Incremental Inprocessing Rules beyond Resolution

Katalin Fazekas[1], Florian Pollitt[2], Mathias Fleury[2] and Armin Biere[2]

[1]TU Wien, Austria
[2]University of Freiburg, Germany

Abstract

Formula simplification in the form of pre- and inprocessing is a crucial part of modern SAT solvers. While most inprocessing techniques focus on eliminating redundant clauses, significant performance improvements have recently been achieved with clause addition techniques. However, not all inprocessing techniques can be employed in incremental use cases. In particular, combining incremental reasoning with clause addition techniques based on more general redundancy properties in a sound and efficient way is an open challenge. In this paper, we extend the incremental inprocessing calculus for SAT solvers to facilitate some of these clause addition techniques and reason about the soundness and completeness of such systems. The resulting framework formally defines sufficient conditions for efficiently implementing in incremental SAT solvers such beyond resolution techniques, including Bounded Variable Addition (BVA) and certain Blocked Clause Addition (BCA) steps.

Keywords

Incremental SAT, Inprocessing, Bounded Variable Addition

1. Introduction

Modern SAT solvers often implement a diverse repertoire of different *inprocessing techniques* in an effort to simplify the notoriously complex NP-complete decision problems they seek to solve. Most of these techniques focus on reducing the search space by eliminating certain *redundant* parts of the formula while preserving its satisfiability. Some notable examples include Blocked Clause Elimination (BCE) [1], Bounded Variable Elimination (BVE) [2], and Equivalent Literal Substitution (ELS) [3, 4, 5, 6]. In contrast to these techniques, other approaches focus on introducing redundant clauses and new variables in a way that preserves satisfiability, rather than eliminating them (see for example [7, 8, 9]). Such *clause addition techniques* gained renewed attention after SBVA-CaDiCaL [10] won the SAT Competition in 2023 [11]. This winner tool combined CaDiCaL, a modern state of the art SAT solver, with an efficient implementation of Bounded Variable Addition (BVA) [12] as a preprocessing technique. Beyond BVA, many other clause addition techniques, such as Blocked Clause Addition (BCA) [13] and reasoning over cardinality constraints [9] or decision diagrams [7], require a proof system based on some form of extended resolution [14, 15, 16] to certify. Nevertheless, an abstract framework that captures in a unified way and reasons about the usual inprocessing techniques of SAT solvers, including both clause addition and elimination techniques, was introduced by Järvisalo et al. in [17].

There are many use cases for SAT solvers, for example in model checking [18], or in sub-reasoning steps of MaxSAT [19] and Satisfiability modulo Theories (SMT) solvers [20, 21], where a sequence of similar SAT problems are solved rather than a single decision problem. Such use cases can greatly benefit from *incremental SAT solving* [22, 23] where the results of previous solver runs can be reused during reasoning, potentially avoiding much repeated work. However, in order to be able to reuse the previous solver steps, formula simplification in incremental SAT solvers requires additional care [24]. Nadel at al. [25] showed how to handle certain clause elimination techniques in incremental solvers. A refinement of the inprocessing rules for incremental use cases was introduced in [26], where all clause elimination techniques are supported, but only implied clauses can be learned. In this paper, we

PoS'25: Pragmatics of SAT, August 11, 2025, Glasgow, Scotland
katalin.fazekas@tuwien.ac.at (K. Fazekas); ipollittf@informatik.uni-freiburg.de (F. Pollitt); fleury@cs.uni-freiburg.de (M. Fleury); biere@cs.uni-freiburg.de (A. Biere)
0000-0002-0497-3059 (K. Fazekas); 0009-0001-4337-6919 (F. Pollitt); 0000-0002-1705-3083 (M. Fleury); 0000-0001-7170-9242 (A. Biere)

extend the incremental inprocessing rules of [26] so that clause addition techniques based on extended resolution [15, 16] are enabled. This extension allows to capture on an abstract level incremental inprocessing in the presence of a variety of clause addition techniques, including forms of BVA, BCA and extended resolution.

The paper is organized as follows: Section 2 provides an overview of the used notation and background, while Section 3 presents our new derivation rule that captures clause addition techniques based on more general redundancy properties and reasons about its correctness. In Section 4 we illustrate the use of the new calculus and Section 5 concludes the paper.

2. Preliminaries

Boolean Satisfiability We assume the standard semantics and syntax of propositional logic and its decision problem (SAT). Given a set of Boolean variables $\mathcal{V}$, a propositional formula F over $\mathcal{V}$ in conjunctive normal form is a conjunction of clauses, where each clause is a disjunction of literals and a literal is a Boolean variable or its negation. A (partial) truth assignment τ over a set of variables $\mathcal{V}$ is a set of non-contradicting literals over $\mathcal{V}$. An assignment τ satisfies (resp. falsifies) a literal l if $l \in \tau$ (resp. $\neg l \in \tau$). If neither $l \in \tau$ nor $\neg l \in \tau$, then the literal is unassigned and the truth assignment is partial. A clause is satisfied by a truth assignment τ if at least one of its literals is satisfied. A truth assignment satisfies a CNF formula F, noted as $\tau(F) = \top$, if it satisfies every clause of it. We use $\tau_1 \circ \tau_2$ to denote the composition of truth assignments τ_1 and τ_2, i.e., $(\tau_1 \circ \tau_2)\ (F) = \tau_1(\tau_2(F))$. Note that if $dom(\tau_1) \cap dom(\tau_2) = \emptyset$, then $(\tau_1 \circ \tau_2)\ (F) = \tau_1(\tau_2(F)) = \tau_2(\tau_1(F)) = (\tau_2 \circ \tau_1)\ (F)$.

