Inprocessing Rules[⋆]

Matti Järvisalo[1], Marijn Heule[2,3], and Armin Biere[3]

[1] Department of Computer Science & HIIT, University of Helsinki, Finland
[2] Department of Software Technology, Delft University of Technology, The Netherlands
[3] Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria

Abstract. Decision procedures for Boolean satisfiability (SAT), especially modern conflict-driven clause learning (CDCL) solvers, act routinely as core solving engines in various real-world applications. Preprocessing, i.e., applying formula rewriting/simplification rules to the input formula before the actual search for satisfiability, has become an essential part of the SAT solving tool chain. Further, some of the strongest SAT solvers today add more reasoning to search by *interleaving* formula simplification and CDCL search. Such *inprocessing SAT solvers* witness the fact that implementing additional deduction rules in CDCL solvers leverages the efficiency of state-of-the-art SAT solving further. In this paper we establish formal underpinnings of inprocessing SAT solving via an abstract inprocessing framework that covers a wide range of modern SAT solving techniques.

1 Introduction

Decision procedures for Boolean satisfiability (SAT), especially modern conflict-driven clause learning (CDCL) [1,2] SAT solvers, act routinely as core solving engines in many industrial and other real-world applications today. Formula simplification techniques such as [3,4,5,6,7,8,9,10,11,12,13,14] applied before the actual satisfiability search, i.e., in preprocessing, have proven integral in enabling efficient conjunctive normal form (CNF) level Boolean satisfiability solving for real-world application domains, and have become an essential part of the SAT solving tool chain. Taking things further, some of the strongest SAT solvers today add more reasoning to search by *interleaving* formula simplification and CDCL search. Such *inprocessing SAT solvers*, including the successful state-of-the-art CDCL SAT solvers PRECOSAT [15], CRYPTOMINISAT [16], and LINGELING [17], witness the fact that implementing additional deduction rules within CDCL solvers leverages the efficiency of state-of-the-art SAT solving further.

To illustrate the usefulness of preprocessing and inprocessing in improving the performance of current state-of-the-art SAT solvers, we modified the 2011 SAT Competition version of the state-of-the-art SAT solver LINGELING that is based on the inprocessing CDCL solver paradigm. The resulting patch[4] allows to either disable all preprocessing or to just disable inprocessing during search. We have run the original version and these two versions on the benchmarks from the application track—the most important competition category from the industrial perspective—of the last two SAT competitions organized in 2009 and 2011. The results are shown in Table 1.

[⋆] The 1st author is supported by Academy of Finland (grants 132812 and 251170), 2nd and 3rd authors by Austrian Science Foundation (FWF) NFN Grant S11408-N23 (RiSE).

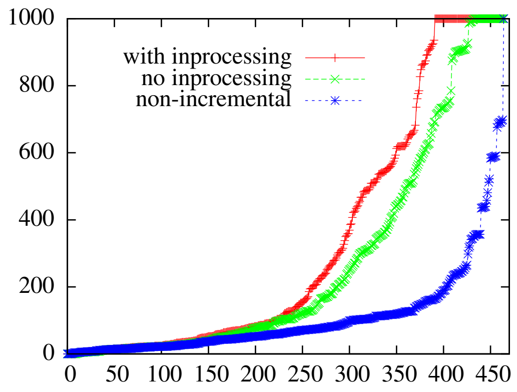
[4] `http://fmv.jku.at/lingeling/lingeling-587f-disable-pre-and-inprocessing.patch`

Table 1. Results of running the original 2011 competition version 587f of LINGELING on the application instances from 2009 and from 2011, then without inprocessing and in the last row without any pre- nor inprocessing. The experiments were obtained on a cluster with Intel Core 2 Duo Quad Q9550 2.8-GHz processors, 8-GB main memory, running Ubuntu Linux. Memory consumption was limited to 7 GB and run-time to 900 seconds. The single-threaded sequential version of LINGELING was used with one solver instance per processor.

	2009				2011			
LINGELING	solved	SAT	UNSAT	time	solved	SAT	UNSAT	time
original version 587f	196	79	117	114256	164	78	86	144932
only preprocessing	184	72	112	119161	159	77	82	145218
no pre- nor inprocessing	170	68	102	138940	156	78	78	153434

The CNF preprocessor SATELITE introduced in [7] applied *variable elimination*, one of the most effective simplification techniques in state-of-the-art SAT solvers. As already shown in [7] preprocessing can also be extremely useful within *incremental* SAT solving. This form of preprocessing, which is performed at each incremental call to the SAT solver, can be considered as an early form of inprocessing. Fig. 1 confirms this observation in the context of incremental SAT solving for bounded model checking.

However, developing and implementing sound inprocessing solvers in the presence of a wide range of different simplification techniques (including variable elimination, blocked clause elimination, distillation, equivalence reasoning) is highly non-trivial. It requires in-depth understanding on how different techniques can be combined together and interleaved with the CDCL algorithm in a satisfiability-preserving way. Moreover, the fact that many simplification techniques only preserve satisfiability but not logical equivalence poses additional challenges, since in many practical applications of SAT



	*	bounds	time
with inpr.	158	153975	304158
no inpr.	115	125436	335393
non-incr.	67	49915	369104

* = #solved + #(bound 1000 reached)

bounds = $\sum$ reached bounds

Time is summed up over all benchmarks, while unfinished runs are counted with the time limit of 900 seconds (same experimental setup as in Tab. 1).

Fig. 1. Running the bounded model checker BLIMC, which is part of the LINGELING distribution with and without inprocessing on the single property benchmarks of the Hardware Model Checking Competition 2011 up to bound 1000. With inprocessing 153975 bounds were reached, while without inprocessing only 125436. The figure shows the maximum bound reached (successfully checked) on the y-axis for each of the 465 benchmark (x-axis). Benchmarks are sorted by the maximum bound. For completeness we also include a run in non-incremental mode, which reaches only 49915 bounds. In this mode a new CNF is generated and checked for each bound with a fresh SAT solver instance separately, but with both pre- and inprocessing enabled.

solvers a solution is required for satisfiable formulas, not only the knowledge of the satisfiability of the input formula. Hence, when designing inprocessing SAT solvers for practical purposes, one also has to address the intricate task of solution reconstruction.

In this paper we propose an abstract framework that captures generally the deduction mechanisms applied within inprocessing SAT solvers. The framework consists of four generic and clean deduction rules. Importantly, the rules specify general conditions for sound inprocessing SAT solving, against which specific inprocessing techniques can be checked for correctness. The rules also capture solution reconstruction for a wide range of simplification techniques that do not preserve logical equivalence: while solution reconstruction algorithms have been proposed previously for specific inprocessing techniques [18,11], we show how a simple linear-time algorithm covers solution reconstruction for a wide range of techniques.

