EPiC Series in Computing
Volume 41, 2016, Pages 24–38
GCAI 2016. 2nd Global Conference on Artificial Intelligence

Unsatisfiability Proofs for Parallel SAT Solver Portfolios with Clause Sharing and Inprocessing

Tobias Philipp

International Center for Computational Logic, Technische Universität Dresden

Abstract

State-of-the-art SAT solvers are highly tuned systematic-search procedures augmented with formula simplification techniques. They emit unsatisfiability proofs in the DRAT format to guarantee correctness of their answers. However, the DRAT format is inadequate to model some parallel SAT solvers such as the award-winning system *Plingeling*. In *Plingeling*, each solver in the portfolio applies clause addition and elimination techniques. Clause sharing is restricted to clauses that do not contain *melted* literals. In this paper, we develop a transition system that models the computation of such parallel portfolio solvers. The transition system allows us to formally reason about portfolio solvers, and we show that the formalism is sound and complete. Based on the formalism, we derive a new proof format, called parallel DRAT, which can be used to certify UNSAT answers.

1 Introduction

The satisfiability problem is one of the most prominent problems in computer science and artificial intelligence. It has many applications such as in hardware and software verification [7], planning [27, 42], and bioinformatics [30]. Today, SAT solvers are highly tuned [3, 15, 23, 38] systematic search procedure augmented with *clause learning* [37], *clause removal* [2, 3, 12], and *formula simplification techniques* [26]. The *portfolio approach* [16] is a simple but successful approach for the parallelization of SAT solvers. It exploits different search strategies by running different SAT solvers on the same input formula. *Clause sharing* is an important improvement in parallel portfolios that allows to share clauses among the solver incarnations in the portfolio. In fact, this improvement allows to solve a formula faster than every sequential SAT solver on its own. The solver *Plingeling* [6] is based on the portfolio approach and received the gold medal in the application track of the SAT competition 2014 and 2013, and obtained the second prize in the SAT race 2015. *Plingeling*'s portfolio consists of several instances of *Lingeling*, that apply clause elimination techniques such as blocked clause elimination, and clause addition techniques that change the semantics of their working formula. Whenever one instance applies such a technique w.r.t. some literal, it marks the literal as *melted*. Intuitively, this means that the meaning of the formula with respect to this literal is changed. Then, clause sharing between two solver incarnations is restricted to clauses that do not contain *melted literals*. If we do not restrict clause sharing in this setting, one can share clauses that are not logical consequences of the initial formula, which may turn a satisfiable formula into an unsatisfiable one. Consequently, a SAT solver may incorrectly report that a formula is unsatisfiable.

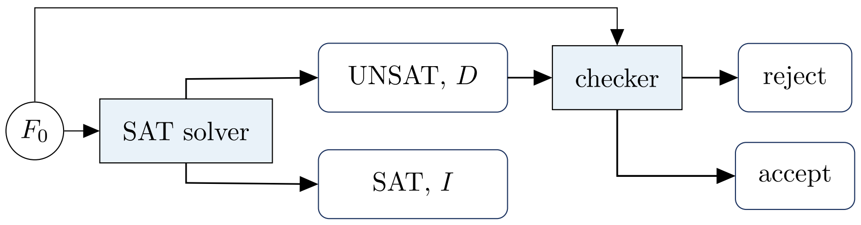

Figure 1: The certifying computations approach, where F_0 is the input formula, I is an interpretation, and D is a certificate of unsatisfiability.

On the other hand, the source code of SAT solvers became highly complex which results in wrong answers given by intensively-tested SAT solvers: three solvers that participated in the SAT competition 2009, and five solvers that participated in the SAT competition 2007 were buggy, and returned incorrect results [9]. The critical case is when formulas are incorrectly reported to be unsatisfiable, since the answer is hard to verify. Subtle bugs in different components of SAT solvers were reported in [26, 34]. Moreover, some bugs only occur in some configurations of SAT solvers, as demonstrated recently with *SpyBug* [33].

One approach for improving the reliability of SAT solvers is to mechanically verify them, as done in [35]. However, mechanically-verified SAT solvers are currently significantly slower than state-of-the-art solvers, written in C and C++, such as *CryptoMiniSAT*, *Lingeling*, *Glucose*, and *Riss*. Therefore, several proof formats were proposed to certify UNSAT from SAT solvers. The idea is that SAT solvers produce a certificate that can easily be checked by an independent program (see Fig. 1). In the case the checker accepts the certificate together with the input formula, we know that the input formula is unsatisfiable assuming that the small checker works correctly.

Today, the DRAT format (Deletion Resolution Asymmetric Tautology) is the de facto standard, and emitting proofs in the DRAT format is a requirement in the main track of the SAT competition 2016. A proof in the DRAT format is a sequence of clauses, which have been learned or deleted during the run in a sequential SAT solver, and includes all known formula simplification techniques. Recently, the DRAT format received media attention because SAT solvers solved the Pythagorean Triples Problem and its 200 Terabytes proof was expressed in the format [22]. However, parallel SAT solvers such as *Plingeling* cannot express their proofs in the DRAT format, since the proofs constructed from the sequential incarnations cannot be merged into a single DRAT proof.

We propose to formalize the computation of SAT solvers which allows us to formally reason about these systems. In particular, one can construct certificates from it. For parallel SAT solvers based on the instance decomposition approach, formal models and proof formats exist such as [21, 40], and also formalisms that model some portfolios with arbitrary clause sharing but limited formula simplifications [20, 34]. However, they do not include the setting by *Plingeling*.

Our contributions

1. We develop a formal model $\mathcal{P}_1$ that describes parallel SAT solvers where each solver can apply arbitrary formula simplification techniques, and clause sharing is restricted to clauses that do not contain melted literals. We show soundness and completeness of $\mathcal{P}_1$. It models the computations performed by *Plingeling*.

2. We derive a new proof format from $\mathcal{P}_1$, called PDRAT (Parallel DRAT), which can be used to verify UNSAT answers from such parallel SAT solvers. It is a conservative generalization of the DRAT format in the sense that it is equivalent to DRAT for portfolio solvers consisting of a single solver. We show that a formula is unsatisfiable if and only if a PDRAT refutation of F exists.

3. We present an efficient method to check such parallel DRAT proofs and show correctness of the method.

The paper is structured as follows: in Section 2 we present propositional logic and redundancy criteria, Sect. 3 presents the formal model, showing soundness and completeness of the model. Afterwards, the proof format is derived in Sect. 4. Section 5 concludes the paper.

2 Background

2.1 Propositional Logic

We consider an infinite set of propositional variables $\mathcal{V}$. A literal L is either a propositional variable A or its negation $\neg A$. The complement of a literal L is denoted by $\overline{L}$, i.e. $\overline{A} = \neg A$ and $\overline{\neg A} = A$. Clauses are sets of literals, and formulas are finite sets of clauses, where

The semantics of formulas is built on interpretations. An *interpretation* I is a mapping from the set $\mathcal{V}$ of all Boolean variables to the set $\{\top, \bot\}$ of truth values, represented by the set of variables mapped to $\top$ under I. The interpretation I *satisfies* the variable A, in symbols, $I \models A$, if and only if $A \in I$. It *satisfies* the negated variable $\neg A$, in symbols, $I \models \neg A$, if and only if $A \notin I$. It *satisfies* the clause C, in symbols $I \models C$, if and only if there is a literal $L \in C$ such that $I \models L$. For a formula F, the interpretation I *satisfies* the formula F, in symbols $I \models F$, if and only if for every clause $C \in F$ we find that the interpretation I satisfies the clause C. A *model* I of a formula F is an interpretation I that satisfies the formula F. If such a model I of F exists, the formula F is *satisfiable*. Otherwise, the formula F is *unsatisfiable*.