Incremental SAT Instead of solving a single satisfiability problem, we are interested in solving a sequence of SAT formulae $\langle F^0, F^1, \ldots, F^n \rangle$ over a *global* variable set $\Gamma \subseteq \mathcal{V}$, where each formula extends the previous one with new clauses, i.e., starting from an initial SAT problem F^0, for each *phase* $i = 0 \ldots n-1$ we have that $F^{i+1} = F^i \wedge \Delta_{i+1}$ over the variable set Γ. As each phase extends the previous formula with further clauses, if F^i is unsatisfiable, then F^j for all $j > i$ is unsatisfiable as well. Incremental SAT solvers also support *assumptions*, literals temporarily assumed to be true in a given phase. In practice they are *frozen variables* [23, 24], thus we do not consider them explicitly in this paper.

Existentially Quantified SAT We further introduce $\Lambda \subseteq \mathcal{V}$, a (sufficiently large) set of *latent* Boolean variables, s.t. $\Gamma \cap \Lambda = \emptyset$. Given a SAT problem F over $\mathcal{V} = \Gamma \cup \Lambda$, we denote with $\exists\Lambda\ (F)$ the quantified Boolean formula (QBF) where all Λ-variables are existentially quantified, all Γ-variables are free and the formula matrix is F. Further, we assume every truth assignment to be a total function over Λ and Γ, unless stated otherwise. For an introduction of standard QBF syntax and semantics, see for example [27], while here we define explicitly the most relevant related concepts that are used throughout this paper.

Definition 1 (Projected satisfaction)**.** *Let τ be a truth assignment over Γ, we say that τ satisfies $\exists\Lambda\ (F)$ iff there exists a truth assignment λ over Λ, such that $(\tau \circ \lambda)\ (F) = \top$.*

Definition 2 (Projected implication)**.** *Given a formula F and a clause C, both over variables $\Gamma \cup \Lambda$, we say that F Λ-implies C, denoted as $F \models_\Lambda C$, iff all satisfying truth assignments (according to Def. 1 over Γ) of $\exists\Lambda\ (F)$ satisfy $\exists\Lambda\ (F \wedge C)$ as well.*

Definition 3 (Projected equivalence)**.** *Given formulas F and G over variables $\Gamma \cup \Lambda$, we say that F is logically equivalent to G under Λ, denoted as $F \equiv_\Lambda G$, iff exactly the same truth assignments over Γ satisfy $\exists\Lambda\ (F)$ and $\exists\Lambda\ (G)$.*

In other words, for all τ over Γ s.t. there exists λ over Λ with $(\tau \circ \lambda)\ (F) = \top$, it holds that there exists also a λ' over Λ s.t. $(\tau \circ \lambda')\ (G) = \top$. Note that λ and λ' can be different, but in certain cases they are the exact same truth assignment, as the following proposition shows.

Proposition 4. *Given a formula F and a clause C over $\Gamma \cup \Lambda$, if $F \models C$, then both $\models \exists \Lambda\ (F \rightarrow C)$ and $F \models_\Lambda C$ holds.*

The following proposition makes explicit the relation between projected equivalence and implication. Notice that projected equivalence and implication are both transitive properties, i.e., if $F \equiv_\Lambda G$ and $G \equiv_\Lambda H$, then $F \equiv_\Lambda H$ (resp. if $F \models_\Lambda G$ and $G \models_\Lambda H$, then $F \models_\Lambda H$).

Proposition 5. *Given a formula F and a clause C over $\Gamma \cup \Lambda$, $F \models_\Lambda C$ iff $F \equiv_\Lambda F \wedge C$.*

Though these definitions and propositions first may seem to be straightforward, they include several subtle details that were refined and exploited through a long line of previous research, see for example in [28, 29, 30, 31, 32, 33].

Incremental Inprocessing This paper extends the incremental inprocessing calculus that was introduced in [26] and later slightly refined in [34, Section 6.3]. To make the paper more self contained, here we sum up the most important terminology and notation with minor adjustments to our current concerns. Formula pre- and inprocessing steps are concerned about efficiently identifying *redundant clauses* (see Def. 7), that we define based on the most general redundancy property introduced in [35]. To make explicit the reason of found redundancies when it is necessary, we use the notion of *witness labelled clauses* (see Def. 6).

Definition 6 (Witness Labelled Clause [26])**.** *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 $(\omega : C)$.*

Definition 7 (Redundancy [26, 35])**.** *A witness labelled clause $(\omega : C)$ is redundant w.r.t. a formula F if $\omega(C) = \top$ and $F|_{\neg C} \models F|_{\neg C \circ \omega}$. This is also denoted as $F \wedge C \equiv^{\omega}_{sat} F$.*

As Def. 7 indicates, adding or removing redundant clauses is satisfiability preserving. Moreover, knowing the witness of removed redundant clauses allows to transform a solution of the simplified formula to a solution of the removed redundant clauses as well. Note that we use a stronger definition $F|_{\neg C} \models F|_{\neg C \circ \omega}$ (essentially the same as in [36]) instead of the original version $F|_{\neg C} \models F|_{\omega}$ [35] used in [26], as solution reconstruction otherwise requires larger witnesses, as the following example shows.

Example 8 (Total Models and Variable Elimination)**.** *Consider the clauses $(a \vee b) \wedge (\neg a \vee \neg b)$. Variable elimination on a would find that both clauses are redundant. However, using only a (resp. $\neg a$) as a witness does not satisfy the original redundancy criteria. One could always extend the witness to set all variables of the clause, which however "taints" more literals than necessary (*cf. *"clean condition" of* Restore *in Fig. 1).*

Example 8 was observed in a formalization of (non-incremental) reconstruction in Isabelle as part of a Master thesis [37]. It is not a counterexample to incremental model reconstruction [26] (proofs remain correct and transfer to the stronger version) but it shows that the original redundancy criteria is not strong enough to cover the implementation in CaDiCaL. Practically it means that solution reconstruction needs to work with total assignments and has to assign all variables initially and not on-the-fly during the reconstruction procedure.

