Incremental Inprocessing in SAT Solving

Katalin Fazekas[1(✉)], Armin Biere[1], Christoph Scholl[2]

[1] Johannes Kepler University, Linz, Austria
katalin.fazekas@jku.at, armin.biere@jku.at
[2] Albert–Ludwigs–University, Freiburg, Germany
scholl@informatik.uni-freiburg.de

Abstract. Incremental SAT is about solving a sequence of related SAT problems efficiently. It makes use of already learned information to avoid repeating redundant work. Also preprocessing and inprocessing are considered to be crucial. Our calculus uses the most general redundancy property and extends existing inprocessing rules to incremental SAT solving. It allows to automatically reverse earlier simplification steps, which are inconsistent with literals in new incrementally added clauses. Our approach to incremental SAT solving not only simplifies the use of inprocessing but also substantially improves solving time.

1 Introduction

Solving a sequence of related SAT problems incrementally [1,2,3,4] is crucial for the efficiency of SAT based model checking [5,6,7,8], and important in many domains [9,10,11,12]. Utilizing the effort already spent on a SAT problem by keeping learned information (such as variable scores and learned clauses) can significantly speed-up solving similar problems. Equally important are formula simplification techniques such as variable elimination, subsumption, self-subsuming resolution, and equivalence reasoning [13,14,15,16].

These simplifications are not only applied before the problem solving starts (*preprocessing*), but also periodically during the actual search (*inprocessing*) [17]. In this paper we focus on how to efficiently combine simplification techniques with incremental SAT solving.

Consider the SAT problem $F^0 = (a \vee b) \wedge (\neg a \vee \neg b)$. Both clauses are redundant and can be eliminated by for instance variable or blocked clause elimination [14,16]. The resulting empty set of clauses is of course satisfiable and the SAT solver could for example simply just assign *false* to both variable as a solution. That is of course not a satisfying assignment of F^0, but can be transformed into one by solution reconstruction [17,18], taking eliminated clauses into account. As we will see later, this would set the truth value of either a or b to true.

Now consider the SAT problem $F^1 = (a \vee b) \wedge (\neg a \vee \neg b) \wedge (\neg a) \wedge (\neg b)$ which is actually an extension of F^0 with the clauses $(\neg a)$ and $(\neg b)$. Simply adding them to our simplified F^0 (i.e. to the empty set of clauses) would result in a formula that again is satisfied by assigning false to each variable. However, using solution reconstruction on that assignment leads to the same solution as before, one that

satisfies $(a \vee b)$, and thus would actually falsify $(\neg a)$ or $(\neg b)$. The solver would incorrectly report that F^1 is satisfiable, and even return an invalid solution. Thus naively using inprocessing in an incremental setting is not sound.

Obviously one can just give up on incrementality and simply solve F^1 from scratch but with pre- and inprocessing. Another trivial approach is to use learned information from solving F^0, but then disable inprocessing. A compromise is to disallow inprocessing partially by *freezing* [7] those variables that are not allowed to be involved in simplifications ("Don't Touch" variables in [8]). This is rather error-prone and cumbersome for the user, and even often impossible [19].

Our approach benefits from most inprocessing techniques, without freezing any variables. It identifies potential problems between an eliminated clause, such as $(a \vee b)$ in the example, and new clauses, such as $(\neg a)$ and $(\neg b)$. In such a case it moves back the eliminated clause to the formula before adding the new clauses. This greatly simplifies the way how incremental SAT solvers can be used.

The specialized approach in [19] focuses on three preprocessing techniques (variable elimination, subsumption and self-subsumption of [14]). It applies a preprocessing phase before each incremental SAT call. Instead of that, we adapt and extend the framework of [17] and present a generic calculus which allows to combine a much broader set of pre- and inprocessing techniques with incremental SAT solving. Actually, we use the *most general redundancy property* [20,21] that covers not only all techniques in [19], but also provides optimized procedures for equivalence literal reasoning [13] and even blocked clause elimination [16]. However, we do not yet support techniques that remove models, such as blocked clause addition [17,22,23,24] (neither does [19]).

Our approach is also more precise than [19] since it allows to distinguish simplification steps applied on different phases of variables, i.e. we provide a literal- and not just variable-based approach. On the practical side, beyond enabling a wider range of pre- and inprocessing techniques, we present a simple algorithm, which yields an efficient implementation as confirmed by our experiments. Using dedicated techniques for inprocessing under assumptions, as [25] extends [19] based on [26], is orthogonal to the approach presented in this paper.

After preliminaries we present our new rules for incremental SAT solving in Sect. 3 which are proven correct in Sect. 4. We discuss implementation details in Sect. 5 followed by experimental results in Sect. 6 before we conclude in Sect. 7.

2 Preliminaries

Satisfiability A *literal* is either a Boolean variable (v), or its negation ($\neg v$). A *clause* is a disjunction of literals, and a *formula* in conjunctive normal form (CNF) is a conjunction of clauses. If convenient, we consider a clause as a set of literals and a formula as a set of clauses. A (partial) *truth assignment* τ is a consistent set of literals assigning truth values to variables as follows. In case $v \in \tau$, then v is assigned *true* by τ (denoted as $\tau(v) = \top$), while if $\neg v \in \tau$, then v is assigned *false* ($\tau(v) = \bot$). A truth assignment *satisfies* a literal ℓ (denoted as $\tau(\ell) = \top$) if $\ell \in \tau$ and it *falsifies* it (denoted as $\tau(\ell) = \bot$) if $\neg\ell \in \tau$, where

$\neg\ell = \neg v$ if $\ell = v$ and $\neg\ell = v$ if $\ell = \neg v$. Neither satisfied nor falsified literals are *undefined*. A clause is a *tautology* if it contains both a literal and the negation of it. The application of a truth assignment τ to an arbitrary formula F, denoted as $\tau(F)$ or $F_{|\tau}$, is defined as usual. When it is convenient, we will use sets of literals directly as truth assignments. We further use $\tau_1 \circ \tau_2$ to denote the composition of truth assignments τ_1 and τ_2 in the natural way, i.e., $(\tau_1 \circ \tau_2)(F) = \tau_1(\tau_2(F))$.

The *satisfiability problem* (SAT) for a CNF asks whether there is a truth assignment such that all clauses contain at least one satisfied literal. A truth assignment satisfying a formula is also called a *model*. Formulas F_1, F_2 are *logically equivalent*, denoted as $F_1 \equiv F_2$, if they are satisfied by exactly the same truth assignments, while they are *satisfiability equivalent*, denoted as $F_1 \equiv_{sat} F_2$, if both of them are satisfiable or both of them are unsatisfiable.

Incremental SAT problems An incremental SAT problem $\mathcal{F}$ is a sequence of clause sets $\langle \Delta_0, \ldots, \Delta_n \rangle$. In phase $i = 0, \ldots, n$ the task is to determine the satisfiability of $F^i = \wedge_{s=0\ldots i} \Delta_s$, the conjunction of all added clauses up to this point. If F^i is unsatisfiable, then F^j for all $j > i$ is unsatisfiable as well, as each iteration just augments the set of clauses. The focus of this paper is on the case where F^i is satisfiable. We rely on the common approach to always choose the given assumptions, literals that are assumed to be true in a phase, as first decisions during search and thus w.l.o.g. do not need to consider assumptions in this paper explicitly. See Minisat [3] for implementation details or [27,28] for abstract solvers following that approach. However, the variables of assumptions are not allowed to be eliminated or occur in witnesses (e.g., as blocking literal in blocked clauses [16]), i.e., they have to be considered *frozen* [7,8] internally.

Example 1. Consider the incremental SAT problem $\mathcal{F} = \langle \{(a \vee b)\}, \{a, b\} \rangle$. It consists of two SAT queries: $F^0 = (a \vee b)$ and $F^1 = F^0 \wedge a \wedge b = (a \vee b) \wedge a \wedge b$.