Two formulas F and F' are *equisatisfiable*, in symbols $F \equiv_{\mathsf{sat}} F'$, if and only if either both are satisfiable or both are unsatisfiable. The formula F *entails* the formula F' if and only if every model of the formula F is a model of the formula F'. Two formulas F and F' are *semantically equivalent*, in symbols $F \equiv F'$, if and only if the formula F entails the formula F' and vice versa.

Let C and D be two clauses and L be a literal such that $L \in C$ and $\overline{L} \in D$. Then, the *resolvent of C and D upon L* is $(C \setminus \{L\}) \cup (D \setminus \{\overline{L}\})$. A *tautological* clause is a clause containing A and $\neg A$ for some variable A, and a clause C subsumes D, if $C \subseteq D$.

2.2 Redundancy Properties

The Resolution Asymmetric Tautology (RAT) property is based on the notion of *asymmetric literal addition (ALA)* [26]:

$$\mathsf{ALA}_F(C) = C \cup \{\overline{L} \mid \{L_1, \ldots, L_n, L\} \in F \text{ and } \{L_1, \ldots, L_n\} \subseteq C\}$$

We consider the recursive application of asymmetric literal addition:

$$\begin{array}{lcl} \mathsf{ALA}_F(C) \uparrow 0 & = & C \\ \mathsf{ALA}_F(C) \uparrow n+1 & = & \mathsf{ALA}_F(\mathsf{ALA}_F(C) \uparrow n) \end{array}$$

A clause C is an *asymmetric tautology (AT)* w.r.t. the formula F, if there is $n \in \mathbb{N}$ such that the clause $\mathsf{ALA}_F(C) \uparrow n$ is a tautology. Notice that there is a small technical differences to the original version of ALA and asymmetric tautology: We phrase ALA as a deterministic mathematical function, and asymmetric tautologies now precisely correspond to clauses that can be inferred by reverse unit propagation.

Example 1. *Consider the following formula $F = \{\{p, q\}, \{p, \neg q, r\}, \{\neg q, \neg r\}\}$. Then the following holds*

$$\begin{array}{lcllcl}
\mathsf{ALA}_F(\{p\}) \uparrow 0 & = & \{p\} & \mathsf{ALA}_F(\{q\}) \uparrow 0 & = & \{q\} \\
\mathsf{ALA}_F(\{p\}) \uparrow 1 & = & \{p, \neg q\} & \mathsf{ALA}_F(\{q\}) \uparrow 1 & = & \{q, \neg p\} \\
\mathsf{ALA}_F(\{p\}) \uparrow 2 & = & \{p, \neg q, \neg r, r\} & \mathsf{ALA}_F(\{q\}) \uparrow 2 & = & \mathsf{ALA}_F(\{q\}) \uparrow 1 \\
\mathsf{ALA}_F(\{p\}) \uparrow 3 & = & \{p, \neg q, \neg r, r, q\} & & & \\
\mathsf{ALA}_F(\{p\}) \uparrow 4 & = & \mathsf{ALA}_F(\{p\}) \uparrow 3 & & &
\end{array}$$

Therefore, $\{p\}$ is an AT w.r.t. F because $\{r, \neg r\} \subseteq \mathsf{ALA}_F(\{p\}) \uparrow 4$, whereas $\{q\}$ is not an AT w.r.t. F because $\mathsf{ALA}_F(\{q\}) \uparrow n$ is not a tautology for all $n \geq 0$.

Note that ALA is monotone in F and C. Replacing a clause C in F by $\mathsf{ALA}_F(C)$ preserves semantical equivalence. Learned conflict clauses are asymmetric tautologies [4] as well as *subsumed clauses*, *tautologies*, and *resolvents* [26].

Järvisalo et al. introduced the following redundancy criteria based on an asymmetric tautologies in [26]: The clause C is a *resolution asymmetric tautology (RAT) upon L w.r.t. F*, if *(1)* the clause C is an asymmetric tautology w.r.t. the formula F, or *(2)* there is a literal $L \in C$ such that the resolvent of C and D upon L is an asymmetric tautology w.r.t. the formula F for every $D \in F$ with $\overline{L} \in D$.

Example 2. *The clauses $\{p\}, \{\neg q\}, \{\neg r\}, \{q, r\}$ are the minimal resolution asymmetric tautologies in the formula F from Example 1. Then* 1. *$\{p\}$ is a RAT upon p w.r.t. F because there is no clause $D \in F$ with $\neg p \in D$.* 2. *$\{\neg q\}$ is a RAT upon $\neg q$, because there is only one resolvent $\{p\}$ which is an AT w.r.t. F.* 3. *$\{\neg r\}$ is a RAT upon $\neg r$, because there is only one resolvent $\{p, \neg q\}$, which is an AT because it is subsumed by the clause $\{p\}$ which is an AT w.r.t. F.* 4. *$\{q, r\}$ is a RAT upon q because the resolvent $\{p, r\}$ is an AT and the resolvent $\{r, \neg r\}$ is a trivial AT.*

Several formula simplification techniques were proposed, including the following ones, that can be characterized in terms of RAT: *bounded variable elimination and addition* [11,32,41,45], *blocked clause elimination* [25], *blocked clause addition* [26,28], *equivalent literal elimination* [13], *probing* [31], *extended resolution and reencoding* [32,46], *symmetry breaking* [10,17], and *BDD-based reasoning* [44].

3 A Formal Model for Parallel SAT Portfolios

For sequential SAT solvers, several formalizations exist, which model different parts of SAT solvers [1, 8, 24, 26, 35, 39]. Here, we model a sequential SAT solver as a pair (M, F), where M is a finite set of *melted* literals, and F is the *working formula.* The solver can modify the working formula, by adding learned clauses to it. Note that this operation preserves semantical equivalence. In the case that the solver removes a clause C from F, we require that the resulting formula is equivalent w.r.t. satisfiability to the original formula. Furthermore, the solver can add a clause C to the formula F, whenever there is a literal L such that C is a RAT w.r.t. L in F. In this case, the solver adds L to the set of melted literals.

A portfolio consists of a finite number of sequential SAT solvers. Initially, they all work on copies of the input formula, and process them independently. Therefore, a state of a portfolio system of m solvers is simply a snapshot of all sequential SAT solvers, i.e. $((M_1, F_1), \dots, (M_m, F_m))$. Clause sharing is a technique that allows to share clauses between the sequential SAT solvers. In *Plingeling*, solver j can send a clause C to a distinct solver i, if $C \in F_j$ and C does not contain melted literals of solver i and j. In this case, we can add C to F_k. In the case that one of the solvers in the portfolio found a model, or the empty clause appear in one of the working formulas, the complete procedure terminate with this answer.