Lemma 9 (Satisfying Redundant Clauses)**.** *Assume $F \equiv^{\omega}_{sat} F \wedge C$ and let τ be a total assignment over $\Lambda \cup \Gamma$ s.t. $\tau(F) = \top$ and $\tau(C) = \bot$. Then, $(\tau \circ \omega)\ (F \wedge C) = \top$.*

Proof Sketch. Very similar to the original proof in [26], but the stronger $\tau \circ \alpha = \tau$ (where $\alpha = \neg C$) holds and therefore α vanishes everywhere in the proof. □

Due to the refinement in Def. 7, hereafter we assume found models to be *total*. This change only makes proofs easier, because the assumption is stronger than in the original reconstruction. In practice, SAT solvers most often stop their search only once every variable was assigned without a conflict, thus this change does not have practical consequences.

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

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

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

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

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

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

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

where $\sharp$ is $\varphi \wedge \rho \models C$, $\flat$ is $\varphi \wedge C \equiv^{\omega}_{sat} \varphi$, $\varnothing$ is $\varphi \models C$, ∂ is C is clean w.r.t. σ' and $\mathcal{I}$ is that $\mathrm{vars}(\Delta) \subseteq \Gamma$ and each clause in Δ is clean w.r.t. σ.

Figure 1: Incremental inprocessing rules as introduced in [26].

We capture inprocessing steps in an incremental SAT solver as derivation steps in a formal calculus that operates on abstract states. An abstract state, denoted as $\varphi\,[\,\rho\,]\,\sigma$, consists of three parts: (i) φ: a set of irredundant clauses (ii) ρ: a set of learned redundant clauses and (iii) σ: an ordered sequence of witness labelled clauses representing the reconstruction stack of the solver where the previously removed redundant clauses are stored. Given a sequence of incremental SAT problems $\langle F^0, F^1, \ldots, F^n \rangle$ over a set of variables Γ, the derivation starts from the initial state $\emptyset\,[\,\emptyset\,]\,\varepsilon$ and the possible derivation rules, together with their corresponding side conditions defining when a rule is actually applicable, are shown in Fig. 1.

A clause can be removed from the irredundant clause database in two cases: (i) when it is implied by all the other clauses in φ, and in that case the clause can be completely deleted (see DROP), or (ii) when it is redundant w.r.t. φ (see WEAKEN$^+$), and in that case it is moved to the end of the reconstruction stack. Reintroducing a clause C from the reconstruction stack σ into φ via RESTORE requires to first restore all those clauses from the stack that could have not been removed from φ if C was present. To recognize a superset of such clauses, we use the notion of clean clauses:

Definition 10 (Clean Clauses [26]). *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 $\neg C \cap \omega = \emptyset$.*

Redundant clauses can be either deleted by FORGET or moved to the irredundant set by STRENGTHEN at any point of time. A new redundant clause C can be learned by LEARN$^-$ when it is implied by the formula $\varphi \wedge \rho$ (see $\sharp$). Note that though this step captures learning clauses derived by resolution during conflict analysis, it does not support clause addition techniques based on more general redundancies.

Rule ADDCLAUSES captures when users add new clauses to the problem (i.e., when a new solving phase starts). In that case, before applying ADDCLAUSES, all previously done relevant clause elimination steps must be undone (via RESTORE) relying again on the notion of clean clauses (see $\mathcal{I}$). Further, in contrast to the original calculus, here we make explicit the set of variables on what the formulas are defined (Γ). As we will discuss it later, this refinement of $\mathcal{I}$ simplifies our reasoning but does not necessarily affect user experience. That is, it is not required to know ahead of time which variables will reoccur in clauses that are added later. In the next sections we will refer to derivation of state $\varphi^i_{j+1}\,[\,\rho^i_{j+1}\,]\,\sigma^i_{j+1}$ 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 of steps where a new phase $i+1$ starts from $\varphi^i_{k_i}\,[\,\rho^i_{k_i}\,]\,\sigma^i_{k_i}$ by adding Δ_{i+1} to $\varphi^i_{k_i}$.

We use the following revised[1] version of reconstruction function definition that captures solution reconstruction proposed by Järvisalo et al. [38] of inprocessing SAT solvers.

Definition 11 (Reconstruction Function [34]). *Given a truth assignment τ and a witness labelled clause $(\omega : C)$ the reconstruction function is defined as*

$$\mathcal{R}_{(\omega:C)}(\tau) = \begin{cases} \tau & \text{if } \tau(C) = \top \\ (\tau \circ \omega) & \text{otherwise.} \end{cases}$$

[1] Christoph Scholl proposed this simplified version of the reconstruction function while working on [26].

$$\frac{\varphi\ [\,\rho\,]\ \sigma}{\varphi\ [\,\rho \wedge (\ell \vee C)\,]\ \sigma}\ \boxed{\Sigma}$$

EXTEND

where $\boxed{\Sigma}$ is that $\ell \in \Lambda$ (and so $\ell \notin \Gamma$) and $\varphi \wedge \rho \wedge \sigma \equiv^{\ell}_{sat} \varphi \wedge \rho \wedge \sigma \wedge (\ell \vee C)$ holds.

Figure 2: New transition to capture clause addition witnessed by latent literal ℓ.

The reconstruction function for a sequence of witness labelled clauses σ is defined as $\mathcal{R}_{\varepsilon} = \mathrm{id}$ and $\mathcal{R}_{\sigma} = \mathcal{R}_{(\omega_1:C_1)\cdot\ldots\cdot(\omega_n:C_n)} = \mathcal{R}_{(\omega_1:C_1)} \circ \cdots \circ \mathcal{R}_{(\omega_n:C_n)}$.

Notice that from this definition, due to the associativity of function composition, it immediately follows that for a given truth assignment τ and a sequence of witness labelled clauses $\sigma \cdot \sigma'$ we have that $\mathcal{R}_{\sigma\cdot\sigma'}(\tau) = \mathcal{R}_{\sigma}(\mathcal{R}_{\sigma'}(\tau))$, while in [26] it required an additional lemma.