Redundancy in SAT Inprocessing in SAT solving relies on the concept of adding and removing *redundant* clauses. To simplify matters, in this paper we use the most general redundancy notion [20,21]. It covers most techniques used in current SAT solvers including resolution asymmetric tautology (RAT), which was used in the original work on inprocessing [17]. As [20,21] points out, any clause redundancy can produce a "witness", e.g. a blocking literal in case of a blocked clause, which allows polynomial solution reconstruction. The following two essential definitions are adapted from [20,21]:

Definition 1 (Witness Labelled Clause). *A set of literals ω and a clause C such that $\omega \cap C \neq \emptyset$ is called a witness labelled clause and written as $(\omega : C)$.*

A witness is a set of literals and can be interpreted as a partial truth assignment. With this interpretation, the truth assignment α which falsifies a given clause C but is undefined otherwise is also written as $\alpha = \neg C$.

Definition 2 (Clause Redundancy). *A witness labelled clause $(\omega : C)$ is redundant with respect to a formula F if $\omega(C) = \top$ and $F|_\alpha \models F|_\omega$ for $\alpha = \neg C$. This is also denoted as $F \wedge C \equiv^{\omega}_{sat} F$.*

As has been shown in [20,21], this is the most general notion of redundancy and allows to simulate all other types of clause redundancy. The corresponding proof (of Thm. 1 in [20,21]) allows to "fix" an assignment using the witness. We formalize that part of the proof and extend it to *partial* truth assignments, which allows to use partial truth assignments in the witness reconstruction process satisfying only the simplified formula and is further useful to produce a partial satisfying assignment after reconstruction (used for instance in [29]).

Proposition 1. *Assume* $F \wedge C \equiv^{\omega}_{sat} F$ *as above. Let* τ *be a (partial) truth assignment with* $\tau(F) = \top$ *and* $\tau(C) \neq \top$*. Then* $\gamma(F \wedge C) = \top$ *with* $\gamma = \tau \circ \omega$*.*

Proof. Clearly $\gamma(C) = \omega(C) = \top$. We need to show $\gamma(D) = \top$ for all $D \in F$. Observe $\alpha \circ \tau = \tau \circ \alpha$ with $\alpha = \neg C$ since $\tau(C) \neq \top$ and α and τ are consistent. Thus, $\top = \tau(F) = (\alpha \circ \tau)(F) = (\tau \circ \alpha)(F) = (\tau \circ \alpha)(F \wedge \neg C) = \tau(F|_\alpha)$ since $\alpha(\neg C) = \top$. Using $F|_\alpha \models F|_\omega$ and because $F|_\omega$ remains satisfied for all extensions of τ, we get $\top = (\beta \circ \tau)(F|_\omega) = (\beta \circ \tau \circ \omega)(F) = (\beta \circ \gamma)(F)$, where $\beta = \neg D$ is the truth assignment falsifying the clause $D \in F$, which in particular gives $(\beta \circ \gamma)(D) = \top$. Since $\beta(D) = \bot$ we obtain $\top = (\beta \circ \gamma)(D) = \gamma(D)$. □

Inprocessing Our goal is to adjust and extend the abstract framework of [17] such that incremental SAT solving with inprocessing can be handled. The derivation performed by an inprocessing SAT solver is modelled as a sequence of abstract states. Each state consist of three components: a set of irredundant clauses φ that the solver aims to satisfy, a set of redundant clauses ρ that can be removed without changing the satisfiability of the formula under consideration and an ordered sequence of witness labelled clauses σ (that are actually just literal-clause pairs in [17]), to keep track of eliminated clauses.

To make the paper more self contained Fig. 1 lists the original rules of [17], together with the proposed RAT instantiation of side conditions $\natural$ and $\flat$. Rule Strengthen strengthens the irredundant set of clauses, by moving a clause from the redundant set into it, while rule Forget allows to eliminate a redundant clause from ρ. Rule Learn introduces a new clause C into the redundant set of clauses in case C has RAT w.r.t. $\varphi \wedge \rho$. Rule Weaken simplifies the irredundant set by eliminating a clause C from it if C has RAT on a literal l of C w.r.t. φ. The eliminated clause is moved to the end of the literal-clause pair sequence σ.

Model Reconstruction One challenge of using inprocessing is to guarantee that a satisfying assignment of the final formula can be transformed to a satisfying assignment of the original, non-processed formula. A sequence of witness

$$\frac{\varphi\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho \wedge C\,]\,\sigma}\ \natural \qquad \frac{\varphi\,[\,\rho \wedge C\,]\,\sigma}{\varphi\,[\,\rho\,]\,\sigma} \qquad \frac{\varphi\,[\,\rho \wedge C\,]\,\sigma}{\varphi \wedge C\,[\,\rho\,]\,\sigma} \qquad \frac{\varphi \wedge C\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho \wedge C\,]\,\sigma \cdot (l : C)}\ \flat$$

Learn Forget Strengthen Weaken

where $\natural$ is "C has RAT w.r.t. $\varphi \wedge \rho$" and $\flat$ is "C has RAT on l w.r.t. φ".

Fig. 1. Instantiated (with RAT) inprocessing rules as introduced in [17]

labelled clauses σ is used as part of the abstract state to keep track of clauses eliminated by WEAKEN during inprocessing. The process of solution reconstruction described through pseudo code in [17] can be formalized as follows:

Definition 3 (Reconstruction Function). *Given a truth assignment τ and a sequence of witness labelled clauses σ, the reconstruction function is defined as*

$$\mathcal{R}(\tau, \varepsilon) = \tau, \qquad \mathcal{R}(\tau, \sigma \cdot (\omega : D)) = \begin{cases} \mathcal{R}(\tau, \sigma) & \text{if } \tau(D) = \top \\ \mathcal{R}((\tau \circ \omega), \sigma) & \text{otherwise.} \end{cases}$$

The reconstruction function takes a (partial) truth assignment τ and a sequence of witness labelled clauses σ as inputs. It traverses σ in reverse order and sets truth values of those literals in τ to true that are witnesses of not yet satisfied clauses in σ. We are now ready to formalize the central concept of this paper:

Definition 4 (Reconstruction Property). *A sequence of witness labelled clauses σ satisfies the reconstruction property w.r.t. a formula F iff for all truth assignments τ satisfying F, the result of the reconstruction function $\mathcal{R}$ on τ and σ is a satisfying assignment for $F \wedge \sigma$. An abstract state $\varphi \,[\, \rho \,]\, \sigma$ satisfies the reconstruction property iff σ satisfies the reconstruction property w.r.t. φ.*

For the expression $F \wedge \sigma$ in this definition we interpret σ as a set of its clauses.

3 Inprocessing Rules for Incremental Solving

Our first goal is to determine how information, such as learned clauses, can be transferred from one incremental solving phase to the next, utilizing that the sub-problem F^{i+1} is an extension of the previously solved sub-problem F^i. Thus, instead of solving F^{i+1} from scratch, previously learned facts are reused to avoid repeated work. This is sound if the incremental approach gives the same answer (satisfiable or unsatisfiable) as solving from scratch.

More formally, it is crucial that $\varphi^i_{k_i} \wedge \Delta_{i+1} \equiv_{sat} F^{i+1}$ holds, where $\varphi^i_{k_i}$ is the set of irredundant clauses at the end of the evaluation of F^i. We also need to make sure that $F^{i+1} \models \rho^i_{k_i}$, i.e. the redundant clause set at the end of the evaluation of F^i can be reused. Furthermore, we need to guarantee that a model for F^{i+1} can be resconstructed from any satisfying assignment of $\varphi^{i+1}_{k_{i+1}}$.