We model the portfolio approach as a transition system: A *state transition system* is a tuple $(\Delta, \rightsquigarrow)$ where Δ is the set of *states* and $\rightsquigarrow \subseteq \Delta \times \Delta$ is the *state transition relation.* Given a state transition system $(\Delta, \rightsquigarrow)$, we define $\overset{0}{\rightsquigarrow} = \{(x, x) \mid x \in \Delta\}$, $\overset{n}{\rightsquigarrow} = \{(x, z) \mid (x, y) \in \overset{n-1}{\rightsquigarrow}$ and $(y, z) \in \rightsquigarrow\}$ for all $n \in \mathbb{N}_{>0}$ and $\overset{*}{\rightsquigarrow} = \cup_{i\in\mathbb{N}} \overset{i}{\rightsquigarrow}$. We write $x \rightsquigarrow y$ instead of $(x, y) \in \rightsquigarrow$. Formally, a *portfolio system with multiplicity* n is a state transition system whose set of states is $\{\mathsf{SAT}, \mathsf{UNSAT}\}$ together with all n-tuples of the form $((M_1, F_1), \dots, (M_n, F_n))$. The *initial state for the input formula* F_0, denoted by $\mathsf{init}(F_0)$, is the n-tuple $((\emptyset, F_0), \dots, (\emptyset, F_0))$. The transition relation of $\mathcal{P}_1$ is composed of the relations presented in Fig. 2, except the UDEL-rule, which will be used later.

$$\rightsquigarrow_{\mathcal{P}_1} := \{\mathsf{SAT}, \mathsf{UNSAT}, \mathsf{AT}, \mathsf{RAT}, \mathsf{DEL}, \mathsf{SHARE}\}.$$

Figure 2 contain on the right column corresponding proof elements, which will be explained in Sect. 4. We have two termination rules: The SAT-rule terminates the computation in the final state SAT if $Solver_i$ found that its working formula F_i is satisfiable. Likewise, the UNSAT-rule terminates the computation in the final state UNSAT if the formula F_i contains the empty clause. The AT-rule models clause learning, i.e. if a clause C is an AT w.r.t. F_i, then we add the clause C to the formula F_i. The AT-rule subsumes in particular the first UIP learning employed by many state-of-the-art SAT solvers. The RAT-rule models clause addition techniques that preserve satisfiability, i.e. blocked clause addition and extended resolution: if a clause C is a RAT upon L w.r.t. F_i, then we add the clause C to the formula F_i, and add L to M_i. The DEL-rule models clause deletion and allows to remove a clause form a formula, if this operation preserves satisfiability of the formula. The SHARE-rule models restricted clause sharing: We add the clause C from the formula F_j to the formula F_i, if $C \cap M_i = \emptyset$ and $C \cap M_j = \emptyset$, i.e. no literal is C occurs in one of the literal sets M_i and M_j.

Clause sharing among the solver incarnations has to be restricted, as otherwise sharing a clause can make a formula unsatisfiable:

3.1 The Portfolio Model is Sound and Complete

We consider the following properties of the introduced portfolio system: *Termination:* the execution starting from an initial state eventually reaches a final state, i.e.,there is no infinite chain $S_1 \rightsquigarrow S_2 \rightsquigarrow \dots$ *Soundness:* the transition system computes correct answers, i.e. for all formulas F_0 and $m > 0$ we have that $\mathsf{init}(F_0, m) \overset{*}{\rightsquigarrow} \mathsf{SAT}$ implies that the formula F_0 is satisfiable and $\mathsf{init}(F_0, m) \overset{*}{\rightsquigarrow} \mathsf{UNSAT}$ implies that the formula F_0 is unsatisfiable. Intuitively, soundness means that every answer in the system is correct. *Completeness:* the transition system is able to infer the correct answer, i.e. for all formulas F_0 and $m > 0$ we have that $\mathsf{init}(F_0, m) \overset{*}{\rightsquigarrow} \mathsf{UNSAT}$, if the formula is unsatisfiable, and $\mathsf{init}(F_0, m) \overset{*}{\rightsquigarrow} \mathsf{SAT}$, if the formula is satisfiable.

It is straightforward from the definition of AT-rule that $\mathcal{P}_1$ is not terminating, since we can add and remove a clause infinitely many often.

SAT	$((M_1, F_1), \ldots, (M_n, F_n)) \rightsquigarrow \mathsf{SAT}$ if F_i is satisfiable for some $i \in \{1, \ldots, n\}$.	
UNSAT	$((M_1, F_1), \ldots, (M_n, F_n)) \rightsquigarrow \mathsf{UNSAT}$ if $\emptyset \in F_i$ for some $i \in \{1, \ldots, n\}$.	
AT	$((M_1, F_1), \ldots, (M_n, F_n)) \rightsquigarrow$ $((M_1, F_1), \ldots, (M_{i-1}, F_{i-1}), (M_i, F_i \cup \{C\}), (M_{i+1}, F_{i+1}), \ldots, (M_n, F_n))$ if C is an AT w.r.t. F_i.	(a, i, C)
RAT	$((M_1, F_1), \ldots, (M_n, F_n)) \rightsquigarrow$ $((M_1, F_1), \ldots, (M_{i-1}, F_{i-1}), (M_i \cup \{L\}, F_i \cup \{C\}), (M_{i+1}, F_{i+1}), \ldots, (M_n, F_n))$ if C is a RAT upon L w.r.t. F_i.	(a, i, C)
DEL	$((M_1, F_1), \ldots, (M_n, F_n)) \rightsquigarrow$ $((M_1, F_1), \ldots, (M_{i-1}, F_{i-1}), (M_i, F_i \setminus \{C\}), (M_{i+1}, F_{i+1}), \ldots, (M_n, F_n))$ if $F_i \equiv_{\mathsf{sat}} F_i \setminus \{C\}$.	(d, i, C)
SHARE	$((M_1, F_1), \ldots, (M_n, F_n)) \rightsquigarrow$ $((M_1, F_1), \ldots, (M_{i-1}, F_{i-1}), (M_i, F_i \cup \{C\}), (M_{i+1}, F_{i+1}), \ldots, (M_n, F_n))$ if $C \in F_j$, $C \cap M_i = \emptyset$, $C \cap M_j = \emptyset$ and $i \neq j$.	(a, i, C)
UDEL	$((M_1, F_1), \ldots, (M_n, F_n)) \rightsquigarrow$ $((M_1, F_1), \ldots, (M_{i-1}, F_{i-1}), (M_i, F_i \setminus \{C\}), (M_{i+1}, F_{i+1}), \ldots, (M_n, F_n))$	

Figure 2: The rules of the the portfolio model. These definitions apply to all formulas F_i, F_i', and M_i, and clauses C, where $i \in \{1, \ldots, n\}$, and the corresponding proof steps are explained in Sect. 4.

Proposition 1. *$\mathcal{P}_1$ does not terminate.*

However, $\mathcal{P}_1$ is sound and complete. We express the invariants of the portfolio model in terms of literal forgetting. Intuitively, the *forgetting in a formula F about a literal set S* expresses the same over $(\mathcal{V} \cup \{\neg A \mid A \in \mathcal{V}\}) \setminus S$ of the formula F, but nothing about the literals in S (see [29, 47]).