The following lemma will help us to show the correctness of solution reconstruction in our extended calculus. For more details on the incremental inprocessing calculus and for a proof of this lemma, see [26, 34, Section 6.3].

Lemma 12 (Clean Reconstruction [26])**.** *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 $\mathcal{R}_{\sigma}(\tau)(C) = \top$.*

3. Incremental Inprocessing with Blocked Clause Addition

While the original (non-incremental) inprocessing calculus of [17] allows learning arbitrary redundant clauses (based on RAT or even more general redundancy properties), the incremental extension of it supports only implied clauses to be learned via LEARN^{-} (see Fig. 1).

In Fig. 2 we introduce a new possible step, called EXTEND, that enables learning clauses based on a more general redundancy property but only in very specific cases: (1) the redundancy has to hold w.r.t. every part of the formula, including those clauses that are currently on the reconstruction stack, and (2) the redundancy has to be witnessed by a single latent literal. Though the requirements of EXTEND are very specific, it captures several clause addition techniques, such as BVA, BCA and extended resolution (see Sect. 4 for some examples). Note that these techniques could be simulated without EXTEND, by simply adding the learned clauses as user input via ADDCLAUSES. However, then the added clauses are part of the irredundant clause database (making it harder to eliminate them) and the reconstruction stack has to be first cleaned with RESTORE steps (see $\boxed{\mathcal{I}}$). In contrast to that, EXTEND adds the new clauses as redundant (just as LEARN^{-}) and no RESTORE steps are necessary before applying it. Preprocessing steps of users (and wrapping systems) can continue using ADDCLAUSES, while EXTEND captures the internal inprocessing steps that are witnessed by the internal latent variables. That is, latent variables are private and only help the reasoning of the SAT solver, while users can only use and reuse any of the Γ-variables. In practice, when users need a fresh variable, it has to be made sure that it is not used already as a latent variable internally in the solver.

Now we show that it is correct to combine EXTEND with the other steps of an incremental solver (see Fig. 1), as long as the latent variables remain unused by the user (see $\boxed{\mathcal{I}}$). First, see that learning by EXTEND is satisfiability preserving and their solution is reconstructible.

Lemma 13. *Assume $F \equiv^{x}_{sat} F \wedge (x \vee C)$ and let τ be a total assignment over $\Gamma \cup \Lambda$ s.t. $\tau(F) = \top$ and $\tau(x \vee C) = \bot$. Then, $(\tau \circ x)\ (F \wedge (x \vee C)) = \top$.*

Proof. Follows from Lemma 9 (note that τ is assumed to be total, as discussed at Example 8). □

Allowing redundant but not implied clauses to be learned means that the redundant clauses are not logical consequences of the irredundant and eliminated ones any more. However, as the following lemmas show, a projected implication still holds in the new calculus.

Lemma 14. *If* $F \equiv^{x}_{sat} F \wedge (x \vee C)$ *where* $x \in \Lambda$*, then* $F \models_\Lambda (x \vee C)$.

Proof. Assume $\tau = (\gamma \circ \lambda)$ is a total assignment over $\Gamma \cup \Lambda$ s.t. $\tau(F) = \top$. Then either also $(\gamma \circ \lambda)\ (F \wedge (x \vee C)) = \top$ holds, or by Lemma 13, we know that $(\gamma \circ (\lambda \circ x))\ (F \wedge (x \vee C)) = \top$. Thus, whenever there exists a truth assignment λ over Λ s.t. $(\gamma \circ \lambda)\ (F) = \top$, there exists a truth assignment λ' s.t. $(\gamma \circ \lambda')\ (F \wedge (x \vee C)) = \top$ by defining λ' either as λ or as $(\lambda \circ x)$. □

Lemma 15. *Starting from the initial state, in any reachable state* $\varphi\ [\rho]\ \sigma$*, the property* $\varphi \wedge \sigma \models_\Lambda \varphi \wedge \sigma \wedge \rho$ *holds.*

Proof. We prove it by induction on the length of the derivation. In the initial state the property trivially holds ($\varphi^0_0 = \emptyset$ without Λ variables and both σ^0_0 and ρ^0_0 are empty). Assuming that $\varphi \wedge \sigma \models_\Lambda \rho$ holds, we show the induction step by a case distinction on the next applied rule. Rules Weaken$^+$ and Restore only move a clause between φ and σ, AddClauses only strengthens $\varphi \wedge \sigma$, while Forget only weakens ρ, thus in these cases the property is trivially maintained. With Drop a clause is removed from φ, but $\varphi \setminus \{C\} \equiv \varphi$ is ensured by ø. Rule Strengthen weakens ρ by moving a clause of it to φ, thereby also strengthens $\varphi \wedge \sigma$, thus the property is maintained. When a clause C is learned by Learn$^-$ from the inductive assumption we know that for any truth assignment γ over Γ if there exists a λ over Λ s.t. $(\gamma \circ \lambda)\ (\varphi \wedge \sigma) = \top$, then there exists a λ' over Λ s.t. $(\gamma \circ \lambda')\ (\varphi \wedge \sigma \wedge \rho) = \top$. Since $\varphi \wedge \sigma \wedge \rho \models C$ (due to ♯), we know that $(\gamma \circ \lambda')\ (\varphi \wedge \sigma \wedge \varphi \wedge C) = \top$ holds as well. When the new Extend rule is applied, one needs to show that $\varphi \wedge \sigma \models_\Lambda \varphi \wedge \sigma \wedge \rho \wedge (x \vee C)$. Let γ be a truth assignment over Γ s.t. there exists a λ over Λ s.t. $(\gamma \circ \lambda)\ (\varphi \wedge \sigma) = \top$. Then, from the ind. assumption follows that there also exists a truth assignment λ' s.t. $(\gamma \circ \lambda')\ (\varphi \wedge \sigma \wedge \rho) = \top$ holds. And since $\varphi \wedge \sigma \wedge \rho \models_\Lambda (x \vee C)$ (due to Σ and Lemma 14), we know that then there exists as well a λ'' s.t. $(\tau \circ \lambda'')\ (\varphi \wedge \sigma \wedge \rho \wedge (x \vee C)) = \top$. □

From the projected implication established in Lemma 15, it can be shown that there is a projected equivalence maintained by every step of our calculus.

