof checkers.

References

1. Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Solving difficult sat instances in the presence of symmetry. In: Design Automation Conference, 2002. Proceedings. 39th. pp. 731–736 (2002)
2. Biere, A.: Two pigeons per hole problem. In: Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions. p. 103 (2013)
3. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of bdds. In: DAC. pp. 317–320 (1999)
4. Chatalic, P., Simon, L.: Multi-resolution on compressed sets of clauses. In: 12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2000), 13-15 November 2000, Vancouver, BC, Canada. pp. 2–10 (2000)
5. Cook, S.A.: A short proof of the pigeon hole principle using extended resolution. SIGACT News 8(4), 28–32 (Oct 1976)
6. Crawford, J.M., L., G.M., Luks, E.M., Roy, A.: Symmetry-breaking predicates for search problems. In: Knowledge Representation and Reasoning – KR 1996. pp. 148–159. Morgan Kaufmann (1996)
7. Cruz-Filipe, L., Heule, M.J.H., Hunt Jr., W.A., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In: CADE. Lecture Notes in Computer Science, vol. 10395, pp. 220–236. Springer (2017)
8. Cruz-Filipe, L., P., J.M., Schneider-Kamp, P.: Efficient certified resolution proof checking. In: TACAS 2017. Lecture Notes in Computer Science, vol. 10205, pp. 118–135 (2017)
9. D'Silva, V., Kroening, D., Weissenbacher, G.: A survey of automated techniques for formal software verification. IEEE Trans. on CAD of Integrated Circuits and Systems 27(7), 1165–1178 (2008)
10. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3569, pp. 61–75. Springer (2005)
11. Haken, A.: The intractability of resolution. Theor. Comput. Sci. 39, 297–308 (1985)
12. Heule, M.J.H.: The DRAT format and DRAT-trim checker. CoRR, abs/1610.06229, 2016
13. Heule, M.J.H., Hunt Jr., W.A., Kaufmann, M., Wetzler, N.D.: Efficient, verified checking of propositional proofs. In: Ayala-Rincón, M., Muñoz, C.A. (eds.) Interactive Theorem Proving: 8th International Conference, ITP 2017, Brasília, Brazil, September 26–29, 2017, Proceedings. pp. 269–284. Springer (2017)
14. Heule, M.J.H., Hunt Jr., W.A., Wetzler, N.D.: Verifying refutations with extended resolution. In: Automated Deduction – CADE-24. pp. 345–359. Springer Berlin Heidelberg (2013)
15. Heule, M.J.H., Hunt Jr., W.A., Wetzler, N.D.: Expressing Symmetry Breaking in DRAT Proofs, pp. 591–606. Springer International Publishing (2015)
16. Heule, M.J.H., Järvisalo, M., Lonsing, F., Seidl, M., Biere, A.: Clause elimination for SAT and QSAT. J. Artif. Intell. Res. 53, 127–168 (2015)
17. Heule, M.J.H., Kiesl, B., Biere, A.: Short proofs without new variables. In: de Moura, L. (ed.) Proc. of the 26th Int. Conference on Automated Deduction (CADE-26). LNCS, vol. 10395, pp. 130–147. Springer, Heidelberg (2017)
18. Heule, M.J.H., Kiesl, B., Seidl, M., Biere, A.: PRuning through satisfication. In: HVC. LNCS, Springer (2017), to be Published

19. Heule, M.J.H., Seidl, M., Biere, A.: A unified proof system for QBF preprocessing. In: IJCAR. Lecture Notes in Computer Science, vol. 8562, pp. 91–106. Springer (2014)
20. Heule, M.J.H., Seidl, M., Biere, A.: Solution validation and extraction for QBF preprocessing. J. Autom. Reasoning 58(1), 97–125 (2017)
21. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR. LNCS, vol. 7364, pp. 355–370. Springer (2012)
22. Kiesl, B., Seidl, M., Tompits, H., Biere, A.: Super-blocked clauses. In: Automated Reasoning: 8th International Joint Conference, IJCAR 2016. pp. 45–61. Springer (2016)
23. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics 96-97, 149–176 (1999)
24. Lammich, P.: Efficient verified (un)sat certificate checking. In: Automated Deduction – CADE 26. pp. 237–254. Springer International Publishing, Cham (2017)
25. Luo, M., Li, C., Xiao, F., Manyà, F., Lü, Z.: An effective learnt clause minimization approach for CDCL SAT solvers. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17. pp. 703–711 (2017)
26. Marques-Silva, J.P., Lynce, I., Malik, S.: Conflict-Driven Clause Learning SAT Solvers, FAIA, vol. 185, chap. 4, pp. 131–153. IOS Press (February 2009)
27. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of Reasoning 2, pp. 466–483. Springer (1983)
28. Urquhart, A.: Hard examples for resolution. Journal of the ACM 34(1), 209–219 (1987)
29. Wetzler, N.D., Heule, M.J.H., Hunt Jr., W.A.: DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In: Theory and Applications of Satisfiability Testing – SAT 2014. pp. 422–429. Springer International Publishing, Cham (2014)
30. Zhang, L., Malik, S.: Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In: DATE. pp. 10880–10885 (2003)