Definition 1. *Let S be a set of literals. Then $I \models \mathrm{forget}(S, F)$, iff there is a model J of F such that $J \models L$ implies $I \models L$ for all literals L with $L \notin S$ [47].*

Example 3. *Consider the formula F from Example 1. Then* $\mathrm{forget}(\{p\}, F) \equiv \{\{\neg r, \neg q\}\}$*. We know that $\{p\}$ is a RAT w.r.t. F. Moreover* $\mathrm{forget}(\{\neg p\}, F \cup \{\{p\}\}) \equiv \{\{\neg r, \neg q\}\}$.

We can give a semantically equivalent propositional formula for the expression $\mathrm{forget}(\{p\}, F)$, i.e. the forgetting operator can be expanded into propositional logic:

$$\mathrm{forget}(\{p\}, F) \equiv (F \wedge \neg p) \vee (F[p/\top]),$$

where $F[p/\top]$ is the formula obtained by F by replacing each occurrence of the variable p with the truth symbol $\top$. The forgetting operator can also be expressed in terms of quantified Boolean formulas, e.g.

$$\mathrm{forget}(\{\neg p\}, F) \equiv \exists q. F[p/q] \wedge (q \rightarrow p),$$

where q is a fresh variable. Likewise, forgetting of a variable p and its negation $\neg p$ is the same as variable elimination if we consider formulas in conjunctive normal form: we replace clauses in a formula F containing p or $\neg p$ with the all possible resolvents over p.

Proposition 2. *Let F, F' be formulas, C be a clause and S be a literal set. Then, the following holds:*

1. *F is satisfiable iff $\text{forget}\,(S, F)$ is satisfiable.*
2. *$F \models \text{forget}\,(S, F)$.*
3. *If $F \equiv F'$, then $\text{forget}\,(S, F) \equiv \text{forget}\,(S, F')$.*
4. *$\text{forget}\,(M_2, \text{forget}\,(M_1, F)) \equiv \text{forget}\,(M_1 \cup M_2, F)$.*
5. *If $F \models C$, then $\text{forget}\,(S, F) \equiv \text{forget}\,(S, F \cup \{C\})$.*
6. *Let $C \cap S = \emptyset$. Then $F \models C$ iff $\text{forget}\,(S, F) \models C$.*
7. *If C is a RAT upon L w.r.t. F, then $\text{forget}\,(\{L\}, F) \equiv \text{forget}\,(\{L\}, F \cup \{C\})$.*
8. *If $\text{forget}\,(M_1, F_1) \models \text{forget}\,(M_2, F_2)$, then $\text{forget}\,(M_1 \cup \{L\}, F_1) \models \text{forget}\,(M_2 \cup \{L\}, F_2)$*

Proof. The statements (1) – (4) were proven in [47, Prop 7]. (5) follows from the fact that forgetting is a semantic operator (see (3)). (6) and (8) are easy to see. (7) is proven in [26]. □

The following lemma expresses some invariants in $\mathcal{P}_1$, and is used to show soundness of $\mathcal{P}_1$, i.e. if $\mathcal{P}_1$ terminates with SAT, then the input formula is satisfiable, and if $\mathcal{P}_1$ terminates in UNSAT, then the input formula is unsatisfiable. The invariants state that the working formulas F_i are entailed by the input formula w.r.t. the forgetting about the melted literals M_i, and furthermore, the input formula and the working formulas are equivalent w.r.t. satisfiability.

Lemma 1. *Let F_0 be a formula, $n > 0$ and $m \in \mathbb{N}$. Assume that*

$$\mathsf{init}(F_0, n) \overset{m}{\rightsquigarrow}_{\mathcal{P}_1} ((M_1, F_1), \ldots, (M_n, F_n))$$

Then the following properties hold:

$\mathsf{inv\text{-}1}$	$\text{forget}\,(M_i, F_0) \models \text{forget}\,(M_i, F_i)$ *for every $i \in \{1, \ldots, n\}$, and*
$\mathsf{inv\text{-}2}$	$\text{forget}\,(M_i, F_0) \equiv_{\mathsf{sat}} \text{forget}\,(M_i, F_i)$ *for every $i \in \{1, \ldots, n\}$.*

Proof. We show the statement by induction on the number m of transition steps. For the base case $m = 0$, $\mathsf{inv\text{-}1}$ trivially holds since $F_i = F_0$, and $\mathsf{inv\text{-}2}$ follows by Prop. 2.2. For the induction step, assume that the claim holds for the state $((F_1, M_1), \ldots, (F_n, M_n))$ and that

$$((F_1, M_1), \ldots, (F_n, M_n)) \rightsquigarrow_{\mathsf{R}} ((F_1', M_1'), \ldots, (F_n', M_n'))$$

for some rule R. Note that $\mathsf{R} \notin \{\mathsf{SAT}, \mathsf{UNSAT}\}$. We distinguish between the applied rule R:

AT-*rule*: Then, there is an $i \in \{1, \ldots, n\}$ and C such that the following holds: i) $M_k = M_k'$ for all $k \in \{1, \ldots, n\}$, ii) $F_k = F_k'$ for all $k \in \{1, \ldots, n\} \setminus \{i\}$, iii) $F_i' = F_i \cup \{C\}$, and iv) $F_i \models C$. Since iv) we know that $F_i' \equiv F_i$. Consequently, the literal sets M_i were not modified, and the semantics of the formulas kept untouched. Since the literal forgetting operator is a semantic operator, it cannot distinguish between equivalent formulas.

RAT-*rule*: Then, there is an $i \in \{1, \ldots, n\}$, C and L such that the following holds: i) $M_k = M_k'$ for all $k \in \{1, \ldots, n\} \setminus \{i\}$, ii) $F_k = F_k'$ for all $k \in \{1, \ldots, n\} \setminus \{i\}$, iii) C is a RAT upon L w.r.t. F_i, iv) $F_i' = F_i \cup \{C\}$, and v) $M_i = M_i \cup \{L\}$. It follows by induction hypothesis and the facts i) and ii) that $\text{forget}\,(M_k, F_0) \models \text{forget}\,(M_k, F_k)$ for all $k \in \{1, \ldots, n\} \setminus \{i\}$. It remains to show that $\text{forget}\,(M_i \cup \{L\}, F_0) \models \text{forget}\,(M_i \cup \{L\}, F_i \cup \{C\})$. By induction hypothesis we know that $\text{forget}\,(M_i, F_0) \models \text{forget}\,(M_i, F_i)$. By Prop. 2.8 we know that $\text{forget}\,(M_i \cup \{L\}, F_0) \models \text{forget}\,(M_i \cup \{L\}, F_i)$. Moreover, by Prop. 2.4 we know that $\text{forget}\,(M_i \cup \{L\}, F_i) = \text{forget}\,(M_i, \text{forget}\,(\{L\}, F_i))$. Then by Prop. 2.7 and iii), we know that $\text{forget}\,(\{L\}, F_i) \equiv \text{forget}\,(\{L\}, F_i \cup \{C\})$. Consequently, we conclude $\text{forget}\,(M_i \cup \{L\}, F_i) \equiv \text{forget}\,(M_i \cup \{L\}, F_i \cup \{C\})$ by Prop. 2.3. $\mathsf{inv\text{-}2}$ can be proven as in the AT-rule.