To establish notation and to emphasize what we would like to improve in this paper, we briefly describe how inprocessing in a non-incremental solver (as in e.g. [30] with cloning) would look like using only the original inprocessing rules of [17] (shown in Fig. 1). Each phase $i = 0, \ldots, n$ of solving an incremental problem $\mathcal{F}$ consists of a derivation of a formula $\varphi^i_{k_i} \wedge \rho^i_{k_i}$ as a sequence of states $\langle \varphi^i_0 \,[\, \rho^i_0 \,]\, \sigma^i_0, \ldots, \varphi^i_{k_i} \,[\, \rho^i_{k_i} \,]\, \sigma^i_{k_i} \rangle$, where (for all $j = 1, \ldots, k_i$)

(a) $\varphi^0_0 = F^0,\ \rho^0_0 = \emptyset,\ \sigma^0_0 = \varepsilon$

(b) $\varphi^i_{j-1} \,[\, \rho^i_{j-1} \,]\, \sigma^i_{j-1}$ results in $\varphi^i_j \,[\, \rho^i_j \,]\, \sigma^i_j$ as application of a rule in Fig. 1

(c) $\varphi^{i+1}_0 = F^{i+1},\ \rho^{i+1}_0 = \emptyset,\ \sigma^{i+1}_0 = \varepsilon.$

The *initial state* defined in (a) starts the derivation with F^0 as irredundant set of clauses, with an empty σ and without any redundant clause. Then following (b) the solver applies the rules of Fig. 1 until it reaches a state $\varphi^i_{k_i}\ [\,\rho^i_{k_i}\,]\ \sigma^i_{k_i}$ in which satisfiability of $\varphi^i_{k_i} \wedge \rho^i_{k_i}$ is determined. The new phase starts by adding a set of clauses to the problem, as described by (c). Such a derivation only relies on the original rules of [17], so each phase has to restart with completely empty ρ and σ and no information learned from solving F^i can be reused to solve F^{i+1}.

To capture inprocessing in an incremental solver we have to extend and modify the calculus of [17] (in Fig. 1). The initial state in (a) and the components of abstract states remain the same as in [17] (see Sect. 2), except that σ is more general. In our new calculus it consists of witness labelled clauses instead of literal-clause pairs, which allows to capture any redundancy property (not just RAT). We will refer on that component of a state as *reconstruction stack*.

Next sections describe the derivations of $\varphi^i_{j+1}\ [\,\rho^i_{j+1}\,]\ \sigma^i_{j+1}$ from $\varphi^i_j\ [\,\rho^i_j\,]\ \sigma^i_j$ for each $0 \leq j < k_i$ in each phase $i = 0, \ldots, n$ and show a sound way to start a new phase $i+1$ from state $\varphi^i_{k_i}\ [\,\rho^i_{k_i}\,]\ \sigma^i_{k_i}$ when adding Δ_{i+1} to $\varphi^i_{k_i}$.

3.1 Constrained Learning

The side condition $\sharp$ of rule Learn in Fig. 1 allows to learn clauses that remove models of the current formula. However, as the following example demonstrates, this is not correct in the context of incremental solving.

Example 2. Consider the incremental SAT problem $\mathcal{F} = \langle\{(a \vee b)\}, \{a, b\}\rangle$. First in phase $i = 0$ the evaluation of F^0 starts from the initial state $(a \vee b)\ [\,\emptyset\,]\ \varepsilon$. Now the clause $(\neg a \vee \neg b)$ can be learned since it has the RAT property w.r.t. $(a \vee b)$ (this is $\sharp$ in Fig. 1). Then, rule Strengthen can be applied on $(\neg a \vee \neg b)$ which yields state $(a \vee b) \wedge (\neg a \vee \neg b)\ [\,\emptyset\,]\ \varepsilon$, with a satisfiable set of irredundant clauses. In the next phase $i = 1$ we add Δ_1 and target to solve the formula $F^1 = (a \vee b) \wedge a \wedge b$, which still is satisfiable. However, conjoining Δ_1 to the irredundant clause set of the last state of the previous phase leads to the state $(a \vee b) \wedge (\neg a \vee \neg b) \wedge a \wedge b\ [\,\emptyset\,]\ \varepsilon$ with an unsatisfiable irredundant clause set.

Thus in our calculus the precondition of learning (Learn$^-$) is $\varphi \wedge \rho \models C$, i.e. we allow to learn only *implied* clauses. Compared to [17] our new rule Learn$^-$ is weaker due to this stronger side condition. It still covers most learning techniques in current SAT solvers, except forms of extended resolution such as blocked clause addition [17,22,23,24]. Learned clauses can be forgotten (Forget) or moved to the irredundant formula (Strengthen) as in [17].

3.2 Stronger Weakenings

We decompose the original weakening rule (Weaken in Fig. 1) of [17] into two rules: Weaken$^+$, as the name suggests, weakens the current formula by eliminating a clause C from the irredundant set while pushing it to the reconstruction stack. The Drop rule allows to weaken the current formula by eliminating an

$$\frac{\varphi \wedge C \,[\,\rho\,]\, \sigma}{\varphi \,[\,\rho\,]\, \sigma \cdot (\omega : C)}\ \text{ƀ} \qquad \text{WEAKEN}^+$$

$$\frac{\varphi \wedge C \,[\,\rho\,]\, \sigma}{\varphi \,[\,\rho\,]\, \sigma}\ \text{ø} \qquad \text{DROP}$$

where ƀ is $\varphi \wedge C \equiv^{\omega}_{sat} \varphi$ and ø is $\varphi \models C$

Fig. 2. New weakening and dropping rules

implied clause from the irredundant set. Removal of implied clauses from φ does not introduce (nor remove) models and so it is not necessary to save these clauses on the reconstruction stack. In our implementation the DROP rule is also used for more advanced equivalence-literal reasoning techniques [13,31,32]. Further, in current implementations weakening is always immediately followed by a forget step (simulating WEAKEN$^+$).

3.3 Incremental Clause Addition

The main feature of incremental SAT solving is the possibility to extend the previously solved formula with a set of new clauses. In non-incremental SAT solving, clauses determined to be redundant, always remain redundant. In incremental SAT solving arbitrary clauses can be added and thus previous simplifications might need to be reconsidered and potentially reversed.

Example 3. Consider the incremental SAT problem $\mathcal{F} = \langle \{F^0\}, \{(\neg a \vee b)\} \rangle$, where $F^0 = (a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b)$ and $F^1 = F^0 \wedge (\neg a \vee b)$. Phase $i = 0$ starts from the state $(a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b)\ [\,\emptyset\,]\ \varepsilon$. Resolving the first clause on a always produces tautological resolvents (i.e. it is blocked [16]). Thus WEAKEN$^+$ can be applied with witness a. Afterwards no other irredundant clause contains literal b and so both remaining irredundant clauses are blocked on $\neg b$. Thus they can be eliminated by WEAKEN$^+$ too, which results in state $\emptyset\ [\,\emptyset\,]\ (a : (a \vee b)) \cdot (\neg b : (\neg a \vee \neg b)) \cdot (\neg b : (a \vee \neg b))$, without irredundant clauses left, and the solver concludes F^0 to be satisfiable. Adding the new clause $(\neg a \vee b)$ to incrementally solve F^1 yields a state with a satisfiable set of irredundant clauses. But F^1 is actually unsatisfiable, so just adding $(\neg a \vee b)$ is not sound.

There are different ways to avoid unsoundness. An obvious way is to simply disallow simplifications over variables (or actually literals in our calculus) that might occur in later phases. In essence, this is the solution implemented through freezing in current SAT solvers [7], which ensures that the reconstruction stack does not contain frozen variables as witnesses. These frozen variables are then the only variables of the current formula that are allowed to reoccur in new clauses. We capture this property as follows.