Our abstract framework has similarities to the abstract DPLL(T) framework [19] and its extensions [20,21], and the proof strategies approach of [22], in describing deduction via transition systems. However, in addition to inprocessing as built-in feature, our framework captures SAT solving on a more generic level than [19], not being restricted to DPLL-style search procedures, and at the same time it gives a fine-grained view of inprocessing SAT solving. We show how the rules of our framework can be instantiated to obtain both known and novel inprocessing techniques. We give examples of how the correctness of such specific techniques can be checked based on the generic rules in our framework. Furthermore, we show that our rules in the general setting are extremely powerful, even capturing Extended Resolution [23].

Arguing about correctness of combinations of different solving techniques in concrete SAT solver development is tremendously simplified by our framework. One example is the interaction of learned clauses with *variable elimination* [7]. After variable elimination is performed on the irredundant (original) clauses during inprocessing, the question is what to do with learned clauses that still contain eliminated variables. While current implementations simply forget (remove) such learned clauses, it follows easily from our framework that it is sound to keep such learned clauses and use them subsequently for propagation. It is also easy to observe e.g. that one can (selectively) turn eliminated or blocked clauses into learned clauses to preserve propagation power.

Another more intricate example from concrete SAT solver development occurs in the context of *blocked clauses* [12]. An intermediate version of LINGELING contained a simple algorithm for adding new redundant (learned) binary clauses, which are blocked, but only w.r.t. irredundant (original) clauses, thus disregarding already learned clauses. This would be convenient since focusing on irredundant clauses avoids having full occurrence lists for learned clauses. Further, marking the added clauses as redundant implies that they would not have to be considered in consecutive variable eliminations, and thus might enable to eliminate more variables without increasing the number of clauses. However, we found examples that proved this approach to be incorrect. An attempt to fix this problem was to include those added clauses in further blocked clause *removal* and *addition* attempts, and only ignore them during variable elimination. This version was kept in the code for some months without triggering any inconsistencies. However, this is incorrect, and can be easily identified via our formal framework.

After preliminaries (Sect. 2), we review redundancy properties (Sect. 3) and their extensions (Sect. 4) based on different clause elimination and addition procedures. The abstract inprocessing rules are discussed in Sect. 5, followed by an instantiation of the rules using a specific redundancy property and a related generic solution reconstruction approach (Sect. 6). Based on this instantiation of the rules, we show how the rules capture a wide range of modern SAT solving techniques and, via examples, how the rules catch incorrect variations of these techniques (Sect. 7).

2 Preliminaries

For a Boolean variable x, there are two *literals*, the positive literal x and the negative literal $\neg x$. A *clause* is a disjunction of literals and a CNF formula a conjunction of clauses. A clause can be seen as a finite set of literals and a CNF formula as a finite set of clauses. A truth assignment is a function τ that maps literals to $\{0, 1\}$ under the assumption $\tau(x) = v$ if and only if $\tau(\neg x) = 1 - v$. A clause C is satisfied by τ if $\tau(l) = 1$ for some literal $l \in C$. An assignment τ satisfies F if it satisfies every clause in F; such a τ is a *model* of F.

Two formulas are *logically equivalent* if they are satisfied by exactly the same set of assignments, and *satisfiability-equivalent* if both formulas are satisfiable or both unsatisfiable. The length of a clause C is the number of literals in C. A *unit clause* has length one, and a *binary clause* length two. The set of binary clauses in a CNF formula F is denoted by F_2. The resolution rule states that, given two clauses $C_1 = \{l, a_1, \ldots, a_n\}$ and $C_2 = \{\neg l, b_2, \ldots, b_m\}$, the clause $C_1 \otimes C_2 = \{a_1, \ldots, a_n, b_1, \ldots, b_m\}$, called the *resolvent* $C_1 \otimes_l C_2$ (or simply $C_1 \otimes C_2$ when clear from context) of C_1 and C_2, can be inferred by *resolving* on the literal l. For a CNF formula F, let F_l denote the set of clauses in F that contain the literal l. The resolution operator $\otimes_l$ can be lifted to sets of clauses by defining $F_l \otimes_l F_{\neg l} = \{C \otimes_l C' \mid C \in F_l,\, C' \in F_{\neg l}\}$.

3 Clause Elimination and Addition

Clause elimination procedures [11] are an important family of CNF simplification techniques which are to an extent orthogonal with resolution-based techniques [12]. Intuitively, clause elimination refers to removing from CNF formulas clauses that are *redundant* (with respect to some specific redundancy property) in the sense that satisfiability is preserved under removal.

Definition 1. *Given a CNF formula F, a specific clause elimination procedure $\mathcal{P}\mathrm{E}$ removes clauses that have a specific property $\mathcal{P}$ from F until fixpoint. In other words, $\mathcal{P}\mathrm{E}$ on input F modifies F by repeating the following until fixpoint: if there is a clause $C \in F$ that has $\mathcal{P}$, let $F := F \setminus \{C\}$.*

Clause Addition Procedures, the dual of clause elimination procedures, add to (instead of removing from) CNF formulas clauses that are *redundant* (with respect to some specific redundancy property) in the sense that satisfiability is preserved under adding.

Definition 2. *Given a CNF formula F, a specific clause addition procedure $\mathcal{P}\mathrm{A}$ adds clauses that have a specific property $\mathcal{P}$ to F until fixpoint. In other words, $\mathcal{P}\mathrm{A}$ on input F modifies F by repeating the following until fixpoint: if there is a clause C that has $\mathcal{P}$, let $F := F \cup \{C\}$.*

While clause elimination procedures have been studied and exploited to a much broader extent than clause addition, the latter has already proven important both from the theoretical and the practical perspectives, as we will discuss further in Sect. 7.

For establishing concrete instantiations of clause elimination and addition procedures, redundancy properties on which such procedures are based on need to be defined. We will now review various such properties, following [11].

3.1 Notions of Redundancy

A clause is a *tautology* if it contains both x and $\neg x$ for some variable x. Given a CNF formula F, a clause $C_1 \in F$ *subsumes* (another) clause $C_2 \in F$ in F if and only if $C_1 \subset C_2$, and then C_2 is *subsumed by* C_1.

Given a CNF formula and a clause $C \in F$, (*hidden literal addition*) $\mathrm{HLA}(F, C)$ is the *unique* clause resulting from repeating the following clause extension steps until fixpoint: if there is a literal $l_0 \in C$ such that there is a clause $(l_0 \vee l) \in F_2 \setminus \{C\}$ for some literal l, let $C := C \cup \{\neg l\}$.

For a clause C, (*asymmetric literal addition*) $\mathrm{ALA}(F, C)$ is the *unique* clause resulting from repeating the following until fixpoint: if $l_1, \ldots, l_k \in C$ and there is a clause $(l_1 \vee \cdots \vee l_k \vee l) \in F \setminus \{C\}$ for some literal l, let $C := C \cup \{\neg l\}$.