Lemma 16. *In any derivation in our calculus starting from the initial state the property* $\varphi^i_j \wedge \sigma^i_j \equiv_\Lambda \varphi^i_{j+1} \wedge \sigma^i_{j+1}$ *holds in each phase* $i = 0 \ldots n$ *and for each* j *with* $0 \leq j < k_i$.

Proof. By induction over the transition, ignoring the indices i and j. Only rules Strengthen and Drop change the combined formula $\varphi \wedge \sigma$. However, Strengthen strengthens with an Λ-implied clause (due to Lemma 15), while Drop guarantees logical equivalence by ø. □

Lemma 17. *In any derivation starting from the initial state, if no input clauses contain latent variables then the property* $F^i \models_\Lambda \varphi^i_0 \wedge \sigma^i_0 \equiv_\Lambda \varphi^i_1 \wedge \sigma^i_1 \equiv_\Lambda \cdots \equiv_\Lambda \varphi^i_{k_i} \wedge \sigma^i_{k_i}$ *holds.*

Proof. Since the input clauses have no Λ-variables, for all $0 \leq i \leq n$ it holds that $\Delta_i \equiv \exists\Lambda\ (\Delta_i)$ and so for any formula F, we have that $\exists\Lambda\ (F) \wedge \Delta_i \equiv \exists\Lambda\ (F \wedge \Delta_i)$. Further, from Lemma 16 follows that $F^0 = \varphi^0_0 \wedge \varepsilon \equiv_\Lambda \cdots \equiv_\Lambda \varphi^0_{k_0} \wedge \sigma^0_{k_0}$. Now, by an inductive argument and by Lemma 16, we get that $F^{i+1} = F^i \wedge \Delta_{i+1} \equiv_\Lambda \varphi^i_{k_i} \wedge \sigma^i_{k_i} \wedge \Delta_{i+1} = \varphi^{i+1}_0 \wedge \sigma^{i+1}_0 \equiv_\Lambda \cdots \equiv_\Lambda \varphi^{i+1}_{k_{i+1}} \wedge \sigma^{i+1}_{k_{i+1}}$. □

As a corollary, we get that the learned clauses can be kept even after new clauses were added, as long as users are not using latent variables.

Corollary 18. *If no input clauses contain latent variables then* $F^{i+1} \models_\Lambda \rho^i_{k_i}$ *holds.*

What remains to show is that solution reconstruction works as before in our extended calculus. For that, we use a refined definition of reconstruction property from [34, Sect. 6.3].

Definition 19 (Reconstruction Property [34])**.** *A state* $\varphi\ [\rho]\ \sigma$ *satisfies the reconstruction property w.r.t. a formula* F *iff for all total truth assignment* τ *over* $\Gamma \cup \Lambda$ *satisfying* φ*, the result of the reconstruction function* $\mathcal{R}_\sigma(\tau)$ *is a satisfying assignment of* F.

Theorem 20. *Starting from the initial state, in any reachable state $\varphi_j^i \; [\, \rho_j^i \,] \; \sigma_j^i$, the reconstruction property holds w.r.t. F^i.*

Proof Sketch. Relying on our refined definitions and lemmas (Def. 11, Lemma 12, and Lemma 9), it can be shown by induction that every step (including EXTEND) maintains the property. The proof remains technically the same as in [34] (see in Appendix). □

Putting everything together, we get our final lemma showing that the incremental calculus with EXTEND is correct.

Theorem 21 (Correctness)**.** *Starting from the initial state, in any reachable state $\varphi_j^i \; [\, \rho_j^i \,] \; \sigma_j^i$, the property $F^i \equiv_{sat} \varphi_j^i \wedge \rho_j^i$ holds.*

Proof. If $\varphi_j^i \wedge \rho_j^i$ is satisfiable, then F^i is also satisfiable due to Thm. 20. From Lemma 15 follows that if $\varphi_j^i \wedge \rho_j^i$ is unsatisfiable, then φ_j^i is unsatisfiable as well, and then, due to Lemma 17, F^i must be unsatisfiable too. □

4. Discussion

The following example illustrates how our extended calculus can capture BCA and thereby for example BVA or other forms of extended resolution.

Example 22. *Let $F^0 = (a \vee b) \wedge (b \vee c) \wedge (c \vee d) \wedge (d \vee a)$ be the first formula to be solved. Once all clauses are added (via ADDCLAUSES), the derivation starts with $F^0 \; [\, \emptyset \,] \; \varepsilon$. Assume that the solver decides to introduce a new latent variable l, with the definition $(l \leftrightarrow (\neg a \vee \neg c))$. For that, it needs to add (in an arbitrary order) the following three blocked clauses: $\{(a \vee l), (c \vee l), (\neg a \vee \neg c \vee \neg l)\}$. Since $l, \neg l \notin F^0$ and the reconstruction stack is empty, we can see that each clause fulfils $\boxed{\Sigma}$ hence applying EXTEND thrice can lead to a state $F^0 \; [\, (a \vee l) \wedge (c \vee l) \wedge (\neg a \vee \neg c \vee \neg l) \,] \; \varepsilon$. Now, the clauses $(b \vee \neg l)$ and $(d \vee \neg l)$ are both implied and so can be learned via LEARN$^-$. With STRENGTHEN the solver can move some of the learned clauses, leading to a state $F^0 \wedge (a \vee l) \wedge (c \vee l) \wedge (b \vee \neg l) \wedge (d \vee \neg l) \; [\, (\neg a \vee \neg c \vee \neg l) \,] \; \varepsilon$. Now DROP can be used to drop* all *clauses of F^0 and FORGET can also remove the remaining redundant definition clause. As a result, we get $(a \vee l) \wedge (c \vee l) \wedge (b \vee \neg l) \wedge (d \vee \neg l) \; [\, \emptyset \,] \; \varepsilon$. At that point, since the reconstruction stack is empty, arbitrary l-free clauses can be added via ADDCLAUSES.*

Here the solver fully defined the meaning of the new latent variable (by learning three blocked clauses), but in theory it would have been correct to learn only some of these clauses.