Definition 5 (Clean Clause). *A clause C is clean w.r.t. a sequence of witness labelled clauses σ iff for all $(\omega : D) \in \sigma$ we have that $\neg C \cap \omega = \emptyset$.*

Example 4. The clause $(a \vee b)$ is not clean w.r.t. $(\neg b : (\neg a \vee \neg b)) \cdot (\neg b : (a \vee \neg b))$ because $\neg(a \vee b) \cap (\neg b) \neq \emptyset$. On the other hand, $(\neg a \vee \neg b)$ is clean w.r.t. the witness labelled clause sequence $(\neg b : (a \vee \neg b))$ since $\neg(\neg a \vee \neg b) \cap (\neg b) = \emptyset$.

$$\frac{\varphi\ [\rho]\ \sigma}{\varphi \wedge \Delta\ [\rho]\ \sigma}\ \boxed{\mathcal{I}}$$

AddClauses

where $\boxed{\mathcal{I}}$ is that each clause of Δ is clean w.r.t. σ

Fig. 3. New rule to capture clause set augmentation

With this definition the freezing approach guarantees that every added clause is clean w.r.t. the reconstruction stack. Building on that observation, we can now introduce clause addition (AddClauses in Fig. 3), where the side condition requires that each new clause in Δ is clean w.r.t. the reconstruction stack σ. If the added clauses are clean w.r.t. the reconstruction stack, then every assignment satisfying them will remain satisfying after applying the reconstruction function:

Lemma 1. *If a clause C is clean w.r.t. a sequence of witness labelled clauses σ, then for all truth assignments τ with $\tau(C) = \top$ we have that $\mathcal{R}(\tau, \sigma)(C) = \top$.*

Proof. By induction on the length of σ. The base case $\sigma = \varepsilon$ is trivial. Now consider $\sigma \cdot (\omega : D)$ and $\tau' = \tau$ if $\tau(D) = \top$, $\tau' = \tau \circ \omega$ otherwise. If $\tau(D) = \top$, then $\tau'(C) = \tau(C) = \top$. For $\tau(D) \neq \top$ there is $\ell \in C$ with $\tau(\ell) = \top$. As C is clean w.r.t. $(\omega : D)$, i.e., $\neg C \cap \omega = \emptyset$, we have $\neg\ell \notin \omega$ and so $\tau'(\ell) = (\tau \circ \omega)(\ell) = \tau(\ell) = \top$. This also holds if $\ell \in \omega$, since then $\omega(\ell) = \top$. Now it follows by induction applied to τ' and σ: $\top = \mathcal{R}(\tau', \sigma)(C) = \mathcal{R}(\tau, \sigma \cdot (\omega : D))(C)$. □

Thus, as long as all our clause elimination steps are based on witnesses that never occur in new clauses, we can add clauses without any problem in new incremental calls. However, this approach requires to know in advance in every phase i every literal of every Δ_j with $j > i$. Beyond that, it allows less clauses to be eliminated. Fortunately we can do better.

Instead of prohibiting simplifications, we allow arbitrary inprocessing as in a non-incremental SAT solver, but later reverse simplifications inconsistent with new clauses. It would be easy to just reverse all simplifications by reintroducing all eliminated clauses, but this is costly (as our experiments show). Therefore, it would be desirable to reverse a minimal subset of simplifications, but such a minimal set is in general difficult to identify.

As compromise we try to cheaply identify a sufficient subset of problematic simplifications as follows. If a new clause is not clean w.r.t. the reconstruction stack, we reverse those simplifications which have a negated literal of the new clause in the witness. Reversing all these problematic steps yields a clean reconstruction stack for all new clauses that in turn allows to apply rule AddClauses.

3.4 Reversing Weakening

The side condition of rule AddClauses identifies which simplifications need to be reversed in order to add a set of new clauses to the formula. What is missing is a rule to actually reverse these steps. The challenge with reversing

$$\frac{\varphi\ [\,\rho\,]\ \sigma \cdot (\omega : C) \cdot \sigma'}{\varphi \wedge C\ [\,\rho\,]\ \sigma \cdot \sigma'}\ \boxed{\partial}$$

RESTORE

where $\boxed{\partial}$ is "C is clean w.r.t. σ'"

Fig. 4. New rule to reverse a weakening step

clause eliminations is that many simplification steps are dependent on each other, e.g., in F^0 of Ex. 3 the last two clauses became blocked only after the first simplification step. Therefore one can not just arbitrarily reverse simplifications:

Example 5. Consider again the inprocessing of F^0 in Example 3, with the final state $\emptyset\ [\,\emptyset\,]\ (a : (a \vee b)) \cdot (\neg b : (\neg a \vee \neg b)) \cdot (\neg b : (a \vee \neg b))$. Assume we reverse the first simplification step, i.e. we move $(a \vee b)$ from the reconstruction stack to the irredundant clauses. The truth assignment $\tau = \{\neg a, b\}$ would satisfy the irredundant clauses of the resulting state $(a \vee b)\ [\,\emptyset\,]\ (\neg b : (\neg a \vee \neg b)) \cdot (\neg b : (a \vee \neg b))$. The reconstruction function on that assignment and the current stack would be $\mathcal{R}(\tau, (\neg b : (\neg a \vee \neg b)) \cdot (\neg b : (a \vee \neg b)))$. Since $\tau(a \vee \neg b) \neq \top$, it first updates τ with the witness of that clause and becomes $\tau' = (\tau \circ \{\neg b\}) = \{\neg a, \neg b\}$. Then, τ' satisfies the next clause of the stack and so $\mathcal{R}(\tau', (\neg b : (\neg a \vee \neg b))) = \mathcal{R}(\tau', \varepsilon) = \tau'$. However, $\tau'(a \vee b) = \bot$. Thus, reversing only the first simplification step led to a state where we failed to reconstruct a solution for F^0.

Our main contribution is the rule RESTORE in Fig. 4 which provides a sound way to reintroduce *selected* clauses from the stack back to the formula using the concept of clean clauses of Def. 5 as precondition.

Example 6. Consider again formula F^0 of Example 3. Example 4 shows that the first clause of the stack is not clean w.r.t. its suffix $((a \vee b)$ w.r.t. $(\neg b : (\neg a \vee \neg b)) \cdot (\neg b : (a \vee \neg b)))$, but the second and third clauses are both clean $((\neg a \vee \neg b)$ w.r.t. $(\neg b : (a \vee \neg b))$ and $(a \vee \neg b)$ w.r.t. $\varepsilon)$. Restoring the second clause leads to the state $(\neg a \vee \neg b)\ [\,\emptyset\,]\ (a : (a \vee b)) \cdot (\neg b : (a \vee \neg b))$. A satisfying assignment of $(\neg a \vee \neg b)$ is $\tau = \{\neg a, \neg b\}$. The reconstruction function on τ and the current stack would be then $\mathcal{R}(\tau, (a : (a \vee b)) \cdot (\neg b : (a \vee \neg b))) = \mathcal{R}(\tau, (a : (a \vee b)))$, since $\tau(a \vee \neg b) = \top$. Because $\tau(a \vee b) \neq \top$, τ needs to be updated with the witness a, $\tau' = \tau \circ \{a\} = \{a, \neg b\}$. Then $\mathcal{R}(\tau, (a : (a \vee b))) = \mathcal{R}(\tau', \varepsilon) = \tau'$. The resulting assignment τ' satisfies not just the irredundant formula but each clause of the stack as well. Similarly, starting from any other satisfying assignment of $(\neg a \vee \neg b)$, the result of the reconstruction function satisfies all clauses.

3.5 Incremental Inprocessing Rules

The final and complete version of our calculus is shown in Fig. 5. To keep the notation simple the precise indexing of the states were so far omitted. Following the convention introduced at the beginning of this section, each single-line rule allows to derive a state $\varphi^i_{j+1}\ [\,\rho^i_{j+1}\,]\ \sigma^i_{j+1}$ from a state $\varphi^i_j\ [\,\rho^i_j\,]\ \sigma^i_j$, with $0 \leq i \leq n$ and $0 \leq j < k_i$, while our double-line rule ADDCLAUSES transits from a state $\varphi^i_{k_i}\ [\,\rho^i_{k_i}\,]\ \sigma^i_{k_i}$ to state $\varphi^{i+1}_0\ [\,\rho^{i+1}_0\,]\ \sigma^{i+1}_0$.

$$\frac{\varphi\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho \wedge C\,]\,\sigma}\;\boxed{\sharp}$$ LEARN$^-$

$$\frac{\varphi\,[\,\rho \wedge C\,]\,\sigma}{\varphi\,[\,\rho\,]\,\sigma}$$ FORGET

$$\frac{\varphi\,[\,\rho \wedge C\,]\,\sigma}{\varphi \wedge C\,[\,\rho\,]\,\sigma}$$ STRENGTHEN

$$\frac{\varphi\,[\,\rho\,]\,\sigma}{\varphi \wedge \Delta\,[\,\rho\,]\,\sigma}\;\boxed{\mathcal{I}}$$ ADDCLAUSES

$$\frac{\varphi \wedge C\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho\,]\,\sigma \cdot (\omega : C)}\;\boxed{\flat}$$ WEAKEN$^+$

$$\frac{\varphi \wedge C\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho\,]\,\sigma}\;\boxed{\varnothing}$$ DROP

$$\frac{\varphi\,[\,\rho\,]\,\sigma \cdot (\omega : C) \cdot \sigma'}{\varphi \wedge C\,[\,\rho\,]\,\sigma \cdot \sigma'}\;\boxed{\partial}$$ RESTORE

where $\boxed{\sharp}$ is $\varphi \wedge \rho \models C$, $\boxed{\flat}$ is $\varphi \wedge C \equiv^{\omega}_{sat} \varphi$, $\boxed{\varnothing}$ is $\varphi \models C$,
$\boxed{\partial}$ is C is clean w.r.t. σ' and $\boxed{\mathcal{I}}$ is that each clause in Δ is clean w.r.t. σ

Fig. 5. Incremental inprocessing rules

4 Formal Correctness

First we show that learned clauses are still valid in the next phase, and then prove that solutions can be reconstructed in each satisfiable state. In these proofs the set of irredundant clauses are always considered in *combination* together with the clauses on the reconstruction stack, i.e., $\varphi^i_j \wedge \sigma^i_j$. An important finding of our paper is that these combined formulas always imply the redundant clauses.