Given a CNF formula F and a clause C, a literal $l \in C$ *blocks* C w.r.t. F if (i) for each clause $C' \in F$ with $\neg l \in C'$, the resolvent $C \otimes_l C'$ is a tautology, or (ii) $\neg l \in C$, i.e., C is itself a tautology. A clause C is *blocked* w.r.t. F if there is a literal l that blocks C w.r.t. F. For such an l, we say that C is blocked *on* $l \in C$ w.r.t. F.

What follows is a list of properties based on which various clause elimination procedures [11,12] can be defined.

S	(*subsumption*)	C is subsumed in F.
HS	(*hidden subsumption*)	$\mathrm{HLA}(F, C)$ is subsumed in F.
AS	(*asymmetric subsumption*)	$\mathrm{ALA}(F, C)$ is subsumed in F.
T	(*tautology*)	C is a tautology.
HT	(*hidden tautology*)	$\mathrm{HLA}(F, C)$ is a tautology.
AT	(*asymmetric tautology*)	$\mathrm{ALA}(F, C)$ is a tautology.
BC	(*blocked*)	C is blocked w.r.t. F.
HBC	(*hidden blocked*)	$\mathrm{HLA}(F, C)$ is blocked w.r.t. F.
ABC	(*asymmetric blocked*)	$\mathrm{ALA}(F, C)$ is blocked w.r.t. F.

As concrete examples, BC gives the clause elimination procedure *blocked clause elimination* (BCE) [12], and HT *hidden tautology elimination* (HTE) [11].

A relevant question is how the above-listed properties are related to each other. Especially, if any C having property $\mathcal{P}$ also has property $\mathcal{P}'$, then we know that a clause elimination procedure based on $\mathcal{P}'$ can remove at least the same clauses as a clause elimination procedure based on $\mathcal{P}$ (similarly for clause addition procedures). The relationships between these properties (first analyzed as the *relative effectiveness* in the special case of *clause elimination procedures* in [11]) are illustrated in Fig. 2. The properties prefixed with R are new and will be defined next.

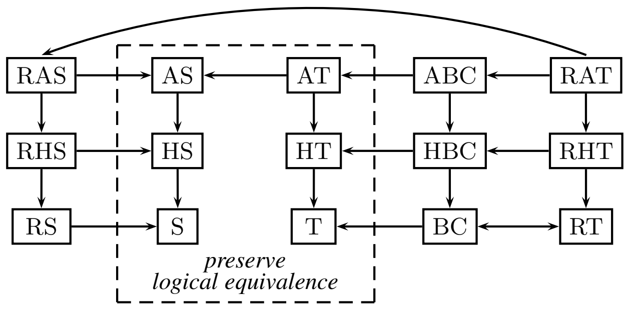


Fig. 2. Relationships between clause redundancy properties. An edge from $\mathcal{P}$ to $\mathcal{P}'$ means that any clause that has property $\mathcal{P}'$ also has property $\mathcal{P}$. A missing edge from $\mathcal{P}$ to $\mathcal{P}'$ means that there are clauses with property $\mathcal{P}'$ that do not have property $\mathcal{P}$. Clause elimination and addition procedures based on the properties inside the *preserve logical equivalence* box preserve logical equivalence under elimination and addition [11].

4 Extended Notions of Redundancy

Clause elimination procedures can be extended by using the resolution rule as a specific kind of "look-ahead step" within the procedures. This turns a specific clause elimination procedure $\mathcal{P}\mathrm{E}$ based on property $\mathcal{P}$ into the clause elimination procedure $\mathrm{R}\mathcal{P}\mathrm{E}$ based on a property $\mathrm{R}\mathcal{P}$. Analogously, a specific clause addition procedure $\mathcal{P}\mathrm{A}$ based on property $\mathcal{P}$ turns into the clause addition procedure $\mathrm{R}\mathcal{P}\mathrm{A}$ based on a property $\mathrm{R}\mathcal{P}$.

Definition 3. *Given a CNF formula F and a clause $C \in F$, C* has property $\mathrm{R}\mathcal{P}$ *iff either (i) C has the property $\mathcal{P}$, or (ii) there is a literal $l \in C$ such that for each clause $C' \in F$ with $\neg l \in C'$, each resolvent in $C \otimes_l C'$ has $\mathcal{P}$ (in this case C has* $\mathrm{R}\mathcal{P}$ on *l).*

Example 1. Consider the formula $F = (a \vee b \vee x) \wedge (\neg x \vee c \vee d) \wedge (a \vee b \vee c)$. The only resolvent of $(a \vee b \vee x)$ on x is $(a \vee b \vee c \vee d)$ which is subsumed by $(a \vee b \vee c)$. Therefore $(a \vee b \vee x)$ has property RS (resolution subsumption).

The intuition is that the "resolution look-ahead" step can reveal additional redundant clauses, resulting in the hierarchy shown in Fig. 2. Notice that the property RT (resolution tautology) is the same as the property BC (blocked).

Proposition 1. *For any CNF formula F and clause C that has* RAT *on $l \in C$ w.r.t. F, F is satisfiability-equivalent to $F \cup \{C\}$.*

Proof. By definition, since C has RAT on $l \in C$ w.r.t. F, all resolvents $C \otimes_l F_{\neg l}$ are asymmetric tautologies w.r.t. F (and w.r.t. the larger $F \cup (C \otimes_l F_{\neg l})$ as well). Hence

F is logically equivalent to $F \cup (C \otimes_l F_{\neg l})$. Now consider a truth assignment τ that satisfies F, but falsifies C. Since C is falsified by τ, and all $C' \in C \otimes_l F_{\neg l}$ are satisfied by τ due to logical equivalence of F and $F \cup (C \otimes_l F_{\neg l})$, τ satisfies at least two literals in each clause in $F_{\neg l}$ (at least one more beside $\neg l$). Hence the truth assignment τ' that is a copy of τ except for $\tau'(l) = 1$ satisfies F and C. $\square$

Proposition 2. *The set of clauses that have* RAS *is a proper subset of the set of clauses that have* RAT.

Proof. Assume a clause C has RAS on $l \in C$ w.r.t. F. If C has AS, then C has AT [11] and hence also RAT. Otherwise, take any resolvent $C' \in C \otimes_l F_{\neg l}$. By definition, C' has AS. Since clauses with AS are a proper subset of the clauses with AT, C has RAT on l w.r.t. F. Moreover, let $F := (a \vee \neg b) \wedge (\neg a \vee b)$. Now $(a \vee \neg b)$ has RAT on a w.r.t. F. However, $(a \vee \neg b)$ does not have RAS w.r.t. F. $\square$

Proposition 3. *The set of clauses that have* ABC *is a proper subset of the set of clauses that have* RAT.