A further interesting insight is that our calculus ensures that the solved formula remains satisfiability equivalent and solution reconstruction gives a model to the original problem. However, as the next example shows, the final value of the latent variables is not defined.

Example 23. *Assume that we start with a formula consisting of a single unit clause, i.e., $F_0 = (a)$ and so the initial state is $\{a\} \; [\, \emptyset \,] \; \varepsilon$. Assume, the solver uses WEAKEN$^+$ to remove the only clause of the formula, ending in the state $\emptyset \; [\, \emptyset \,] \; (a : a)$. Then, the solver can learn via EXTEND the clause $(x \vee \neg a)$ witnessed by x, where $x \in \Lambda$, yielding the state $\emptyset \; [\, \{(x \vee \neg a)\} \,] \; (a : a)$. Now, by STRENGTHEN, the clause can be moved to the irredundant set, resulting in the state $\{(x \vee \neg a)\} \; [\, \emptyset \,] \; (a : a)$. Then, starting from the initial solution $\tau = \{\neg x, \neg a\}$, solution reconstruction results in a model of the input formula $\tau = \{\neg x, a\}$. However, while that the solution satisfies F_0, it falsifies the clause learnt by EXTEND.*

This result is not too surprising, as our extended calculus is based on projected implication, i.e., we only claim that *there exists* an extension of the solution to the latent variables, while the actual value of these variables is not considered. Similarly, the original non-incremental calculus of [17] does not guarantee that clauses added by the solver are satisfied after solution reconstruction.

5. Conclusion and Future Work

The main motivation behind this paper was our intention to implement and combine BVA with the incremental features of CaDiCaL [39]. Though intuitively this should work without much complications, we had to establish in theory that our implementation is actually correct. Thus, here we introduced an extension of the incremental inprocessing calculus [26] with a new rule that allows learning in certain cases based on more general redundancies.

The new framework supports only single witness literals, extending it to more general witnesses, that is necessary to capture super-blocked clauses [13] and other PR-based techniques [35, 40, 41], remains future work. Further, note that our calculus captures only the internal inprocessing steps of the solver and does not allow users to interfere with it. A more general learning mechanism (as it is available in the non-incremental inprocessing calculus [17]) remains intriguing future work.

Acknowledgments

This work was supported in part by the Austrian Science Fund (FWF) under project T-1306, the state of Baden-Württemberg through bwHPC, the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG, and by a gift from Intel Corporation.

Declaration on Generative AI

During preparation, the authors used DeepL and Mistral-AI to check the grammar and spelling of this work. After using these tools and services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] M. Järvisalo, A. Biere, M. Heule, Blocked clause elimination, in: J. Esparza, R. Majumdar (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings, volume 6015 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 129–144. doi:10.1007/978-3-642-12002-2_10.

[2] N. Eén, A. Biere, Effective preprocessing in SAT through variable and clause elimination, in: F. Bacchus, T. Walsh (Eds.), Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings, volume 3569 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 61–75. doi:10.1007/11499107_5.

[3] B. Aspvall, M. F. Plass, R. E. Tarjan, A linear-time algorithm for testing the truth of certain quantified Boolean formulas, Inf. Process. Lett. 8 (1979) 121–123. doi:10.1016/0020-0190(79)90002-4.

[4] A. Van Gelder, Y. K. Tsuji, Satisfiability testing with more reasoning and less guessing, in: D. S. Johnson, M. A. Trick (Eds.), Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, DIMACS/AMS, 1993, pp. 559–586.

[5] C. M. Li, Integrating equivalency reasoning into Davis-Putnam procedure, in: H. A. Kautz, B. W. Porter (Eds.), Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA., AAAI Press / The MIT Press, 2000, pp. 291–296.

[6] A. del Val, Simplifying binary propositional theories into connected components twice as fast, in: R. Nieuwenhuis, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001, Havana, Cuba, December 3-7, 2001, Proceedings, volume 2250 of *Lecture Notes in Computer Science*, Springer, 2001, pp. 392–406. doi:10.1007/3-540-45653-8_27.

