Covered Clauses Are Not Propagation Redundant*

Lee A. Barnett, David Cerna, and Armin Biere

Johannes Kepler University Linz
Altenbergerstraße 69, 4040 Linz, Austria
{lee.barnett,david.cerna,armin.biere}@jku.at

Abstract. Propositional proof systems based on recently-developed redundancy properties admit short refutations for many formulas traditionally considered hard. Redundancy properties are also used by procedures which simplify formulas in conjunctive normal form by removing redundant clauses. Revisiting the covered clause elimination procedure, we prove the correctness of an explicit algorithm for identifying covered clauses, as it has previously only been implicitly described. While other elimination procedures produce redundancy witnesses for compactly reconstructing solutions to the original formula, we prove that witnesses for covered clauses are hard to compute. Further, we show that not all covered clauses are propagation redundant, the most general, polynomially-verifiable standard redundancy property. Finally, we close a gap in the literature by demonstrating the complexity of clause redundancy itself.

Keywords: SAT · Clause Elimination · Redundancy

1 Introduction

Boolean satisfiability (SAT) solvers have become successful tools for solving reasoning problems in a variety of applications, from formal verification [6] and security [27] to pure mathematics [10,19,24]. Significant recent progress in the design of SAT solvers has come as a result of exploiting the notion of clause redundancy (for instance, [14,16,18]). For a propositional formula F in conjunctive normal form (CNF), a clause C is redundant if it can be added to, or removed from, F without affecting whether F is satisfiable [23].

In particular, redundancy forms a basis for clausal proof systems. These systems refute an unsatisfiable CNF formula F by listing instructions to add or delete clauses to or from F, where the addition of a clause C is permitted only if C meets some criteria ensuring its redundancy. By eventually adding the empty clause, the formula is proven to be unsatisfiable. Crucially, the redundancy criteria of a system can also be used as an inference rule by a solver searching for such refutations, or for satisfying assignments.

* Supported by the Austrian Science Fund (FWF) under project W1255-N23, the LogiCS Doctoral College on Logical Methods in Computer Science, as well as the LIT Artificial Intelligence Lab funded by the State of Upper Austria.

Proof systems based on the recently introduced PR (Propagation Redundancy) criteria [15] have been shown to admit short refutations of the famous pigeonhole formulas [11,17]. These are known to have only exponential-size refutations in many systems, including resolution [9] and constant-depth Frege systems [1], but have polynomial-size PR refutations. In fact, many problems typically considered hard have short PR refutations, spurring interest in these systems from the viewpoint of proof complexity [4]. Further, systems based on PR are strong even without introducing new variables, and have the potential to afford substantial improvements to SAT solvers (such as in [16,18]).

The PR criteria is very general, encompassing nearly all other established redundancy criteria, and it is NP-complete to decide whether it is met by a given clause [18]. However, when the clause is given alongside a witness, a partial assignment providing additional evidence for the clause's redundancy, the PR criteria can be polynomially verified [15]. SAT solvers producing refutations in the PR system must find and record a witness for each PR clause addition.

Redundancy is also a basis for clause elimination procedures, which simplify a CNF formula by removing redundant clauses [12,14]. These are useful preprocessing and inprocessing techniques that also make use of witnesses, but for the task of solution reconstruction: correcting satisfying assignments found after simplifying to ensure they solve the original formula. A witness for a clause C details how to fix assignments falsifying C without falsifying other clauses in the formula [15,17], so solvers using elimination procedures that do not preserve formula equivalence typically provide a witness for each removed clause.

Covered clause elimination (CCE) [13] is a strong procedure which removes covered clauses, a generalization of blocked clauses [20,25], and has been implemented in various SAT solvers (for example, [2,3,8]) and the CNF preprocessing tool Coprocessor [26]. CCE does not preserve formula equivalence, but provides no witnesses for the clauses it removes. Instead, it uses a complex technique to reconstruct solutions in multiple steps, requiring at times a quadratic amount of space to reconstruct a single clause [14,21]. CCE has so far only been implicitly described, and it is not clear how to produce witnesses for covered clauses.

In this paper we provide an explicit algorithm for identifying covered clauses, and show that their witnesses are difficult to produce. We also demonstrate that although covered clauses are redundant, they do not always meet the criteria required by PR. This suggests it may be beneficial to consider redundancy properties beyond PR which allow alternative types of witnesses. There has already been some work in this direction with the introduction of the SR (Substitution Redundancy) property by Buss and Thapen [4].

The paper is organized as follows. In section 2 we provide necessary background and terminology, while section 3 reviews covered clause elimination, provides the algorithm for identifying covered clauses, and proves that this algorithm and its reconstruction strategy are correct. Section 4 includes proofs about witnesses for covered clauses, and shows that they are not encompassed by PR. In section 5 we consider the complexity of deciding clause redundancy in general, followed by a conclusion and discussion of future work in section 6.

2 Preliminaries

A literal is a boolean variable x or its negation $\neg x$. A clause is a disjunction of literals, and a formula is a conjunction of clauses. We often identify a clause with the set of its literals, and a formula with the set of its clauses. For a set of literals S we write $\neg S$ to refer to the set $\neg S = \{\neg l \mid l \in S\}$. The set of variables occurring in a formula F is written $var(F)$. The empty clause is represented by $\bot$, and the satisfied clause by $\top$.