Proof. Let C be clause that has ABC on $l \in C$ w.r.t. F. W.l.o.g. assume C to be non-tautological. By [11, Lemma 19], $l \in C$. Take the resolvent $C' = C \otimes_l C''$ for any $C'' \in F_{\neg l}$. First, we show that $\mathrm{ALA}(F, C) \subseteq \mathrm{ALA}(F, C')$. C' overlaps with C'' in all literals except $\neg l$. W.l.o.g. assume $C' \notin F$ (otherwise C' is subsumed by F and thus also has AT w.r.t. F). Therefore, by the definition of ALA, $l \in \mathrm{ALA}(F, C')$. Hence $C \subseteq \mathrm{ALA}(F, C')$. Due to monotonicity of ALA under the assumption $C' \notin F$, we have $\mathrm{ALA}(F, C) \subseteq \mathrm{ALA}(F, C')$. By definition of ABC, the clause $\mathrm{ALA}(F, C) \otimes_l C''$ is a tautology, and hence there is an $l' \in \mathrm{ALA}(F, C) \setminus \{l\}$ with $\neg l' \in C''$. Now, $l' \in \mathrm{ALA}(F, C')$ since $\mathrm{ALA}(F, C) \subseteq \mathrm{ALA}(F, C')$, and $\neg l' \in \mathrm{ALA}(F, C')$ since $C' = C \otimes_l C''$. Thus C' has AT on l w.r.t. F, which implies that C has RAT w.r.t. F.

For proper containment, consider the formula $F = (a \vee b \vee c \vee d) \wedge (a \vee b \vee x) \wedge (\neg x \vee c \vee d) \wedge (\neg a \vee y \vee z) \wedge (\neg b \vee y \vee \neg z) \wedge (\neg c \vee \neg y \vee z) \wedge (\neg d \vee \neg y \vee \neg z)$. No clause in F has ABC. Yet $(a \vee b \vee x)$ has RAT on x w.r.t. F. $\square$

Proposition 4. *The set of clauses which have* RHT *is a proper subset of the set of clauses that have* RAT.

Proof. Assume C has RHT on $l \in C$ w.r.t. F. If C has HT, then C has AT [11] and hence also RAT. Otherwise, take any $C' \in C \otimes_l F_{\neg l}$. By definition, C' has HT. Since clauses with HT are a proper subset of the clauses with AT, C has RAT on l w.r.t. F. Moreover, let $F := (a \vee b \vee x) \wedge (\neg x \vee c) \wedge (a \vee b \vee c) \wedge (\neg a) \wedge (\neg b) \wedge (\neg c)$. Now $(a \vee b \vee x)$ has RAT on x w.r.t. F, but $(a \vee b \vee x)$ does not have RHT. $\square$

5 Inprocessing as Deduction

We will now introduce generic rules for inprocessing CNF formulas. The rules describe inprocessing as a transition system. States in the transition system are described by tuples of the form $\varphi\,[\,\rho\,]\,\sigma$, where φ and ρ are CNF formulas, and σ is a sequence of literal-clause pairs. For inprocessing a given CNF formula F, the initial state is $F\,[\,\emptyset\,]\,\langle\rangle$, where $\emptyset$ denotes the empty CNF formula, and $\langle\rangle$ the empty sequence.

Generally, a state $\varphi\,[\,\rho\,]\,\sigma$ has the following interpretation.

- φ is a CNF formula that consists of the set of *irredundant* clauses. Irredundant means here that all clauses in φ are considered to be "hard" in the sense that, in order to satisfy the input CNF formula F, all clauses in φ are to be satisfied.
- ρ is a CNF formula that consists of *redundant clauses*. In contrast to the irredundant clauses φ, these clauses can be removed from consideration.
- σ denotes a sequence of literal-clause pairs $l{:}C$ with $l \in C$ that are required for solution reconstruction, as explained in detail in Sect. 6.1.

For some intuition on why we separate φ and ρ, note that *learned clauses*, i.e., clauses added through conflict analysis in CDCL solvers, are maintained separately from the clauses in the input formula, and can be forgotten (i.e., removed) since in pure CDCL they are entailed by the input formula. However, in the more generic context of inprocessing SAT solving captured by our framework, clauses in ρ may not be entailed by the original formula F. This is discussed in detail in Sect. 7 using clause addition as an example. In addition, for elimination techniques (such as BCE, variable elimination, and their variants) only the clauses in φ need to be considered when checking redundancy. Nevertheless, the clauses in ρ can be used for e.g. unit propagation.

5.1 Rules of Inprocessing

Our abstract framework for inprocessing SAT solving is based on four rules: LEARN, FORGET, STRENGTHEN, and WEAKEN, presented in Fig. 3. These rules characterize the set of legal next states $\varphi' \,[\, \rho' \,]\, \sigma'$ of a given current state $\varphi \,[\, \rho \,]\, \sigma$ in the form

$$\frac{\varphi \,[\, \rho \,]\, \sigma}{\varphi' \,[\, \rho' \,]\, \sigma'}.$$

Given a CNF formula F, a state $\varphi_k \,[\, \rho_k \,]\, \sigma_k$ is reachable from the state $F \,[\, \emptyset \,]\, \langle\rangle$ iff there is a sequence $\langle \varphi_0 \,[\, \rho_0 \,]\, \sigma_0, \ldots, \varphi_k \,[\, \rho_k \,]\, \sigma_k \rangle$ such that (i) $\varphi_0 = F$, $\rho_0 = \emptyset$, and $\sigma_0 = \langle\rangle$, and (ii) for each $i = 1, \ldots, k$, one of the rules in Fig. 3 allows the transition from $\varphi_{i-1} \,[\, \rho_{i-1} \,]\, \sigma_{i-1}$ to $\varphi_i \,[\, \rho_i \,]\, \sigma_i$. This sequence is called a derivation of $\varphi_k \wedge \rho_k$ from F.

The inprocessing rules are *correct* in the sense that they preserve satisfiability, i.e., starting from the state $F \,[\, \emptyset \,]\, \langle\rangle$, the following invariant holds for all $i = 1, \ldots, k$:

$$\boxed{\text{Formulas } \varphi_i \text{ and } (\varphi_i \wedge \rho_i) \text{ are } \textit{both} \text{ satisfiability-equivalent to } F.}$$

The intuition behind these rules is as follows.

LEARN Allows for introducing (*learning*) a new clause C to the current redundant formula ρ. In the generic setting, the precondition $\sharp$ is that $\varphi \wedge \rho$ and $\varphi \wedge \rho \wedge C$ are satisfiability-equivalent.

FORGET Allows for *forgetting* a clause C from the current set of redundant clauses ρ.

STRENGTHEN Allows for *strengthening* φ by moving a clause C in the redundant formula $\rho \wedge C$ to φ.