DEL-*rule*: Then, there is an $i \in \{1, \ldots, n\}$ and C such that the following holds: i) $M_k = M'_k$ for all $k \in \{1, \ldots, n\}$, ii) $F_k = F'_k$ for all $k \in \{1, \ldots, n\} \setminus \{i\}$, iii) $F'_i = F_i \setminus \{C\}$, and iv) $F_i \equiv_{\mathsf{sat}} F_i \setminus \{C\}$. inv-1 follows by the observation that $F_i \models F_i \setminus \{C\}$. inv-2 follows straightforward by iv).

SHARE-*rule*: Then, there is an $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, n\}$ with $i \neq j$ such that the following holds: i) $C \in F_j$, ii) $F'_i = F_i \cup \{C\}$, iii) $M_k = M'_k$ for all $k \in \{1, \ldots, n\}$, iv) $F_k = F'_k$ for all $k \in \{1, \ldots, n\} \setminus \{i\}$, v) $C \cap M_i = \emptyset$, and vi) $C \cap M_j = \emptyset$. First, we proof inv-1: It follows by induction hypothesis and the facts iii) and iv) that $\text{forget}\,(M_k, F_0) \models \text{forget}\,(M_k, F_k)$ for all $k \in \{1, \ldots, n\} \setminus \{i\}$. It remains to show that $\text{forget}\,(M'_i, F_0) \models \text{forget}\,(M'_i, F'_i)$, i.e. $\text{forget}\,(M_i, F_0) \models \text{forget}\,(M_i, F_i \cup \{C\})$. By vi) and Prop. 2.6 we conclude that $\text{forget}\,(M_j, F_j) \models C$. Consequently we obtain $\text{forget}\,(M_j, F_0) \models C$ by induction hypothesis. Since v) it follows that $F_0 \models C$. Then $\text{forget}\,(M_i, F_0) \models C$ since v). We also know by induction hypothesis that $\text{forget}\,(M_i, F_0) \models \text{forget}\,(M_i, F_i)$. Therefore, $\text{forget}\,(M_i, F_o) \models \text{forget}\,(M_i, F_i) \cup \{C\}$. We conclude $\text{forget}\,(M_i, F_o) \models \text{forget}\,(M_i, F_i \cup \{C\})$ by Prop. 2.5. Second, we proof inv-2: In the case that $\text{forget}\,(M_i, F_0)$ is satisfied by an interpretation I, we can conclude that I is a model of $\text{forget}\,(M_i, F'_i)$ by inv-1. Otherwise, we know that $\text{forget}\,(M_i, F_i)$ is unsatisfiable by induction hypothesis. By Prop. 2.1 we know that F_i is unsatisfiable. Consequently, $F'_i = F_i \cup \{C\}$ is unsatisfiable. Therefore, $\text{forget}\,(M_i, F'_i)$ is unsatisfiable by Prop. 2.1. □

We can now show correctness of the formalism:

Theorem 1. *The portfolio model $\mathcal{P}_1$ is sound and complete.*

Proof. Suppose some formula F_i is satisfiable. Then, by Prop 2.1 we conclude that $\text{forget}\,(M_i, F_i)$ is satisfiable. Moreover, we know by inv-2 that $\text{forget}\,(M_i, F_i) \equiv_{\mathsf{sat}} \text{forget}\,(M_i, F_0)$. Therefore $\text{forget}\,(M_i, F_0)$ is satisfiable. Again, by Prop 2.1 we conclude that F_0 is satisfiable. Soundness with respect to UNSAT answers can be treated analogously. Completeness for satisfiable formulas is due to the fact that one can immediately terminate with SAT. Completeness for unsatisfiable formulas follows from the fact that resolvents are asymmetric tautologies, and since for unsatisfiable formulas, a resolution refutation exists, we can apply the AT-rule until we added the empty clause. □

3.2 Variation with Unlimited Deletion

We will now consider the system $\mathcal{P}_2$, which is a variation of $\mathcal{P}_1$, which is later used to construct a proof format: The SAT-rule is not contained in $\mathcal{P}_2$ and replace the DEL-rule with UDEL-rule that allows to remove every clause. Formally, the transition relation of our system $\mathcal{P}_2$ is of the following rules (see Figure 2): $\rightsquigarrow_{\mathcal{P}_2} := \{\mathsf{UNSAT}, \mathsf{AT}, \mathsf{RAT}, \mathsf{UDEL}, \mathsf{SHARE}\}$. $\mathcal{P}_2$ is non-terminating, and incomplete, since for satisfiable formulas, it is not possible to reach a final state. However, $\mathcal{P}_2$ is sound:

Lemma 2. *Let F_0 be a formula, $n > 0$ and $m \in \mathbb{N}$. Assume that*

$$\mathsf{init}(F_0, n) \overset{m}{\rightsquigarrow}_{\mathcal{P}_2} ((M_1, F_1), \ldots, (M_n, F_n))$$

Then the following properties hold:

inv-1 $\text{forget}\,(M_i, F_0) \models \text{forget}\,(M_i, F_i)$ *for every* $i \in \{1, \ldots, n\}$*, and*
inv-2 *If* $\text{forget}\,(M_i, F_0)$ *is satisfiable, then* $\text{forget}\,(M_i, F_i)$ *is satisfiable for every* $i \in \{1, \ldots, n\}$.

Proof. This can be shown similar to Lemma 1. □

Theorem 2. *The portfolio model $\mathcal{P}_2$ is sound.*

Proof. Suppose some formula F_i contain the empty clause, and therefore unsatisfiable. Then, by Prop 2.1 we conclude that forget (M_i, F_i) is unsatisfiable. Since by inv-2 of Lemma 2 we know that forget (M_i, F_0) must be unsatisfiable. By Prop 2.1 we conclude that F_0 is unsatisfiable. □

Note that $\mathcal{P}_2$ is complete w.r.t. unsatisfiable formulas, and can be shown analogously to the proof in Theorem 1.

4 A New Proof Format for Parallel SAT Portfolios

The DRAT format is based on the idea of *clausal proofs*, suggested by Goldberg et al. [14]. It consists of a sequence of clauses that were added to the working formula by the SAT solver. This made it easy to construct proofs from clause learning SAT solvers [36]. Beame et al. characterized learned clauses as trivial resolution derivations [4], which can be efficiently checked in terms of *reverse unit propagation*. *Clause removal* [3, 12] is also traced in form of deletion information [18]. Later, Järvisalo et al. generalized the concept of trivial resolution derivation to *resolution asymmetric tautologies (RAT)*, thus allowing proof generation for most known formula simplification techniques [26]. In particular, RAT subsumes *extended resolution* [43, 46], which allows to infer fresh variables. Heule et al. developed the *drat-trim* [19, 48] tool based on *backward checking* [18], which efficiently checks unsatisfiability proofs, as well as the mechanically verified checker written in the ACL2 theorem prover [49].

However, the DRAT format is inadequate to be used as clausal proofs for parallel portfolios, since the simultaneously addition of RAT clauses to a formula can make the resulting formula unsatisfiable:

Example 4. *Consider the input formula F in Example 1 and the RAT clauses presented in Example 2. Suppose we have given three solvers* $\mathsf{Solver}_1, \mathsf{Solver}_2$ *and* Solver_3. *Then,* Solver_1 *adds* $\{\neg q\}$, Solver_2 *adds* $\{\neg r\}$, *and* Solver_3 *adds* $\{q, r\}$. *However,* $F \cup \{\{\neg q\}, \{\neg r\}, \{q, r\}\}$ *is unsatisfiable, but F is satisfiable.*

Instead of merging the clauses into a single formula as done in [20], we propose to trace the added and deleted clauses from each solver incarnation separately: Each solver incarnation in the portfolio logs clause addition and deletion information to a central proof logging device. We omit the information whether a clause is an AT, RAT or was obtained by importing the clause from another solver.