Proposition 2. *In any derivation in our calculus starting from the initial state the property $\varphi^i_j \wedge \sigma^i_j \models \rho^i_j$ holds for each phase $i = 0 \ldots n$ and j with $0 \leq j \leq k_i$.*

Proof. In the initial state $\varphi^0_0 \wedge \sigma^0_0 \models \rho^0_0$ trivially holds because ρ^0_0 is empty. Assume that $\varphi^i_j \wedge \sigma^i_j \models \rho^i_j$ holds (for any i and j s.t. $0 \leq i \leq n$ and $0 \leq j < k_i$). We show that any transition maintains the property. In case rule FORGET or STRENGTHEN is applied, ρ^i_{j+1} is weaker than ρ^i_j. In case of FORGET, $\varphi^i_{j+1} = \varphi^i_j$ and $\sigma^i_{j+1} = \sigma^i_j$, while in case of STRENGTHEN φ^i_{j+1} is even stronger than φ^i_j, and thus $\varphi^i_{j+1} \wedge \sigma^i_{j+1} \models \rho^i_{j+1}$ trivially follows in both cases. Rules WEAKEN$^+$ and RESTORE only move a clause between φ^i_j and σ^i_j and so $\varphi^i_j \wedge \sigma^i_j$ remains unchanged. Due to $\boxed{\varnothing}$, in case of DROP, $\varphi^i_j \equiv \varphi^i_{j+1}$, and so it also trivially maintains the property. When LEARN$^-$ transits from state j to $j+1$, we get from the inductive assumption that $\varphi^i_j \wedge \sigma^i_j \models \varphi^i_j \wedge \rho^i_j$ and due to $\boxed{\sharp}$, we know that $\varphi^i_j \wedge \rho^i_j \models C$, and so $\varphi^i_j \wedge \sigma^i_j \models \rho^i_j \wedge C = \rho^i_{j+1}$. When starting a new phase (i.e. moving from i to $i+1$ where $0 \leq i < n$ and j is k_i) only new clauses are added to $\varphi^i_{k_i}$ by ADDCLAUSES, and so $\varphi^{i+1}_0 \wedge \sigma^{i+1}_0 \models \rho^{i+1}_0$ clearly holds. □

With this proposition we can now prove that the combined formulas remain logically equivalent during a derivation, unless new clauses are added.

Proposition 3. *In any derivation starting from the initial state, the property $\varphi^i_j \wedge \sigma^i_j \equiv \varphi^i_{j+1} \wedge \sigma^i_{j+1}$ holds for phase $i = 0 \ldots n$ and each j with $0 \leq j < k_i$.*

Proof. Only the rules STRENGTHEN and DROP change the combined formula. However, STRENGTHEN strengthens with an implied clause (due to Prop. 2), while DROP guarantees logical equivalence due to its side condition. □

From that follows that at any point of a derivation within one phase the combined formula is logically equivalent to the incremental sub-problem:

Corollary 1. $F^i \equiv \varphi_0^i \wedge \sigma_0^i \equiv \varphi_1^i \wedge \sigma_1^i \equiv \cdots \equiv \varphi_{k_i}^i \wedge \sigma_{k_i}^i$.

Proof. $F^0 \equiv \varphi_0^0 \wedge \varepsilon \equiv \cdots \equiv \varphi_{k_0}^0 \wedge \sigma_{k_0}^0$. By an inductive argument and Prop. 3: $F^{i+1} = F^i \wedge \Delta_{i+1} \equiv \varphi_{k_i}^i \wedge \sigma_{k_i}^i \wedge \Delta_{i+1} = \varphi_0^{i+1} \wedge \sigma_0^{i+1} \equiv \cdots \equiv \varphi_{k_{i+1}}^{i+1} \wedge \sigma_{k_{i+1}}^{i+1}$. □

Moreover, an important practical consequence of Cor. 1 and Prop. 2 is that it is sound to keep the learned clauses of the solver when new clauses are added:

Corollary 2. $F^{i+1} \models \rho_{k_i}^i$.

Before we can prove that we can reconstruct a model for the original incremental problem from a model of the current irredundant clauses using the reconstruction stack we need the following lemma.

Lemma 2. *For a given truth assignment τ and a sequence of witness labelled clauses $\sigma \cdot \sigma'$ we have $\mathcal{R}(\tau, \sigma \cdot \sigma') = \mathcal{R}(\mathcal{R}(\tau, \sigma'), \sigma)$.*

Proof. By induction over the length of σ'. The base case $\sigma' = \varepsilon$ is trivial. Now consider $\sigma' = \sigma'' \cdot (\omega : C)$ and let $\tau' = \tau$ if $\tau(C) = \top$, $\tau' = \tau \circ \omega$ otherwise. Since $\mathcal{R}(\tau, \sigma \cdot \sigma') = \mathcal{R}(\tau', \sigma \cdot \sigma'')$ and $\mathcal{R}(\tau, \sigma') = \mathcal{R}(\tau', \sigma'')$, $\mathcal{R}(\tau, \sigma \cdot \sigma') = \mathcal{R}(\mathcal{R}(\tau, \sigma'), \sigma)$ follows from the induction hypothesis applied to τ' and $\sigma \cdot \sigma''$. □

Theorem 1 (Reconstructiveness). *In any derivation starting from the initial state, every state satisfies the reconstruction property of Def. 4.*

Proof. In the initial state the reconstruction stack is empty, and so for any satisfying assignment τ of F^0, $\mathcal{R}(\tau, \varepsilon)(F^0) = \top$. To simplify notation, we first consider only a single phase i (with $0 \leq i \leq n$), and omit the superscript i. Assume that in a state j (where $0 \leq j < k_i$), the reconstruction property holds. Let τ be a truth assignment with $\tau(\varphi_j) = \top$. Then $\mathcal{R}(\tau, \sigma_j)(\varphi_j \wedge \sigma_j) = \top$ follows by induction. In case Learn$^-$ or Forget was applied to state j, we have $\varphi_{j+1} = \varphi_j$ and $\sigma_{j+1} = \sigma_j$, thus the reconstruction property remains true. Rule Strengthen moves a clause C from ρ_j to φ_{j+1} and so $\varphi_{j+1} = \varphi_j \wedge C$ and $\sigma_{j+1} = \sigma_j$. In case $\tau(\varphi_{j+1}) = \top$ we have $\mathcal{R}(\tau, \sigma_{j+1})(\varphi_j \wedge \sigma_{j+1}) = \mathcal{R}(\tau, \sigma_j)(\varphi_j \wedge \sigma_j) = \top$ by induction. Then Prop. 2 gives $\varphi_j \wedge \sigma_j \models C$, thus $\mathcal{R}(\tau, \sigma_{j+1})(\varphi_j \wedge C \wedge \sigma_{j+1}) = \top$. From the side condition of Drop we know that $\tau(\varphi_{j+1} \wedge C) = \top$ whenever $\tau(\varphi_{j+1}) = \top$, and thus $\mathcal{R}(\tau, \sigma_{j+1})(\varphi_{j+1} \wedge \sigma_{j+1}) = \top$ again by induction. When Weaken$^+$ is applied, a redundant clause C is removed from φ_j and pushed to σ_{j+1} (i.e. $\varphi_j = \varphi_{j+1} \wedge C$) witnessed by ω. Assume $\tau(\varphi_{j+1}) = \top$. We apply the induction hypothesis to the truth assignments τ and $(\tau \circ \omega)$ to get:

$$\tau(\varphi_{j+1} \wedge C) = \top \implies \mathcal{R}(\tau, \sigma_j)(\varphi_{j+1} \wedge C \wedge \sigma_j) = \top \tag{1}$$

$$(\tau \circ \omega)(\varphi_{j+1} \wedge C) = \top \implies \mathcal{R}((\tau \circ \omega), \sigma_j)(\varphi_{j+1} \wedge C \wedge \sigma_j) = \top. \tag{2}$$