WEAKEN Allows for *weakening* φ by moving a clause C in the current irredundant formula $\varphi \wedge C$ to ρ. In the generic setting, the precondition $\flat$ is that φ and $\varphi \wedge C$ are satisfiability-equivalent. (The literal l is related to instantiations of the rule based on specific redundancy properties, as further explained in Sect. 6 and Sect. 7.)

$$\frac{\varphi\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho \wedge C\,]\,\sigma}\ \sharp \qquad \frac{\varphi\,[\,\rho \wedge C\,]\,\sigma}{\varphi\,[\,\rho\,]\,\sigma} \qquad \frac{\varphi\,[\,\rho \wedge C\,]\,\sigma}{\varphi \wedge C\,[\,\rho\,]\,\sigma} \qquad \frac{\varphi \wedge C\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho \wedge C\,]\,\sigma, l{:}C}\ \flat$$

LEARN FORGET STRENGTHEN WEAKEN

Fig. 3. Inprocessing rules

Notice that for unsatisfiable CNF formulas the generic precondition $\sharp$ allows for learning the empty clause to φ in a single step. Similarly, for satisfiable CNF formulas the generic precondition $\flat$ allows for weakening φ by moving *all* clauses in φ to ρ. However, in practice mostly polynomial-time checkable redundancy properties are of interest. Such properties are further discussed in Sect. 6 and Sect. 7.

Proposition 5. *The inprocessing rules in Fig. 3 are sound and complete in that:*

(i) *If F is unsatisfiable, then there is a derivation of an unsatisfiable $\varphi_k \wedge \rho_k$, where $k \geq 0$, from F using the rules (completeness).*

(ii) *If there is a derivation of an unsatisfiable $\varphi_k \wedge \rho_k$, where $k \geq 0$, from F using the rules, then F is unsatisfiable (soundness).*

Proof. (i) Since F is unsatisfiable, the LEARN rule can be used for learning the trivially unsatisfiable empty clause. (ii) We observe the following for any $i = 1, \ldots, k$. If LEARN is applied to enter state $\varphi_i\,[\,\rho_i\,]\,\sigma_i$ from $\varphi_{i-1}\,[\,\rho_{i-1}\,]\,\sigma_{i-1}$, by the precondition $\sharp$, φ_{i-1} and $\varphi_i \wedge \rho_i$ are satisfiability-equivalent. If STRENGTHEN or WEAKEN is applied, we have $\varphi_{i-1} \wedge \rho_{i-1} = \varphi_i \wedge \rho_i$. If FORGET is applied, we have $\varphi_{i-1} \wedge \rho_{i-1} \models \varphi_i \wedge \rho_i$. The claim then follows by induction on $i = k, \ldots, 1$. $\square$

One could question whether the precondition $\sharp$ of LEARN, i.e., $\varphi \wedge \rho$ and $\varphi \wedge \rho \wedge C$ are satisfiability-equivalent, could be weakened to "φ and $\varphi \wedge C$ are satisfiability-equivalent". In other words, must the redundant clauses in ρ be taken into account for LEARN? To observe that ρ must indeed be included in $\sharp$, consider the CNF formula consisting of the single clause (a). From the initial state $a\,[\emptyset]\,\langle\rangle$ we obtain $\emptyset\,[a]\,\langle\rangle$ through WEAKEN. In case ρ were ignored in $\sharp$, it would then be possible to apply LEARN and derive $\emptyset\,[a \wedge \neg a]\,\langle\rangle$. However, this would violate the invariant of preserving satisfiability, since $a \wedge \neg a$ is unsatisfiable.

6 Instantiating the Rules based on RAT

In contrast to the very generic preconditions $\sharp$ and $\flat$ under which the inprocessing rules were defined in the previous section, in practical SAT solving redundant clauses are learned and forgotten based on polynomial-time computable redundancy properties. In this section we give an instantiation of the inprocessing rules based on the polynomial-time computable property RAT. RAT is of special interest to us since, as will be shown in Sect. 7, known SAT solving techniques, including preprocessing, inprocessing, clause learning, and resolution, can be captured even when restricting the inprocessing rules using RAT. Moreover, under this property, a model of the original formula can be reconstructed in linear-time based on any model of any derivable φ_k using σ_k. This is important from the practical perspective due to the fact that in many applications a satisfying assignment for the original input formula F is required.

Preconditions based on RAT**.** The preconditions of the inprocessing rules based on the property RAT are the following for a given state $\varphi_i\,[\,\rho_i\,]\,\sigma_i$.

$$\boxed{\text{LEARN: } \sharp \text{ is "} C \text{ has RAT w.r.t. } \varphi_i \wedge \rho_i\text{".}}$$

Notice that LEARN under this precondition does not preserve logical equivalence. For example, consider the formula $F = (a \vee b)$. The LEARN rule can change $(a \vee b)\,[\emptyset]\,\langle\rangle$ into $(a \vee b)\,[C]\,\langle\rangle$, with $C = (\neg a \vee \neg b)$, since C has RAT on $\neg a$ w.r.t. F. The truth assignment $\tau = \{a = 1, b = 1\}$ satisfies F but does not satisfy $F \wedge C$.

$$\boxed{\text{WEAKEN: } \flat \text{ is "} C \text{ has RAT on } l \text{ w.r.t. } \varphi_i\text{".}}$$

Through weakening φ_i by moving a clause $C \in \varphi_i$ to ρ_{i+1}, the new φ_{i+1} may have more models than φ_i, since RAT does not preserve logical equivalence.

6.1 Solution Reconstruction

When the WEAKEN rule is used for a transition from a state $\varphi_i\,[\,\rho_i\,]\,\sigma_i$ to a state $\varphi_{i+1}\,[\,\rho_{i+1}\,]\,\sigma_{i+1}$, the set of models of φ_{i+1} can be a proper superset of the set of models of φ_i. For the practically relevant aspect of mapping any model of φ_{i+1} back to a model of φ_i, a literal pair $l{:}C$, where C is the clause moved from φ_i to ρ_{i+1}, is concatenated to the solution reconstruction stack σ_{i+1}. This is important when the redundancy property used does not guarantee preserving logical equivalence. More concretely, this is required if C has RAT but not e.g. AT.

For certain polynomial-time checkable redundancy properties, σ can be used for mapping models back to models of the original formula in linear time, as explained next. We describe a generic *model reconstruction algorithm* that can be applied in conjunction with the inprocessing rules in case the preconditions $\sharp$ and $\flat$ of LEARN and WEAKEN are restricted to RAT. In particular, for any CNF formula F and state $\varphi\,[\,\rho\,]\,\sigma$ that is reachable from $F\,[\,\emptyset\,]\,\langle\rangle$ using the inprocessing rules, given a model τ of φ, the reconstruction algorithm (Fig. 4) outputs a model of F solely based on σ and τ.

```
Reconstruction (literal-clause pair sequence $\sigma$, model $\tau$ of $\varphi$)
    while $\sigma$ is not empty do
        remove the last literal-clause pair $l{:}C$ from $\sigma$
        if $C$ is not satisfied by $\tau$ then $\tau := (\tau \setminus \{l = 0\}) \cup \{l = 1\}$
    return $\tau$
```

Fig. 4. Pseudo-code of the model reconstruction algorithm.