[7] R. E. Bryant, M. J. H. Heule, Generating extended resolution proofs with a bdd-based SAT solver, ACM Trans. Comput. Log. 24 (2023) 31:1–31:28.
[8] J. E. Reeves, M. J. H. Heule, R. E. Bryant, Preprocessing of propagation redundant clauses, J. Autom. Reason. 67 (2023) 31.
[9] J. E. Reeves, M. J. H. Heule, R. E. Bryant, From clauses to klauses, in: CAV (1), volume 14681 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 110–132.
[10] A. Haberlandt, H. Green, SBVA-CaDiCaL and SBVA-Kissat: Structured bounded variable addition, in: T. Balyo, N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proc. of SAT Competition 2023 – Solver and Benchmark Descriptions, volume B-2023-1 of *Department of Computer Science Report Series B*, University of Helsinki, 2023, p. 18.
[11] T. Balyo, M. J. H. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions, Department of Computer Science Series of Publications B, Department of Computer Science, University of Helsinki, Finland, 2023.
[12] N. Manthey, M. Heule, A. Biere, Automated reencoding of boolean formulas, in: A. Biere, A. Nahir, T. E. J. Vos (Eds.), Hardware and Software: Verification and Testing - 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers, volume 7857 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 102–117. URL: https://doi.org/10.1007/978-3-642-39611-3_14. doi:10.1007/978-3-642-39611-3_14.
[13] B. Kiesl, M. Seidl, H. Tompits, A. Biere, Super-blocked clauses, in: IJCAR, volume 9706 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 45–61.
[14] G. S. Tseitin, On the Complexity of Derivation in Propositional Calculus, Springer Berlin Heidelberg, Berlin, Heidelberg, 1983, pp. 466–483. URL: https://doi.org/10.1007/978-3-642-81955-1_28. doi:10.1007/978-3-642-81955-1_28.
[15] O. Kullmann, On a generalization of extended resolution, Discret. Appl. Math. 96-97 (1999) 149–176.
[16] G. Audemard, G. Katsirelos, L. Simon, A restriction of extended resolution for clause learning SAT solvers, in: AAAI, AAAI Press, 2010, pp. 15–20.
[17] M. Järvisalo, M. Heule, A. Biere, Inprocessing rules, in: B. Gramlich, D. Miller, U. Sattler (Eds.), Proc. of Automated Reasoning - 6th International Joint Conference, IJCAR 2012, volume 7364 of *LNCS*, Springer, 2012, pp. 355–370. URL: https://doi.org/10.1007/978-3-642-31365-3_28. doi:10.1007/978-3-642-31365-3_28.
[18] A. Biere, D. Kröning, Sat-based model checking, in: Handbook of Model Checking, Springer, 2018, pp. 277–303.
[19] F. Bacchus, M. Järvisalo, R. Martins, Maximum satisfiability, in: Handbook of Satisfiability, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021, pp. 929–991.
[20] R. Nieuwenhuis, A. Oliveras, C. Tinelli, Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(T), J. ACM 53 (2006) 937–977.
[21] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: Handbook of Satisfiability, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021, pp. 1267–1329.
[22] N. Eén, N. Sörensson, An extensible sat-solver, in: SAT, volume 2919 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 502–518.
[23] N. Eén, N. Sörensson, Temporal induction by incremental SAT solving, in: BMC@CAV, volume 89 of *Electronic Notes in Theoretical Computer Science*, Elsevier, 2003, pp. 543–560.
[24] S. Kupferschmid, M. Lewis, T. Schubert, B. Becker, Incremental preprocessing methods for use in BMC, Formal Methods Syst. Des. 39 (2011) 185–204.
[25] A. Nadel, V. Ryvchin, O. Strichman, Preprocessing in incremental SAT, in: A. Cimatti, R. Sebastiani (Eds.), Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings, volume 7317 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 256–269. URL: https://doi.org/10.1007/978-3-642-31612-8_20. doi:10.1007/978-3-642-31612-8_20.
[26] K. Fazekas, A. Biere, C. Scholl, Incremental inprocessing in SAT solving, in: M. Janota, I. Lynce

(Eds.), Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings, volume 11628 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 136–154. URL: https://doi.org/10.1007/978-3-030-24258-9_9. doi:10.1007/978-3-030-24258-9_9.

[27] O. Beyersdorff, M. Janota, F. Lonsing, M. Seidl, Quantified boolean formulas, in: Handbook of Satisfiability, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021, pp. 1177–1221.

[28] K. Fazekas, M. Seidl, A. Biere, A duality-aware calculus for quantified boolean formulas, in: J. H. Davenport, V. Negru, T. Ida, T. Jebelean, D. Petcu, S. M. Watt, D. Zaharie (Eds.), 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2016, Timisoara, Romania, September 24-27, 2016, IEEE, 2016, pp. 181–186. URL: https://doi.org/10.1109/SYNASC.2016.038. doi:10.1109/SYNASC.2016.038.

[29] D. Fried, A. Nadel, R. Sebastiani, Y. Shalmon, Entailing generalization boosts enumeration, in: SAT, volume 305 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, pp. 13:1–13:14.

[30] S. Möhle, R. Sebastiani, A. Biere, Four flavors of entailment, in: SAT, volume 12178 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 62–71.

[31] G. Spallitta, R. Sebastiani, A. Biere, Disjoint projected enumeration for SAT and SMT without blocking clauses, CoRR abs/2410.18707 (2024).

[32] R. Sebastiani, Are you satisfied by this partial assignment?, CoRR abs/2003.04225 (2020). URL: https://arxiv.org/abs/2003.04225. arXiv:2003.04225.

[33] S. Möhle, R. Sebastiani, A. Biere, On enumerating short projected models, Discret. Appl. Math. 361 (2025) 412–439.

[34] K. Fazekas, On SAT-based Solution Methods for Computational Problems, Ph.D. thesis, Informatik, Johannes Kepler University Linz, 2020.

[35] M. J. H. Heule, B. Kiesl, A. Biere, Strong extension-free proof systems, J. Autom. Reason. 64 (2020) 533–554.

[36] J. Berg, B. Bogaerts, J. Nordström, A. Oertel, D. Vandesande, Certified core-guided maxsat solving, in: CADE, volume 14132 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 1–22.

[37] K. Wagner, Formalization of the Inprocessing Rules and the Reconstruction Stack in Isabelle, Master's thesis, University of Freiburg, Freiburg, Germany, 2023.

[38] M. Järvisalo, A. Biere, Reconstructing solutions after blocked clause elimination, in: SAT, volume 6175 of *LNCS*, Springer, 2010, pp. 340–345.

[39] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, F. Pollitt, Cadical 2.0, in: A. Gurfinkel, V. Ganesh (Eds.), Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I, volume 14681 of *LNCS*, Springer, 2024, pp. 133–152. doi:10.1007/978-3-031-65627-9_7.

[40] B. Kiesl, A. Rebola-Pardo, M. J. H. Heule, A. Biere, Simulating strong practical proof systems with extended resolution, J. Autom. Reason. 64 (2020) 1247–1267.

[41] L. A. Barnett, A. Biere, Non-clausal redundancy properties, in: A. Platzer, G. Sutcliffe (Eds.), Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings, volume 12699 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 252–272. doi:10.1007/978-3-030-79876-5_15.