4.1 Parallel DRAT

Labeled clauses are expressions of the form (ℓ, j, C), where $\ell \in \{\mathsf{a}, \mathsf{d}\}$, $j \in \mathbb{N}$, and C is a clause. Intuitively, $\ell = \mathsf{a}$ ($\ell = \mathsf{d}$, resp.) expresses that $Solver_j$ adds (deletes, resp.) the clause C. Figure 2 contains for each rule the corresponding labeled clause. A *run in $\mathcal{P}_2$ of multiplicity m on input F* is a sequence of states in $(S_i \mid 0 \le i \le n)$ such that that $S_0 = \mathsf{init}(F_0, m)$, and $S_i \rightsquigarrow S_{i+1}$ for all $i \in \{0, \ldots, n-1\}$. For convenience we write $(S!j)$ for the formula F_j in a state S in $\mathcal{P}_2$ of multiplicity m of the form $((M_1, F_1), \ldots, (M_m, F_m))$.

Definition 2. *A sequence of labeled clauses $(D_i \mid 1 \le i \le n)$* represents *the run in $\mathcal{P}_2$ of multiplicity m on input F $(S_i \mid 0 \le i \le n)$, if for every $i \in \{1, \ldots, n\}$ it holds that: if*

$D_i = (a, j, C)$, then $(S_i!j) = (S_{i-1}!j) \cup \{C\}$, and if $D_i = (d, j, C)$, then $(S_i!j) = (S_{i-1}!j) \setminus \{C\}$. A sequence of labeled clauses $(D_i \mid 1 \leq i \leq n)$ is a PDRAT derivation *in F, if it represents some run in $\mathcal{P}_2$ of some multiplicity on input F. A PDRAT derivation $(D_i \mid 1 \leq i \leq n)$ in F is a* PDRAT refutation of F *if it represents some run $(S_i \mid 0 \leq i \leq n)$ in $\mathcal{P}_2$ of some multiplicity m on input F such that there is $k \in \{1, \ldots, m\}$ with $\emptyset \in (S_n!k)$.*

Intuitively, a sequence of labeled clauses represents a run if the working formulas in the run are modified according to the labeled clauses. A sequence of labeled clauses is a PDRAT derivation if there exists a run that is represented by the sequence. The following example illustrates these definitions:

Example 5. *Consider the following input formula*

$$F_0 = \{\{p, q, r\}, \{p, \neg q, r\}, \{\neg p, q, r\}, \{\neg p, \neg q, r\}\}$$

The clauses $C_1 = \{q, r\}$ and $C_2 = \{\neg q, r\}$ are AT w.r.t. F_0, the clause $C_3 = \{r\}$ is no AT w.r.t. F, an AT w.r.t. $F_0 \cup \{C_1, C_2\}$ and is a RAT upon q w.r.t. F_0. Then labeled clause sequence $D = (\mathsf{a}, 1, C_1)(\mathsf{a}, 1, C_2)(\mathsf{a}, 1, C_3)(\mathsf{a}, 2, C_3)$ is a PDRAT derivation in F_0 since it represents the following two runs:

$$\begin{aligned} \mathsf{init}(F_0, 2) &\rightsquigarrow_{\mathsf{AT}} ((\emptyset, F_0 \cup \{C_1\}), (\emptyset, F_0)) &&\rightsquigarrow_{\mathsf{AT}} ((\emptyset, F_0 \cup \{C_1, C_2\}), (\emptyset, F_0)) \\ &\rightsquigarrow_{\mathsf{AT}} ((\emptyset, F_0 \cup \{C_1, C_2, C_3\}), (\emptyset, F_0)) &&\rightsquigarrow_{\mathsf{SHARE}} ((\emptyset, F_0 \cup \{C_1, C_2, C_3\}), (\emptyset, F_0 \cup \{C_3\})) \end{aligned}$$

$$\begin{aligned} \mathsf{init}(F_0, 2) &\rightsquigarrow_{\mathsf{AT}} ((\emptyset, F_0 \cup \{C_1\}), (\emptyset, F_0)) &&\rightsquigarrow_{\mathsf{AT}} ((\emptyset, F_0 \cup \{C_1, C_2\}), (\emptyset, F_0)) \\ &\rightsquigarrow_{\mathsf{AT}} ((\emptyset, F_0 \cup \{C_1, C_2, C_3\}), (\emptyset, F_0)) &&\rightsquigarrow_{\mathsf{RAT}} ((\emptyset, F_0 \cup \{C_1, C_2, C_3\}), (\{q\}, F_0 \cup \{C_3\})) \end{aligned}$$

Theorem 3. *F is unsatisfiable if and only if there is a PDRAT refutation of F.*

Proof. We show both directions:

- $\Rightarrow$ Suppose that F is unsatisfiable. As $\mathcal{P}_2$ is complete w.r.t. to unsatisfiable formulas, we conclude $\mathsf{init}(F, m) \overset{n}{\rightsquigarrow}_{\mathcal{P}_2} S_n \rightsquigarrow_{\mathcal{P}_2} \mathsf{UNSAT}$ for all $m > 0$ and some $n \in \mathbb{N}$. Therefore, there is a run $(S_i \mid 1 \leq i \leq n)$ in $\mathcal{P}_2$ of multiplicity m on input F such that for some $j \in \{1, \ldots, m\}$ it holds that $\emptyset \in (S_n!j)$. It is easy to see that the construction of the sequence of labeled clauses $(D_i \mid 1 \leq i \leq n)$ given in Fig 2 represents this run. Therefore, it is a PDRAT refutation of F.
- $\Leftarrow$ Suppose there is a PDRAT refutation $(D_i \mid 1 \leq i \leq n)$ of F. Consequently, there exists a run $(S_i \mid 1 \leq i \leq n)$ in $\mathcal{P}_2$ on input F that represents D. Therefore we know $\mathsf{init}(F, m) = S_0 \overset{n}{\rightsquigarrow}_{\mathcal{P}_2} S_n \rightsquigarrow_{\mathcal{P}_2} \mathsf{UNSAT}$ for some $m > 0$. By Theorem 2 we conclude that F is unsatisfiable. □

4.2 Forward-Checking Parallel DRAT Refutations

Algorithm 1 presents a procedure that efficiently answers the question, whether a sequence of labeled clauses is a PDRAT refutation of F. Line 1 and 2 initialize the variables F_i and M_i such that they represent the initial state for the input formula F. Afterwards, it iterates over the labeled clauses, i.e. $i \in \{1, \ldots, n\}$. It first checks whether D_i is of the form (d, j, C). In this case, it removes one occurrence of the clause C from F_j. Otherwise, it checks 1. whether the clause is an AT w.r.t. F_j, or 2. whether it can be imported from F_k, or 3. whether it is a RAT upon *pick*(i) w.r.t. F_j, where the function *pick*(i) is an arbitrary function returning a literal for each i. If one of these checks succeeds, it adds the clause C to F_j, and in the case only the third check succeeds, we add L to M_j. If all checks fail, the algorithm rejects the sequence of labeled clauses in Line 13. Finally, we check whether the empty clause appears in some of the working formulas in Line 14. If the empty clause appears, we accept the derivation, otherwise we reject it.