While the reconstruction algorithm may leave some variables unassigned in the output assignment (model of F), such variables can be arbitrarily assigned afterwards for establishing a full model of F.

Example 2. Consider the state $\varphi_i\,[\,\rho_i\,]\,\sigma_i$ with $\varphi_i = (a \vee b) \wedge (\neg a \vee \neg b)$, $\rho_i = \emptyset$ and $\sigma_i = \langle\rangle$. Apply WEAKEN to reach $\varphi_{i+1}\,[\,\rho_{i+1}\,]\,\sigma_{i+1}$, where $\varphi_{i+1} = (a \vee b)$, $\rho_{i+1} = (\neg a \vee \neg b)$, and $\sigma_{i+1} = \langle \neg a{:}(\neg a \vee \neg b)\rangle$. The assignment $\tau = \{a = 1, b = 1\}$ satisfies φ_{i+1} but not φ_i. The model reconstruction procedure will transform τ into $\{a = 0, b = 1\}$ which satisfies φ_i.

Proposition 6. *Given any CNF formula F, if a state $\varphi\,[\,\rho\,]\,\sigma$ is derivable from F using the inprocessing rules under preconditions based on* RAT*, then, given any model τ of φ,* $\mathsf{Reconstruction}(\sigma_i, \tau)$ *returns a model of F.*

Proof. Follows from the proof of Proposition 1. Assume that $l{:}C$ is the last element in σ_i, τ is the current truth assignment, and that WEAKEN was applied to move C from φ_{i-1} to ρ_i based on the fact that C has RAT on l w.r.t. φ_{i-1}. By the proof of Proposition 1, there are at least two literals that are satisfied by τ in every clause containing $\neg l$ in $\varphi_{i-1} \setminus \{C\}$. Hence, in case $\tau(l) = 0$, we can flip this assignment to $\tau(l) = 1$. □

Interestingly, due to the generality of the inprocessing rules—as explained in the next section—this reconstruction algorithm covers model reconstruction for various simplification techniques that do not preserve logical equivalence, including specific reconstruction algorithms proposed for different cause elimination techniques [18,11] and variable elimination [18], and combinations thereof with other important techniques such as equivalence reasoning [24,3].

7 Capturing SAT Solving Techniques with the Inprocessing Rules

In this section we show how various existing inference techniques—including both known techniques and novel ideas—can be expressed as simple combinations of the LEARN, FORGET, STRENGTHEN, and WEAKEN rules. One should notice, however, that the inprocessing rules can be shown to naturally capture further inprocessing techniques. However, due to the page limit we are unable to discuss further techniques within this version of the paper. We also give examples of how incorrect variants of these techniques can be detected.

Clause elimination procedures based on redundancy property $\mathcal{P}$ can be expressed as deriving $\varphi\,[\,\rho\,]\,\sigma$ from $\varphi \wedge C\,[\,\rho\,]\,\sigma$ in a single step with the precondition that C has the property $\mathcal{P}$ w.r.t. φ. One step of clause elimination is simulated by two application steps of the inprocessing rules: 1. apply WEAKEN to move a redundant clause from φ to ρ; 2. apply FORGET to remove C from ρ. As explained in Sect. 6, the generic inprocessing rules can be instantiated using RAT as the redundancy property of the preconditions $\sharp$ and $\flat$. Since RAT covers all of the other clause redundancy properties discussed in Sect. 3 and 4 (such as blocked clauses, hidden tautologies, etc; also recall Fig. 2), it follows that all of the clause elimination procedures based on these properties are captured by our inprocessing rules, even when restricting the precondition to RAT.

As an example of incorrect clause elimination, consider the idea of eliminating C if it has the property $\mathcal{P}$ w.r.t. $\varphi \wedge \rho$ (and not w.r.t. just φ), allowing weakening φ based on ρ, i.e., also in case a clause in ρ subsumes C. This would allow using e.g. redundant learned clauses in ρ, which can be forgotten later on, for weakening φ. To see that this variant is incorrect consider $\varphi_i\,[\,\rho_i\,]\,\sigma_i$ where $\varphi_i = (a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee b \vee c) \wedge (a \vee b \vee \neg c)$ and $\rho_i = \emptyset$. Note that φ_i is unsatisfiable. The clause $(a \vee b)$ has AT w.r.t. φ_i, since $\mathrm{ALA}(\varphi_i, (a \vee b))$ contains all literals, and hence applying LEARN gives $\varphi_{i+1} = \varphi_i$ and $\rho_{i+1} = (a \vee b)$. Now, $(a \vee b) \in \rho_{i+1}$ subsumes $(a \vee b \vee c) \in \varphi_{i+1}$, and incorrectly applying WEAKEN would give $\varphi_{i+2} = \varphi_{i+1} \setminus (a \vee b \vee c)$ and $\rho_{i+2} = \rho_{i+1} \wedge (a \vee b \vee c)$. However, φ_{i+2} is satisfiable, and the satisfiability-equivalence invariant is broken since $\varphi_{i+2} \wedge \rho_{i+2}$ is unsatisfiable. As a consequence, *it is not correct to use the clauses in ρ to eliminate an irrredundant clause* (such as hidden or asymmetric tautologies, blocked clauses, etc.), *unless the clauses, based on which the eliminated clause is redundant, are added to φ or are already part of φ.*

Pure Literal Elimination is an additional well-known clause elimination procedure: derive $\varphi\,[\,\rho\,]\,\sigma$ from $\varphi \wedge C\,[\,\rho\,]\,\sigma$ given that C contains a *pure literal* l (such that $\neg l$ does not appear in φ). It is easy to observe that this rule is also covered by our inprocessing rule: Any clause in φ that contains a pure literal l has RT (and thus RAT) on l w.r.t. φ. Notice that due to the WEAKEN precondition, only the irredundant clauses φ need to be considered, i.e., redundant (e.g., learned) clauses can still contain $\neg l$.

Clause addition procedures based on redundancy property $\mathcal{P}$ can be expressed as deriving $\varphi\,[\,\rho \wedge C\,]\,\sigma$ from $\varphi\,[\,\rho\,]\,\sigma$ in a single step with the precondition that C has the property $\mathcal{P}$ w.r.t. $\varphi \wedge \rho$. One step of clause addition is simulated by applying LEARN to add C to ρ. Similarly to clause elimination, the generic inprocessing rules can be instantiated using RAT as the redundancy property of the precondition $\sharp$. Again, since RAT covers all of the other clause redundancy properties discussed in Sect. 3 and 4, it follows that all of the clause addition procedures based on these properties are captured by the generic inprocessing rules.