If $\tau(C) = \top$, then $\mathcal{R}(\tau, \sigma_j \cdot (\omega : C))(\varphi_{j+1} \wedge C \wedge \sigma_j) = \top$ due to (1). Furthermore, assuming the side condition of Weaken$^+$, we know that $(\omega : C)$ is redundant

w.r.t. φ_{j+1}. If $\tau(C) \neq \top$, then $(\tau \circ \omega)(\varphi_{j+1} \wedge C) = \top$ using Prop. 1. And with (2) we also get $\mathcal{R}(\tau, \sigma_j \cdot (\omega : C))(\varphi_{j+1} \wedge C \wedge \sigma_j) = \top$ if $\tau(C) \neq \top$. When we restore a clause C by RESTORE, we know that if $\tau(\varphi_j \wedge C) = \top$ then $\tau(C) = \top$. Further, we know from the side condition of RESTORE that C is clean w.r.t. σ', and so with Lemma 1, we obtain $\mathcal{R}(\tau, \sigma')(C) = \top$. From that and from Lemma 2 it follows that $\mathcal{R}(\tau, \sigma \cdot (\omega : C) \cdot \sigma') = \mathcal{R}(\mathcal{R}(\tau, \sigma'), \sigma \cdot (\omega : C)) = \mathcal{R}(\mathcal{R}(\tau, \sigma'), \sigma) = \mathcal{R}(\tau, \sigma \cdot \sigma')$, where $\varphi_j \wedge \sigma \wedge C \wedge \sigma'$ evaluates to true due to the induction hypothesis. When a new phase starts (i.e. $0 \leq i < n$ and $j = k_i$) as Δ_{i+1} is added to $\varphi^i_{k_i}$ by ADDCLAUSES, each new clause is clean w.r.t. $\sigma^i_{k_i}$. Thus, due to Lemma 1, the reconstruction function does not destroy any satisfying assignment of Δ_{i+1}. □

Theorem 2 (Correctness). *In any derivation starting from the initial state, for each phase $i = 0 \ldots n$ we have $F^i \equiv_{sat} \varphi^i_j \wedge \rho^i_j$ for all j with $0 \leq j \leq k_i$.*

Proof. From Prop. 2 and Thm. 1 it follows, that φ^i_j is unsatisfiable if $\varphi^i_j \wedge \rho^i_j$ is unsatisfiable. In this case also F^i is unsatisfiable using Cor. 1. Otherwise, if $\varphi^i_j \wedge \rho^i_j$ is satisfiable, then F^i is satisfiable due to Thm. 1 and again Cor. 1. □

To summarize, our calculus fulfills all the desiderata listed at the beginning of Sect. 3: (*i*) we can reuse the gained information of previous iterations (including learned clauses), (*ii*) we can continue with incremental solving in a satisfiability preserving way, and (*iii*) the reconstruction property guarantees that we can get a solution to the original problem in case of satisfiability.

5 Implementation

Based on our new approach we added incremental inprocessing to the SAT solver CaDiCaL [33]. Rule WEAKEN$^+$ is defined in our calculus based on the most general redundancy property and so it allows to employ every clause elimination procedure implemented in CaDiCaL including variable elimination [14], vivification [34,35], equivalent-literal substitution [31,32], hyper-binary resolution [13], (self-)subsumption [14] and blocked clause elimination [16]. Combining DROP with WEAKEN$^+$ allows efficient equivalence literal substitution, since only two binary clauses have to be stored on the stack for each literal in a strongly connected component [31,32] instead of all clauses with that literal. Similarly, gate-based variable elimination [14] only requires to save gate clauses.

At the heart of our new calculus are the RESTORE and ADDCLAUSES rules. They allow to reverse problematic simplification steps and add new clauses. In practice, SAT solvers are used via an interface (e.g. IPASIR [2] in CaDiCaL) to add new clauses Δ and then asked to solve the extended formula $F \wedge \Delta$. Before solving $F \wedge \Delta$, our approach first performs a sequence of RESTORE steps in order to make each clause in Δ clean w.r.t. the reconstruction stack σ using the algorithm RestoreAddClauses in Fig. 6. Then the new and restored clauses are added to the irredundant clauses and a new incremental solving phase starts.

The algorithm in Fig. 6 presents a simple implementation that identifies a sufficient set of clauses to restore in order to make Δ clean. It follows the idea of

RestoreAddClauses (new clauses Δ, reconstruction stack σ)

1 $(\omega_1 : C_1) \cdots (\omega_n : C_n) := \sigma$
2 **for** i **from** 1 **to** n
3 **if** exists $\ell \in \omega_i$ where $\neg\ell$ occurs in Δ **then**
4 $\Delta := \Delta \cup C_i\,, \quad \sigma := \sigma \setminus (\omega_i : C_i)$
5 **return** $\langle \Delta, \sigma \rangle$

Fig. 6. Algorithm RestoreAddClauses to identify and restore all tainted clauses.

"taint-checking", commonly used to reason about information-flow (see e.g. [36]). First consider every clause that comes from the user as tainted, because it potentially leads to problems. Then check whether these tainted clauses (actually literals of these clauses) trigger any clause on the stack to be restored. In that case the literals of the restored clause become tainted as well and recursively might trigger further clauses. However, restored clauses only need to be clean w.r.t. the reconstruction stack after them (see Restore in Fig. 4), while the clauses in Δ need to be clean w.r.t. the whole reconstruction stack. Therefore, the need for restoring is checked by traversing the stack from bottom to top (left to right). If a clause has to be restored, it can only trigger to restore clauses to its right. Thus, already processed clauses on the left do not have to be reconsidered.

The method takes the new clauses Δ and the current stack σ as input and checks each previous simplification step from left to right (see Line 1-2). Whenever the witness of a simplification has a literal that occurs negated in Δ, the simplification is reversed by restoring the eliminated clause from the stack. The check in Line 3 is actually asking whether there is a clause in Δ (i.e. in the set of new or already restored clauses) that is not clean w.r.t. $(\omega_i : C_i)$. To implement this check efficiently, we mark literals in Δ as tainted and in σ as witness. If the check succeeds, we need to restore the problematic C_i so that at the end we have a clean stack. In Line 4 the restored clause is added to Δ and removed from the stack. At the end of the procedure, Δ contains all the new and restored clauses, which added to the formula together with the new σ achieves the same effect as applying a sequence of Restore steps and a final AddClauses.

6 Experiments

We implemented a new bounded model checker called CaMiCaL for AIGER models [37], as used in the hardware model checking competition (HWMCC) [38]. Unrolling is simulated symbolically through substitution [39] in combination with structural hashing [40,41] and local low-level AIG optimizations [42]. As back-end different configurations of our SAT solver CaDiCaL [33] and other state-of-the-art incremental SAT solvers are used. The model checker was run on all the 300 models of the single safety property track of HWMCC'17 [38] up to bound 1000 with a time limit of 3600 seconds (for each model) and memory limit of 8 GB on our cluster with Intel Xeon E5-2620 v4 @ 2.10GHz CPUs.