Algorithm 1: Forward checking algorithm

```
input : input formula $F$, and labeled clause sequence $(D_i \mid 1 \leq i \leq n)$ of multiplicity $m$
1  $F_k \leftarrow F$ for each $k \in \{1, \ldots, m\}$
2  $M_k \leftarrow \emptyset$ for each $k \in \{1, \ldots, m\}$
3  for $i \leftarrow 1$ to $n$ do
4      $(\ell, j, C) \leftarrow D_i$
5      if $\ell = \mathsf{d}$ then $F_j \leftarrow F_j \setminus \{C\}$
6      else
7          if $C$ is AT w.r.t. $F_j$ then $F_j \leftarrow F_j \cup \{C\}$
8          else if there is $k \in \{1, \ldots, m\}$ s.t. $C \in F_k$, $C \cap M_k = \emptyset$, $C \cap M_j = \emptyset$ and $k \neq j$ then
9              $F_j \leftarrow F_j \cup \{C\}$
10         else if $C$ is RAT upon pick$(i)$ w.r.t. $F_j$ then
11             $M_j \leftarrow M_j \cup \{L\}$
12             $F_j \leftarrow F_j \cup \{C\}$
13         else reject
14 if there is $k \in \{1, \ldots, m\}$ such that $\emptyset \in F_k$ then accept
15 else reject
```

Example 5 shows that a sequence of labeled clauses may represent several runs that are different in the set of melted literals. Observe that in the forward-checking procedure, we check, whether there is a run in the $\mathcal{P}_2$ that prefers AT and SHARE-rule over RAT-rule. Such a run always exists, as $\mathcal{P}_1$ and $\mathcal{P}_2$ are *monotone* in the following sense:

Lemma 3. *If* $((S_1, F_1), \ldots, (S_n, F_n)) \overset{*}{\leadsto} ((S'_1, F'_1), \ldots, (S'_n, F'_n))$ *and* $T_i \subseteq S_i$ *for every* $i \in \{1, \ldots, n\}$ *then* $((T_1, F_1), \ldots, (T_n, F_n)) \overset{*}{\leadsto} ((T'_1, F'_1), \ldots, (T'_n, F'_n))$*, and* $T'_i \subseteq S'_i$ *for every* $i \in \{1, \ldots, n\}$*.*

Proof. By induction over the length of the transition steps. □

We can now state the main theorem:

Theorem 4. *If forward checking accepts* (F, D)*, then* D *is a PDRAT refutation of* F*.*

Proof. Let $D = (D_i \mid 1 \leq i \leq n)$ be a labeled clause sequence and assume that forward checking accepts F, D. The algorithm constructs a sequence of states $(S_i \mid 0 \leq i \leq n)$. As forward checking accepts D, we know that $S_i \leadsto S_{i+1}$ for every $i \in \{1, \ldots, n\}$. Moreover, it holds that $\mathsf{init}(F, m) = ((\emptyset, F_1), \ldots, (\emptyset, F_n))$. As the algorithm additionally checks whether the empty clause appears in some F_k, we know that the labeled clause sequence represents the run $(S_i \mid 0 \leq i \leq n)$, and consequently is a PDRAT refutation of F. □

Corollary 1. *If forward checking accepts* (F, D)*, then* F *is unsatisfiable.*

Proof. Straightforward from Theorems 3 and 4. □

In implementation of DRAT checkers, labeled clauses are represented by a string of space-separated literals, with a possible prefix of “d” to denote deletion information. Moreover, one can assume without loss of generality that the melted literal is always the first literal in this sequence. Then $pick(i)$ function is defined to return the first literal in this sequence. If an implementation of the PDRAT construction in parallel portfolios guarantees this, we can show that forward checking is complete, in a similar way as we have shown soundness.

5 Conclusion

We considered the parallel portfolio approach for solving the propositional satisfiability problem. In this approach, the input formula is given to multiple SAT solvers that run in parallel. Each solver may add clauses that are AT or RAT w.r.t. their working formula, or remove clauses provided that the removal preserves satisfiability. Once a solver adds a RAT upon L w.r.t. to its working formula, L is added to the set of melted literals, which forbid exporting and importing clauses that contain L. If one of the solvers detects the empty clause, the complete procedure terminates with the answer that the input formula is unsatisfiable.

The transition system $\mathcal{P}_1$ models the computation of these parallel SAT solvers, and can be used to reason formally about them. In particular, we presented a set of invariants that can be used to show soundness of the system. The portfolio model $\mathcal{P}_1$ is sound and complete, which in particular shows that more clauses can be shared than in *Plingeling*. To the best of the authors knowledge, existing formalisms did not included this setting: The formal models in [34] imposed the restriction that at most one solver applies clause addition techniques, with the consequence that all clauses can be shared; and the formal model in [40] was based on the instance decomposition approach, and all solvers were restricted to equivalence-preserving clause addition techniques.

Unfortunately, we cannot merge the formulas appearing in the state of the formal models, as it was done [20] to guarantee correctness of portfolio solvers. Therefore, we developed the PDRAT format, which is a generalization of the DRAT proof format. This allows to emit unsatisfiability proofs from parallel portfolios where clause sharing is restricted but formula simplifications are applied with no restriction. To the best of our knowledge, this is the first approach that allows generation of unsatisfiability proofs of the best available portfolio solver. These proofs can then be independently checked by the forward checking procedure: in the case it is accepted, we know that the input formula must be unsatisfiable; otherwise, the checker rejects the certificate, and we discovered a buggy run in the parallel SAT solver. This certifying-computation approach guarantees the correctness of unsatisfiability answers in parallel SAT solver portfolios. A prototypical Haskell implementation of the forward checking procedure is available[1].

In the future, we plan to find a backward checking theme similar as in [18], which is known to significantly improve the efficiency, as well as implementing proof construction in the SAT solvers *PRiss* and *Plingeling*.

References

[1] H. Arnold. A linearized DPLL calculus with clause learning. Technical report, Universität Potsdam. Institut für Informatik, 2009.

[1]https://iccl.inf.tu-dresden.de/web/PDRAT

[2] G. Audemard, J.-M. Lagniez, B. Mazure, and L. Sais. On freezing and reactivating learnt clauses. In K. A. Sakallah and L. Simon, editors, *SAT 2011*, volume 6695 of *LNCS*, pages 188–200, Heidelberg, 2011. Springer.

[3] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In C. Boutilier, editor, *IJCAI 2009*, pages 399–404, Pasadena, 2009. Morgan Kaufmann Publishers Inc.

[4] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligene Research*, 22(1):319–351, 2004.

[5] D. L. Berre, editor. volume 27 of *EPiC Series in Computing*, 2014.

[6] A. Biere. Lingeling, Plingeling and Treengeling entering the SAT competition 2013. FMV Report Series vol. B-2013-1 of Department of Computer Science Series of Publications B, University of Helsinki, 2013.

[7] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *DAC 1999*, pages 317–320, 1999.

[8] J. C. Blanchette, M. Fleury, and C. Weidenbach. A verified SAT solver framework with learn, forget, restart, and incrementality. In N. Olivetti and A. Tiwari, editors, *IJCAR 2016*, volume 9706 of *LNCS*, pages 25–44. Springer, 2016.