Notice that some clause addition procedures do not preserve logical equivalence (recall Fig. 2), and hence can restrict the set of models of $\varphi \wedge \rho$. For such procedures, the inprocessing rules can be applied for checking correctness. As an example, consider *blocked clause addition* (BCA): for adding a clause C to ρ, it is required that C is blocked w.r.t. $\varphi \wedge \rho$. If C is only blocked w.r.t. φ, then BCA is not sound. Consider the formula $\varphi_0 = (a \vee \neg b) \wedge (\neg a \vee b) \wedge (a \vee c) \wedge (\neg c \vee b) \wedge (\neg a \vee \neg c)$. Notice that $(\neg a \vee \neg c)$ has RT (is blocked) on $\neg c$ w.r.t. φ_0. Hence $(\neg a \vee \neg c)$ can be moved from φ_0 to be part of ρ_1 by applying the WEAKEN rule: $\varphi_1 = \varphi_0 \setminus \{(\neg a \vee \neg c)\}$, $\rho_1 = \rho_0 \cup \{(\neg a \vee \neg c)\}$, and $\sigma_1 = \sigma_0 \cup \{\neg c{:}(\neg a \vee \neg c)\}$. Now the clause $(c \vee \neg b)$ is a RT on c w.r.t. φ_1, but not w.r.t. $\varphi_1 \wedge \rho_1$. Adding $(c \vee \neg b)$ to ρ to get $\rho_2 = \rho_1 \cup \{(c \vee \neg b)\}$ and $\varphi_2 = \varphi_1$ makes $\varphi_2 \wedge \rho_2$ unsatisfiable.

This brings us to an interesting observation of the framework. Continuing the above, if $(\neg a \vee \neg c)$ was removed (FORGET) after moving it to ρ (so $\rho_2 = \rho_1 \setminus \{(\neg a \vee \neg c)\}$, $\varphi_2 = \varphi_1$, and $\sigma_2 = \ \sigma_1$), then adding $(c \vee \neg b)$ to ρ via LEARN would be allowed ($\rho_3 = \rho_2 \setminus \{(\neg c \vee \neg b)\}$, $\varphi_3 = \varphi_2$, and $\sigma_3 = \ \sigma_2$) since $(c \vee \neg b)$ has RT on c w.r.t. $\varphi_3 \wedge \rho_3$. Now $\varphi_3 \wedge \rho_3 \wedge \mathrm{CNF}(\sigma_3)$ is unsatisfiable, where $\mathrm{CNF}(\sigma_3)$ is the conjunction of clauses in σ_3. Yet this does not cause a problem. The reconstruction method ensures that for every assignment satisfying φ a model of the original formula F can be constructed. Thus it also holds for assignments that satisfy $\varphi \wedge \rho$. *This illustrates that* LEARN *may add clauses to ρ that are not entailed by the clauses in the original formula*.

Clause Learning based on conflict graphs, which is central in modern CDCL solvers, can be simulated by the inprocessing rules. Since any conflict clause based on a conflict graph is derivable by trivial resolution from the current clause database [25], the inprocessing rules can simulate clause learning by simulating the steps of the resolution derivation, as explained next.

Resolution can also be simulated by the inprocessing rules in a straightforward way: For any φ, $(C \vee D)$ is an AT w.r.t. $\varphi \wedge (C \vee x) \wedge (D \vee \neg x)$, and thus $(C \vee D)$ can be learned by applying LEARN. This implies that *all* resolution-based simplification techniques can also be simulated. An example is *Hyper Binary Resolution* (HBR) [3]: Given a clause of the form $(l \vee l_1 \cdots \vee l_k)$ and k binary clauses of the form $(l' \vee \neg l_i)$, where $1 \leq i \leq k$, the hyper binary resolution rule allows to infer the *hyper binary resolvent* $(l \vee l')$ in one step. In essence, HBR simply encapsulates a sequence of specifically related resolution steps into one step.

Variable Elimination (VE) can also be simulated by our inprocessing rules. When applied in a bounded setting [7], VE is currently one of the most effective preprocessing techniques applied in SAT solvers. Variable elimination as a general version of VE for inprocessing can be characterized as the rule

$$\frac{\varphi \wedge \varphi_x \wedge \varphi_{\neg x} \, [\, \rho \wedge \rho_x \wedge \rho_{\neg x} \,] \, \sigma}{\varphi \wedge \ \varphi_x \otimes_x \varphi_{\neg x} \ [\, \rho \,] \, \sigma, \ x{:}\varphi_x, \neg x{:}\varphi_{\neg x}},$$

where F_l denotes the clauses in a CNF formula F that contain literal l, and $F_l \otimes_l F_{\neg l}$ is the lifting of the resolution operator to sets of clauses. Essentially, VE eliminates a variable x by producing all possible resolvents w.r.t. x, and removes at the same time all clauses containing x. Although not discussed in earlier work, our characterization takes into account the common practice that resolvents due to redundant clauses in ρ do not need to be produced.

To see that our inprocessing rules simulate VE, first apply LEARN to add the resolvents $\varphi_x \otimes \varphi_{\neg x}$ to ρ (all resolvents have AT w.r.t. φ). Second, apply STRENGTHEN to move the resolvents from ρ to φ. Now all clauses in φ_x have RS on x w.r.t. φ, and all clauses in $\varphi_{\neg x}$ have RS on $\neg x$ w.r.t. φ, and hence WEAKEN can be applied for making the clauses in φ_x and $\varphi_{\neg x}$ redundant, after which they can be removed using FORGET.

Notice that two variants of VE are distinguished [7]. The first, VE by *clause distribution* adds all the clauses of $\varphi_x \otimes \varphi_{\neg x}$ to φ. The second, VE by *substitution* adds only a subset of $\varphi_x \otimes \varphi_{\neg x}$ to φ in a satisfiability-preserving way. As a consequence, the latter variant may reduce the amount of unit propagations in the resulting formula compared to the former. However, under the inprocessing rules, the clauses produced by *clause distribution* but not by *substitution* can alternatively be added to ρ instead of φ, so that these clauses can be used subsequently for unit propagation but can still be considered redundant and thus be ignored in consecutive VE steps.

Partial Variable Elimination, as described below, is a novel variant of VE, which can also be naturally expressed via our inprocessing rules. Given a variable x and two subsets of clauses $S_x \subset \varphi_x$ and $S_{\neg x} \subset \varphi_{\neg x}$, if there are non-empty S_x and $S_{\neg x}$ such that all resolvents of $S_x \otimes (\varphi_{\neg x} \setminus S_{\neg x})$ and $S_{\neg x} \otimes (\varphi_x \setminus S_x)$ are tautologies, then we can apply VE *partially* by replacing $S_x \wedge S_{\neg x}$ in φ by $S_x \otimes S_{\neg x}$. We refer to this as *Partial Variable Elimination* (PVE). In practice, the VE rule is bounded by applying it only when the number of clauses is not increased. It is actually possible that PVE on x decreases the number of clauses, e.g., if $|S_x| = 1$ or $|S_{\neg x}| = 1$, while VE on x would increase the number of clauses. The correctness of PVE is immediate by the inprocessing rules, using a similar argument as in the case of VE.