A. Proof of Theorem 20

Theorem 20. *Starting from the initial state, in any reachable state* $\varphi_j^i\ [\,\rho_j^i\,]\ \sigma_j^i$*, the reconstruction property holds w.r.t.* F^i*.*

Proof. In the initial state $i, j = 0$, $\varphi_0^0 = F^0$, $\sigma_0^0 = \varepsilon$, and so for any total satisfying assignment τ of F^0, $\mathcal{R}_\varepsilon(\tau)(F^0) = \tau\ (F^0) = \top$. Assume that in a state j of a phase i (where $0 \leq j < k_i$ and $0 \leq i \leq n$), the reconstruction property w.r.t. F^i holds, i.e., for all complete assignment τ over $\Gamma \cup \Lambda$ if $\tau(\varphi_j^i) = \top$, then $\mathcal{R}_{\sigma_j^i}(\tau)(F^i) = \top$. Now, we need to show that for any total τ if $\tau(\varphi_{j+1}^i) = \top$ holds, then $\mathcal{R}_{\sigma_{j+1}^i}(\tau)(F^i) = \top$ holds as well. Note that F^i is not changed by any steps of the calculus (AddClauses starts a new phase and i is increased). Rules Learn$^-$, Forget, and Extend do not change φ_j^i nor σ_j^i, thus the property remains true. In case of rule Drop, it follows from ø, that if $\tau(\varphi_{j+1}^i) = \top$ then $\tau(\varphi_{j+1}^i \wedge C) = \top$ as well, thus $\mathcal{R}_{\sigma_{j+1}^i}(\tau)(F^i) = \top$ holds by induction, and similarly for Strengthen with $\varphi_{j+1}^i = \varphi_j^i \wedge C$.

For Weaken$^+$, consider a total truth assignment τ' over $\Gamma \cup \Lambda$ s.t. $\tau'(\varphi_{j+1}^i) = \tau'(\varphi_j^i \setminus C) = \top$. If $\tau'(C) = \top$, then $\tau'(\varphi_j^i) = \top$, thus $\mathcal{R}_{\sigma_j^i}(\tau')(F^i) = \top$ by induction. Further, in that case from Def. 11 follows that $\mathcal{R}_{\sigma_{j+1}^i}(\tau') = \mathcal{R}_{\sigma_j^i \cdot (\omega:C)}(\tau') = \mathcal{R}_{\sigma_j^i}(\tau')$ since $\tau'(C) = \top$. If $\tau'(C) \neq \top$, then $\tau'(C) = \bot$ as τ' is total. From ♭, Lemma 12, and Lemma 9 follows that $(\tau' \circ \omega)\ (\varphi_j^i) = \top$ and so by induction ($\tau' \circ \omega$ is still total over $\Gamma \cup \Lambda$), $\mathcal{R}_{\sigma_j^i}(\tau' \circ \omega)(F^i) = \top$ where $\mathcal{R}_{\sigma_j^i}(\tau' \circ \omega) = \mathcal{R}_{\sigma_j^i \cdot (\omega:C)}(\tau')$ since $\tau'(C) \neq \top$. Thus, $\mathcal{R}_{\sigma_j^i \cdot (\omega:C)}(\tau')(F^i) = \mathcal{R}_{\sigma_{j+1}^i}(\tau')(F^i) = \top$.

Assume Restore is applied, where $\sigma_j^i = \sigma \cdot (\omega : C) \cdot \sigma'$. Then for any τ' s.t. $\tau'(\varphi_{j+1}^i) = \top$ we have that $\tau'(C) = \top$ and $\tau'(\varphi_j^i) = \top$ since $\varphi_{j+1}^i = \varphi_j^i \wedge C$. Further, from Def. 11 follows that $\mathcal{R}_{\sigma_{j+1}^i}(\tau') = \mathcal{R}_{\sigma \cdot \sigma'}(\tau') = \mathcal{R}_\sigma(\mathcal{R}_{\sigma'}(\tau'))$. As C is clean w.r.t. σ' (by ∂), from Lemma 12 follows that $\mathcal{R}_{\sigma'}(\tau')(C) = \top$, thus $\mathcal{R}_\sigma(\mathcal{R}_{\sigma'}(\tau')) = \mathcal{R}_{\sigma \cdot (\omega:C)}(\mathcal{R}_{\sigma'}(\tau')) = \mathcal{R}_{\sigma \cdot (\omega:C) \cdot \sigma'}(\tau')$ (again by Def. 11), which is actually $\mathcal{R}_{\sigma_j^i}(\tau')$ where by induction the property holds.

When a new phase starts via AddClauses (i.e., $0 \leq i < n$ and $j = k_i$), formula $\varphi_{k_i}^i$ is extended with a set of clauses Δ_{i+1}. Let τ' be a total assignment over $\Gamma \cup \Lambda$ s.t. $\tau'(\varphi_{k_i}^i \wedge \Delta_{i+1}) = \tau'(\varphi_0^{i+1}) = \top$. Note that $\mathcal{R}_{\sigma_0^{i+1}}(\tau') = \mathcal{R}_{\sigma_{k_i}^i}(\tau')$ as AddClauses does not change the reconstruction stack. By the inductive assumption we have that $\mathcal{R}_{\sigma_{k_i}^i}(\tau')(F^i) = \top$. Further, as $F^{i+1} = F^i \wedge \Delta_{i+1}$, where each clause of Δ_{i+1} is clean w.r.t. $\sigma_{k_i}^i$ (by ℐ), from Lemma 12 follows that $\mathcal{R}_{\sigma_{k_i}^i}(\tau')(\Delta_{i+1}) = \mathcal{R}_{\sigma_0^{i+1}}(\tau')(\Delta_{i+1}) = \top$ as well. Thus, $\mathcal{R}_{\sigma_{k_i}^i}(\tau')(F^i \wedge \Delta_{i+1}) = \mathcal{R}_{\sigma_0^{i+1}}(\tau')(F^{i+1}) = \top$. □