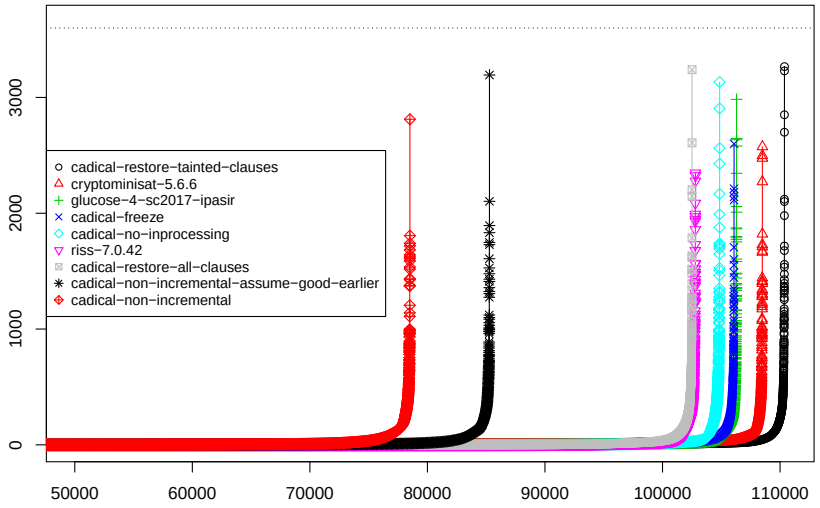

Fig. 7. Experimental results on all the 300 instances of the single safety property track of HWMCC'17. The x-axis corresponds to all bounds solved over all models sorted by the time needed for the SAT call for each bound, which is on the y-axis. The dotted horizontal line at 3600 second shows the time limit for solving all bounds of each model.

Results are presented in a similar way as the well-known cactus plots of the SAT Competition, except that we do not measure the overall running time of the model checker, but the time needed for one (incremental) call to the SAT solver. Figure 7 shows the distribution of these solving times. For example, if the model checker finished proving unsatisfiability for bound 41 after 110 seconds and then proved unsatisfiability for the consecutive bound 42 at 125 seconds then the time difference of 15 seconds is accounted for bound 42 on this instance. At the end each instance contributes as many solving times as bounds for it are solved.

As expected, worst performance is observed when the SAT solver is used in a completely non-incremental way (cadical-non-incremental), even with pre- and inprocessing enabled. It improves, if the model checker is allowed to assume earlier bounds to be good (cadical-non-incremental-assume-good-earlier). Incremental SAT solving is better as configuration cadical-restore-all-clauses shows, which employs pre- and inprocessing, but at the beginning of incremental calls restores all weakened clauses. However, disabling pre- and inprocessing completely during incremental SAT solving (cadical-no-inprocessing) is even better.

Configuration cadical-freeze can use variables for simplification which are not frozen. This again improves performance and there is no need to restore clauses. In bounded model checking (BMC) only variables encoding the next state are used in future calls and freezing them is sufficient. However, it required substantial programming effort to identify the set of frozen variables. Further, optimizations during CNF encoding, including structural hashing [40,41] across time frames or local two-level AIG optimizations [42], make it difficult to predict future use of variables. In other cases freezing might not even be possible [19].

Giving up on freezing makes use of our framework and gave the best solving times as configuration cadical-restore-tainted-clauses shows. This not only simplifies the way the solver is used through the API (no need to freeze variables) but also improves solving time. We measured the time spent in RestoreAddClauses to be less than 1% of the overall running time: 0.14% for our best configuration cadical-restore-tainted-clauses and 0.33% for cadical-restore-all-clauses. Our best configuration only restored 17% of the clauses. Restoring all clauses also lead to 3.4 times more eliminated clauses (applications of WEAKEN$^+$) in total.

Note that one can not get rid of freezing completely, since assumptions (for the "bad" state property in BMC) have to be frozen internally. Keeping freezing in the API might for instance also be useful for CNF simplification [8].

We also have similar results using freezing (as it is necessary for the solver Lingeling [30]) versus restoring tainted clauses for CaDiCaL as SAT solver back-end of our SMT solver Boolector [43]. We solved more benchmarks and decreased solving time significantly with the consequence that CaDiCaL is likely to replace Lingeling as incremental SAT solver back-end in the future.

We also considered other highly ranked SAT solvers in incremental tracks of the SAT Competition [2,44,45]: Glucose 4 [46], CryptoMiniSAT 5.6.6 [45,47] and Riss 7.0.42 [48]. CryptoMiniSAT performs significantly better than the other two external solvers. It is the only external solver which performs inprocessing during solving, including distillation [49]. Even though CryptoMiniSAT implements the same solution as [19] for incremental bounded variable elimination (BVE), this feature cannot be enabled through the API, and is disabled in our experiments. According to Mate Soos (private communication) scheduling BVE efficiently for incremental SAT solving is difficult for CryptoMiniSAT. We simply schedule BVE in CaDiCaL in the same way as during stand-alone SAT solving, with a persistent schedule across incremental invocations. Note that CaDiCaL only tries to eliminate variables and clauses which are newly added (or restored).

Source code of CaDiCaL and CaMiCaL and experimental data related to Fig. 7 can be found at `http://fmv.jku.at/incrinpr` including additional plots.

7 Conclusion

This paper presents a calculus that extends the framework of [17] to capture incremental SAT solving. It uses the most general clause redundancy property and is able to simulate most simplifications implemented in state-of-the-art SAT solvers. Our proposed approach is simple, eases the burden of using SAT solvers, can be implemented efficiently, and also reduces solving time substantially. As future work we want to support techniques which remove models, such as blocked clause addition, and techniques for simplifying under assumptions.

Acknowledgments. This research has been supported by the Austrian Science Fund (FWF) under projects W1255-N23 and S11408-N23. We thank Mathias Preiner and Aina Niemetz for their help in experimenting with Boolector and Håkan Hjort for providing feedback on using an incremental version of CaDiCaL.

References

1. Audemard, G., Lagniez, J., Simon, L.: Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction. In Järvisalo, M., Gelder, A.V., eds.: SAT. Volume 7962 of LNCS., Springer (2013) 309–317
2. Balyo, T., Biere, A., Iser, M., Sinz, C.: SAT Race 2015. Artificial Intelligence **241** (2016) 45–65
3. Eén, N., Sörensson, N.: An extensible SAT-solver. In Giunchiglia, E., Tacchella, A., eds.: SAT. Volume 2919 of LNCS., Springer (2003) 502–518
4. Hooker, J.N.: Solving the incremental satisfiability problem. J. Log. Program. **15**(1&2) (1993) 177–186
5. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without bdds. In Cleaveland, R., ed.: Proc. of the 5th Int. Conf. on Tools and Algorithms for Construction and Analysis of Systems, TACAS '99. Volume 1579 of LNCS., Springer (1999) 193–207
6. Bradley, A.R.: SAT-based model checking without unrolling. In Jhala, R., Schmidt, D.A., eds.: Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings. Volume 6538 of LNCS., Springer (2011) 70–87
7. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electr. Notes Theor. Comput. Sci. **89**(4) (2003) 543–560
8. Kupferschmid, S., Lewis, M.D.T., Schubert, T., Becker, B.: Incremental preprocessing methods for use in BMC. Formal Methods in System Design **39**(2) (2011) 185–204
9. Gocht, S., Balyo, T.: Accelerating SAT based planning with incremental SAT solving. In Barbulescu, L., Frank, J., Mausam, Smith, S.F., eds.: Proc. of the 27th International Conference on Automated Planning and Scheduling, ICAPS 2017, AAAI Press (2017) 135–139
10. Martins, R., Joshi, S., Manquinho, V.M., Lynce, I.: On using incremental encodings in unsatisfiability-based MaxSAT solving. JSAT **9** (2014) 59–81
11. Nadel, A.: Boosting minimal unsatisfiable core extraction. In Bloem, R., Sharygina, N., eds.: Proc. of 10th Int. Conf. on Formal Methods in Computer-Aided Design, FMCAD 2010, IEEE (2010) 221–229
12. Sebastiani, R.: Lazy satisability modulo theories. JSAT **3**(3-4) (2007) 141–224
13. Bacchus, F., Winter, J.: Effective preprocessing with hyper-resolution and equality reduction. In Giunchiglia, E., Tacchella, A., eds.: Selected Revised Papers of the 6th Int. Conf. on Theory and Applications of Satisfiability Testing. Volume 2919 of LNCS., Springer (2003) 341–355
14. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In Bacchus, F., Walsh, T., eds.: Proc. of the 8th Int. Conf. on Theory and Applications of Satisfiability Testing. Volume 3569 of LNCS., Springer (2005) 61–75
15. Heule, M., Järvisalo, M., Biere, A.: Efficient CNF simplification based on binary implication graphs. In Sakallah, K.A., Simon, L., eds.: Proc. of the 14th Int. Conf. on Theory and Applications of Satisfiability Testing - SAT 2011. Volume 6695 of LNCS., Springer (2011) 201–215
16. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In Esparza, J., Majumdar, R., eds.: Proc. of the 16th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2010. Volume 6015 of LNCS., Springer (2010) 129–144