Extended Resolution can also be simulated. This shows that LEARN, although perhaps not evident by its simple definition, is extremely powerful even when restricting the precondition to RAT only.

For a given CNF formula F, the *extension rule* [23] allows for iteratively adding definitions of the form $x \equiv a \wedge b$ (i.e. the CNF formula $(x \vee \neg a \vee \neg b) \wedge (\neg x \vee a) \wedge (\neg x \vee b)$) to F, where x is a new variable and a, b are literals in the current formula. The resulting formula $F \wedge E$ then consists of the original formula F and the *extension* E, the conjunction of the clauses iteratively added to F using the extension rule. In *Extended Resolution* [23] one can first apply the *extension rule* to add a conjunction of clauses (an *extension*) E to a CNF formula F, before using the resolution rule to construct a resolution proof of $F \wedge E$. This proof system is extremely powerful: surpassing the power of Resolution, it can even polynomially simulate extended Frege systems.

However, it is easy to observe that the LEARN rule simulates the extension rule: the clause $(x \vee \neg a \vee \neg b)$ has RAT on x w.r.t. $\varphi \wedge \rho$ and can thus be added to ρ by applying LEARN. The clauses $(\neg x \vee a)$ and $(\neg x \vee b)$ have RAT on $\neg x$ w.r.t. $\varphi \wedge (x \vee \neg a \vee \neg b) \wedge \rho$.

From a practical perspective, it follows that our inprocessing framework captures also the deduction applied in the recently proposed extensions of CDCL solvers that apply the Extension rule in a restricted fashion [26,27].

Finally, we would like to point out that the inprocessing rules capture various additional techniques that have proven important in practice. While we are unable (due to the page limit) to provide a more in-depth account of these techniques and how they are simulated by the inprocessing rules, such techniques include (as examples) *self-subsumption* (which has proven important both when combined with variable elimination [7] and when applied during search [28,29]), equivalence reasoning [24,3], including e.g. equivalent literal substitution, and also more recent techniques that can be defined for removing and adding literals from/to clauses (such as *hidden literal elimination* [13]).

8 Conclusion

Guaranteeing correctness of new inference techniques developed and implemented in state-of-the-art SAT solvers is becoming increasingly non-trivial as complex combinations of inference techniques are implemented within the solvers. We presented an abstract framework that captures the inference of inprocessing SAT solvers via four clean inference rules, providing a unified generic view to inprocessing, and furthermore captures sound solution reconstruction in a unified way. In addition to providing an in-depth understanding of the inferences underlying inprocessing solvers, we believe that this framework opens up possibilities for developing novel inprocessing and learning techniques that may lift the performance of SAT solvers even further.

References

1. Marques-Silva, J.P., Sakallah, K.A.: GRASP: a search algorithm for propositional satisfiability. IEEE Trans. Computers **48**(5) (1999) 506–521
2. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proc. DAC, ACM (2001) 530–535
3. Bacchus, F.: Enhancing Davis Putnam with extended binary clause reasoning. In: Proc. AAAI, AAAI Press (2002) 613–619

4. Bacchus, F., Winter, J.: Effective preprocessing with hyper-resolution and equality reduction. In: Proc. SAT 2003. Volume 2919 of LNCS., Springer (2004) 341–355
5. Subbarayan, S., Pradhan, D.K.: NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In: Proc. SAT. Volume 3542 of LNCS., Springer (2005)
6. Gershman, R., Strichman, O.: Cost-effective hyper-resolution for preprocessing CNF formulas. In: Proc. SAT. Volume 3569 of LNCS., Springer (2005) 423–429
7. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Proc. SAT. Volume 3569 of LNCS., Springer (2005) 61–75
8. Jin, H., Somenzi, F.: An incremental algorithm to check satisfiability for bounded model checking. Electronic Notes in Theoretical Computer Science **119**(2) (2005) 51–65
9. Han, H., Somenzi, F.: Alembic: An efficient algorithm for CNF preprocessing. In: Proc. DAC, IEEE (2007) 582–587
10. Piette, C., Hamadi, Y., Saïs, L.: Vivifying propositional clausal formulae. In: Proc. ECAI, IOS Press (2008) 525–529
11. Heule, M.J.H., Järvisalo, M., Biere, A.: Clause elimination procedures for CNF formulas. In: Proc. LPAR-17. Volume 6397 of LNCS., Springer (2010) 357–371
12. Järvisalo, M., Biere, A., Heule, M.J.H.: Simulating circuit-level simplifications on CNF. Journal of Automated Reasoning (2012) OnlineFirst 2011.
13. Heule, M.J.H., Järvisalo, M., Biere, A.: Efficient CNF simplification based on binary implication graphs. In: Proc. SAT. Volume 6695 of LNCS. (2011) 201–215
14. Heule, M.J.H., Järvisalo, M., Biere, A.: Covered clause elimination. In: LPAR-17 Short Papers. (2010) `http://arxiv.org/abs/1011.5202`.
15. Biere, A.: P{re,i}coSAT@SC'09. In: SAT 2009 Competitive Event Booklet. (2009)
16. Soos, M.: CryptoMiniSat 2.5.0, SAT Race 2010 solver description (2010)
17. Biere, A.: Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. FMV Technical Report 10/1, Johannes Kepler University, Linz, Austria (2010)
18. Järvisalo, M., Biere, A.: Reconstructing solutions after blocked clause elimination. In: Proc. SAT. Volume 6175 of LNCS., Springer (2010) 340–345
19. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). Journal of the ACM **53**(6) (2006) 937–977
20. Nieuwenhuis, R., Oliveras, A.: On SAT modulo theories and optimization problems. In: Proc. SAT. Volume 4121 of LNCS., Springer (2006) 156–169
21. Larrosa, J., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: A framework for certified boolean branch-and-bound optimization. Journal of Automated Reasoning **46**(1) (2011)
22. Andersson, G., Bjesse, P., Cook, B., Hanna, Z.: A proof engine approach to solving combinational design automation problems. In: Proc. DAC, ACM (2002) 725–730
23. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of Reasoning 2. Springer (1983) 466–483
24. Li, C.M.: Integrating equivalency reasoning into Davis-Putnam procedure. In: Proc. AAAI, AAAI Press (2000) 291–296
25. Beame, P., Kautz, H.A., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. J. Artif. Intell. Res. **22** (2004) 319–351
26. Audemard, G., Katsirelos, G., Simon, L.: A restriction of extended resolution for clause learning SAT solvers. In: Proc. AAAI, AAAI Press (2010)
27. Huang, J.: Extended clause learning. Artificial Intelligence **174**(15) (2010) 1277–1284
28. Han, H., Somenzi, F.: On-the-fly clause improvement. In: Proc. SAT. Volume 5584 of LNCS., Springer (2009) 209–222
29. Hamadi, Y., Jabbour, S., Saïs, L.: Learning for dynamic subsumption. In: Proc. ICTAI, IEEE (2009) 328–335