[9] R. Brummayer and A. Biere. Fuzzing and delta-debugging SMT solvers. In *Workshop SMT 2010*, pages 1–5. ACM, 2009.

[10] J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *KR 1996*, pages 148–159. Morgan Kaufmann, 1996.

[11] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In F. Bacchus and T. Walsh, editors, *SAT 2005*, volume 3569 of *LNCS*, pages 61–75, Heidelberg, 2005. Springer.

[12] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, Heidelberg, 2004. Springer.

[13] A. V. Gelder. Toward leaner binary-clause reasoning in a satisfiability solver. *Annals of Mathematics and Artificial Intelligence*, 43(1):239–253, 2005.

[14] E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *DATE 2003*, pages 10886–10891, Washington, DC, USA, 2003. IEEE Computer Society.

[15] C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1–2):67–100, 2000.

[16] Y. Hamadi, S. Jabbour, and L. Sais. ManySAT: a parallel SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6(4):245–262, 2009.

[17] M. Heule, W. A. Hunt, Jr, and N. Wetzler. Expressing symmetry breaking in DRAT proofs. In A. P. Felty and A. Middeldorp, editors, *CADE 25*, volume 9195 of *LNCS*, pages 591–606. Springer, 2015.

[18] M. Heule, W. A. Hunt Jr, and N. Wetzler. Trimming while checking clausal proofs. In B. Jobstman and S. Ray, editors, *FMCAD 2013*, pages 181–188. IEEE, 2013.

[19] M. Heule, W. A. Hunt Jr, and N. Wetzler. Verifying refutations with extended resolution. In M. P. Bonacina, editor, *CADE 2013*, volume 7898 of *LNCS*, pages 345–359. Springer, 2013.

[20] M. Heule, N. Manthey, and T. Philipp. Validating unsatisfiability results of clause sharing parallel SAT solvers. In Berre [5].

[21] M. J. H. Heule and A. Biere. Compositional propositional proofs. In M. Davis, A. Fehnker, A. McIver, and A. Voronkov, editors, *LPAR 2015*, volume 9450 of *Lecture Notes in Computer Science*, pages 444–459. Springer, 2015.

[22] M. J. H. Heule, O. Kullmann, and V. W. Marek. Solving and verifying the Boolean Pythagorean Triples Problem via cube-and-conquer. *CoRR*, abs/1605.00723, 2016.

[23] S. Hölldobler, N. Manthey, and A. Saptawijaya. Improving resource-unaware SAT solvers. In C. G. Fermüller and A. Voronkov, editors, *LPAR 2010*, volume 6397 of *LNCS*, pages 519–534, Heidelberg, 2010. Springer.

[24] S. Hölldobler, N. Manthey, T. Philipp, and P. Steinke. Generic CDCL – A formalization of modern propositional satisfiability solvers. In Berre [5].

[25] M. Järvisalo, A. Biere, and M. Heule. Blocked clause elimination. In J. Esparza and R. Majumdar, editors, *TACAS 2010*, volume 6015 of *LNCS*, pages 129–144, Heidelberg, 2010. Springer.

[26] M. Järvisalo, M. J. H. Heule, and A. Biere. Inprocessing rules. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR 2012*, volume 7364 of *LNCS*, pages 355–370, Heidelberg, 2012. Springer.

[27] H. Kautz and B. Selman. Planning as satisfiability. In B. Neumann, editor, *ECAI 1992*, pages 359–363, New York, 1992. John Wiley & Sons, Inc.

[28] O. Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 96-97:149–176, 1999.

[29] J. Lang, P. Liberatore, and P. Marquis. Propositional independence – formula-variable independence and forgetting. *Journal of Artificial Intelligence Research*, 18:391–443, 2003.

[30] I. Lynce and J. Marques-Silva. Efficient haplotype inference with Boolean satisfiability. In *AAAI 2006*, pages 104–109, Menlo Park, California, 2006. AAAI Press.

[31] I. Lynce and J. P. Marques-Silva. Probing-based preprocessing techniques for propositional satisfiability. In *ICTAI 2003*, pages 105–110, Sacramento, California, USA, 2003. IEEE Computer Society.

[32] N. Manthey, M. J. H. Heule, and A. Biere. Automated reencoding of Boolean formulas. In A. Biere, A. Nahir, and T. Vos, editors, *HVC 2012*, volume 7857 of *LNCS*, pages 102–117, Heidelberg, 2013. Springer.

[33] N. Manthey and M. Lindauer. SpyBug: Automated bug detection in the configuration space of SAT solvers. In *SAT 2016*, pages 554–561, 2016.

[34] N. Manthey, T. Philipp, and C. Wernhard. Soundness of inprocessing in clause sharing SAT solvers. In M. Järvisalo and A. Van Gelder, editors, *SAT 2013*, volume 7962 of *LNCS*, pages 22–39, Heidelberg, 2013. Springer.

[35] F. Marić. Formalization and implementation of modern SAT solvers. *Journal of Automated Reasoning*, 43(1):81–119, 2009.

[36] J. P. Marques Silva and K. A. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD 1996*, pages 220–227, Washington, 1996. IEEE Computer Society.

[37] J. P. Marques Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[38] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC 2001*, pages 530–535, New York, 2001. ACM.

[39] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Abstract DPLL and abstract DPLL modulo theories. In F. Baader and A. Voronkov, editors, *LPAR 2004*, volume 3452 of *LNCS*, pages 36–50, Heidelberg, 2005. Springer.

[40] T. Philipp. An expressive model for instance decomposition based parallel SAT solvers. In C. Lutz and S. Ranise, editors, *FroCos 2015*, volume 9322 of *LNCS*, pages 101–116. Springer, 2015.

[41] J. Rintanen. Compact representation of sets of binary constraints. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *ECAI 2006*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 143–147. IOS Press, 2006.

[42] J. Rintanen. Engineering efficient planners with SAT. In L. D. Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. J. F. Lucas, editors, *ECAI 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 684–689. IOS Press, 2012.

[43] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[44] C. Sinz and A. Biere. Extended resolution proofs for conjoining BDDs. In D. Grigoriev, J. Harrison, and E. A. Hirsch, editors, *CSR 2006*, volume 3967 of *Lecture Notes in Computer Science*, pages 600–611. Springer, 2006.

[45] S. Subbarayan and D. K. Pradhan. NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In H. H. Hoos and D. G. Mitchell, editors, *SAT 2004*, volume 3542 of *LNCS*, pages 276–291, Heidelberg, 2005. Springer.

[46] G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic*, Part II:115–125, 1968.

[47] C. Wernhard. Projection and scope-determined circumscription. *Journal of Symbolic Computation*, 47(9):1089–1108, 2012.

[48] N. Wetzler, M. Heule, and W. A. Hunt, Jr. Drat-trim: Efficient checking and trimming using expressive clausal proofs. In C. Sinz and U. Egly, editors, *SAT 2014*, volume 8561 of *LNCS*, pages 422–429. Springer, 2014.

[49] N. Wetzler, M. J. Heule, and W. A. Hunt Jr. Mechanical verification of SAT refutations with extended resolution. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP 2013*, volume 7998 of *LNCS*, pages 229–244, Heidelberg, 2013. Springer.