17. Järvisalo, M., Heule, M., Biere, A.: Inprocessing rules. In Gramlich, B., Miller, D., Sattler, U., eds.: Proc. of Automated Reasoning - 6th International Joint Conference, IJCAR 2012. Volume 7364 of LNCS., Springer (2012) 355–370
18. Järvisalo, M., Biere, A.: Reconstructing solutions after blocked clause elimination. In: SAT. Volume 6175 of LNCS., Springer (2010) 340–345
19. Nadel, A., Ryvchin, V., Strichman, O.: Preprocessing in incremental SAT. In Cimatti, A., Sebastiani, R., eds.: Proc. of the 15th Int. Conf. on Theory and Applications of Satisfiability Testing - SAT 2012. Volume 7317 of LNCS., Springer (2012) 256–269
20. Heule, M.J.H., Kiesl, B., Biere, A.: Short proofs without new variables. In de Moura, L., ed.: Proc. of the 26th International Conference on Automated Deduction, CADE 26. Volume 10395 of LNCS., Springer (2017) 130–147
21. Heule, M.J.H., Kiesl, B., Biere, A.: Strong extension-free proof systems. Journal of Automated Reasoning (Feb 2019) To be published.
22. Audemard, G., Katsirelos, G., Simon, L.: A restriction of extended resolution for clause learning sat solvers. In: Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010), AAAI Press (2010)
23. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics **96-97** (1999) 149–176
24. Manthey, N., Heule, M.J.H., Biere, A.: Automated reencoding of boolean formulas. In: Proc. of the 8th Int. Haifa Verification Conference (HVC 2012). Volume 7857 of LNCS., Heidelberg, Springer (2013)
25. Nadel, A., Ryvchin, V., Strichman, O.: Ultimately incremental SAT. In Sinz, C., Egly, U., eds.: Proc. of the 17th Int. Conf. on Theory and Applications of Satisfiability Testing - SAT 2014. Volume 8561 of LNCS., Springer (2014) 206–218
26. Nadel, A., Ryvchin, V.: Efficient SAT solving under assumptions. In Cimatti, A., Sebastiani, R., eds.: Proc. of the 15th Int. Conf. on Theory and Applications of Satisfiability Testing - SAT 2012. Volume 7317 of LNCS., Springer (2012) 242–255
27. Blanchette, J.C., Fleury, M., Lammich, P., Weidenbach, C.: A verified SAT solver framework with learn, forget, restart, and incrementality. J. Autom. Reasoning **61**(1-4) (2018) 333–365
28. Fazekas, K., Bacchus, F., Biere, A.: Implicit hitting set algorithms for maximum satisfiability modulo theories. In Galmiche, D., Schulz, S., Sebastiani, R., eds.: Proc. of Automated Reasoning - 9th International Joint Conference, IJCAR 2018. Volume 10900 of LNCS., Springer (2018) 134–151
29. Balyo, T., Fröhlich, A., Heule, M., Biere, A.: Everything you always wanted to know about blocked sets (but were afraid to ask). In Sinz, C., Egly, U., eds.: Proc. of the 17th Int. Conf. on Theory and Applications of Satisfiability Testing - SAT 2014. Volume 8561 of LNCS., Springer (2014) 317–332
30. Biere, A.: Yet another local search solver and Lingeling and friends entering the SAT competition 2014. In Balint, A., Belov, A., Heule, M., Järvisalo, M., eds.: SAT Competition 2014. Department of Computer Science Series of Publications B, University of Helsinki (2014) 39–40
31. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters **8**(3) (1979) 121–123
32. Brafman, R.I.: A simplifier for propositional formulas with many binary clauses. In Nebel, B., ed.: Proc. of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Morgan Kaufmann (2001) 515–522

33. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2018. In Heule, M., Järvisalo, M., Suda, M., eds.: Proc. of SAT Competition 2018 – Solver and Benchmark Descriptions. Volume B-2018-1 of Department of Computer Science Series of Publications B., University of Helsinki (2018) 13–14
34. Luo, M., Li, C., Xiao, F., Manyà, F., Lü, Z.: An effective learnt clause minimization approach for CDCL SAT solvers. In Sierra, C., ed.: Proc. of the 26th International Joint Conference on Artificial Intelligence, IJCAI 2017, ijcai.org (2017) 703–711
35. Piette, C., Hamadi, Y., Sais, L.: Vivifying propositional clausal formulae. In Ghallab, M., Spyropoulos, C.D., Fakotakis, N., Avouris, N.M., eds.: Proc. of the 18th European Conference on Artificial Intelligence, ECAI 2008. Volume 178 of Frontiers in Artificial Intelligence and Applications., IOS Press (2008) 525–529
36. Schwartz, E.J., Avgerinos, T., Brumley, D.: All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In: 31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA, IEEE Computer Society (2010) 317–331
37. Biere, A., Heljanko, K., Wieringa, S.: AIGER 1.9 and beyond. Technical report, FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria (2011)
38. Biere, A., van Dijk, T., Heljanko, K.: Hardware model checking competition 2017. In Stewart, D., Weissenbacher, G., eds.: Formal Methods in Computer Aided Design, FMCAD 2017, IEEE (2017) 9
39. Jussila, T., Biere, A.: Compressing BMC encodings with QBF. Electr. Notes Theor. Comput. Sci. **174**(3) (2007) 45–56
40. Heule, M., Järvisalo, M., Biere, A.: Revisiting hyper binary resolution. In Gomes, C.P., Sellmann, M., eds.: Proc. of the 10th Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2013. Volume 7874 of LNCS., Springer (2013) 77–93
41. Kuehlmann, A., Paruthi, V., Krohm, F., Ganai, M.K.: Robust boolean reasoning for equivalence checking and functional property verification. IEEE Trans. on CAD of Integrated Circuits and Systems **21**(12) (2002) 1377–1394
42. Brummayer, R., Biere, A.: Local two-level And-Inverter graph minimization without blowup. In: Proceedings of the 2nd Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2006). (2006)
43. Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2 , BtorMC and Boolector 3.0. In Chockler, H., Weissenbacher, G., eds.: Proc. of the 30th Int. Conf. on Computer Aided Verification, CAV 2018. Volume 10981 of LNCS., Springer (2018) 587–595
44. Balyo, T., Heule, M., Järvisalo, M., eds.: Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions. Volume B-2016-1 of Department of Computer Science Series of Publications B., University of Helsinki (2016)
45. Balyo, T., Heule, M., Järvisalo, M., eds.: Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions. Volume B-2017-1 of Department of Computer Science Series of Publications B., University of Helsinki (2017)
46. Audemard, G., Simon, L.: Glucose and Syrup in the SAT17. In Balyo, T., Heule, M., Järvisalo, M., eds.: Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions. Volume B-2017-1 of Department of Computer Science Series of Publications B., University of Helsinki (2017) 16–17
47. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In Kullmann, O., ed.: Proc. of the 12th Int. Conf. on Theory and Applications of Satisfiability Testing - SAT 2009. Volume 5584 of LNCS., Springer (2009) 244–257

48. Manthey, N.: Riss 7. In Balyo, T., Heule, M., Järvisalo, M., eds.: Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions. Volume B-2017-1 of Department of Computer Science Series of Publications B., University of Helsinki (2017) 29
49. Han, H., Somenzi, F.: Alembic: An efficient algorithm for CNF preprocessing. In: Proc. of the 44th Design Automation Conference, DAC 2007, IEEE (2007) 582–587