An *assignment* is a function from a set of variables to the truth values *true* and *false*. An assignment is *total* for a formula F if it assigns a value for every variable in $var(F)$, otherwise it is partial. An assignment is represented by the set of literals it assigns to true. The *composition* of assignments τ and υ is

$$\tau \circ \upsilon(x) = \begin{cases} \tau(x) & \text{if } x, \neg x \notin \upsilon \\ \upsilon(x) & \text{otherwise} \end{cases}$$

for a variable x in the domain of τ or υ. For a literal l, we write τ_l to represent the assignment $\tau \circ \{l\}$. An assignment *satisfies* (resp., *falsifies*) a variable if it assigns that variable true (resp., false). Assignments are lifted to functions assigning literals, clauses, and formulas in the usual way.

Given an assignment τ and a clause C, the *partial application* of τ to C is written $C|_\tau$ and is defined as follows: $C|_\tau = \top$ if C is satisfied by τ, otherwise, $C|_\tau = \{l \mid l \in C \text{ and } \neg l \notin \tau\}$. Likewise, the partial application of the assignment τ to a formula F is written $F|_\tau$ and defined by: $F|_\tau = \top$ if σ satisfies F, otherwise $F|_\tau = \{C|_\tau \mid C \in F \text{ and } C|_\tau \neq \top\}$. *Unit propagation* refers to the iterated application of the *unit clause rule*, replacing F by $F|_{\{l\}}$ for each unit clause $(l) \in F$, until there are no unit clauses left.

We write $F \vDash G$ to indicate that every assignment satisfying F, and which is total for G, satisfies G as well. Further, we write $F \vdash_1 G$ to mean F implies G *by unit propagation*: for every $D \in G$, unit propagation on $\neg D \wedge F$ produces $\bot$.

A clause C is *redundant* with respect to a formula F if the formulas $F \setminus \{C\}$ and $F \cup \{C\}$ are satisfiability-equivalent: both satisfiable, or both unsatisfiable [23]. The following theorem provides a characterization of clause redundancy based on logical implication.

Theorem 1 (Heule, Kiesel, and Biere [17]). *A non-empty clause C is redundant with respect to a formula F (with $C \notin F$) if and only if there is a partial assignment ω such that ω satisfies C, and $F|_\alpha \vDash F|_\omega$, where $\alpha = \neg C$.*

As a result, redundancy can be shown by providing a *witnessing assignment* ω (or *witness*) and demonstrating that $F|_\alpha \vDash F|_\omega$. When the logical implication relation "$\vDash$" is replaced with "$\vdash_1$," the result is the definition of a *propagation redundant* or PR *clause*, and ω is called a PR *witness* [15]. Determining whether a clause is PR with respect to a formula is NP-complete [18], but since it can be decided in polynomial time whether $F \vdash_1 G$ for arbitrary formulas F and G, it can be efficiently decided whether a given assignment is a PR witness.

A *clause elimination procedure* iteratively identifies and removes clauses satisfying a particular redundancy property from a formula, until no such clauses remain. A simple example is *subsumption elimination*, which removes any clauses $C \in F$ that are *subsumed* by another clause $D \in F$; that is, $D \subseteq C$. Subsumption elimination is *model-preserving*, as it only removes clauses C such that any assignment satisfying $F \setminus \{C\}$ also satisfies $F \cup \{C\}$.

Some clause elimination procedures are not model-preserving. *Blocked clause elimination* [20,25] iteratively removes from a formula F any clauses C satisfying the following property: C is *blocked* by a literal $l \in C$ if for every clause $D \in F$ containing $\neg l$, there is some other literal $k \in C$ with $\neg k \in D$. For a blocked clause C, there may be assignments satisfying $F \setminus \{C\}$ which falsify $F \cup \{C\}$. However, blocked clauses are redundant, so if $F \cup \{C\}$ is unsatisfiable, then so is $F \setminus \{C\}$, thus blocked clause elimination is still *satisfiability-preserving*.

Clause elimination procedures which are not model-preserving must provide a way to *reconstruct* solutions to the original formula out of solutions to the reduced formula. Witnesses provide a convenient framework for reconstruction: if C is redundant with respect to F, and τ is a total or partial assignment satisfying F but not C, then $\tau \circ \omega$ satisfies $F \cup \{C\}$, for any witness ω for C with respect to F [7,17]. For reconstructing solutions after removing multiple clauses, a sequence σ of *witness-labeled clauses* $(\omega : C)$, called a *reconstruction stack*, can be maintained and used as follows [7,21].

Definition 1. *Given a sequence σ of witness-labeled clauses, the* reconstruction function *(w.r.t. σ) is defined recursively as follows, for an assignment τ:* [1]

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

For a set of clauses S, a sequence σ of witness-labeled clauses satisfies the *reconstruction property* for S, or is a *reconstruction sequence* for S, with respect to a formula F if $\mathcal{R}_\sigma(\tau)$ satisfies $F \cup S$ for any assignment τ satisfying $F \setminus S$. As long as a witness is recorded for each clause C removed by a non-model-preserving procedure, even combinations of different clause elimination procedures can be used to simplify the same formula. Specifically, $\sigma = (\omega_1 : C_1) \cdots (\omega_n : C_n)$ is a reconstruction sequence for $\{C_1, \ldots, C_n\} \subseteq F$ if ω_i is a witness for C_i with respect to $F \setminus \{C_1, \ldots, C_i\}$, for all $1 \leq i \leq n$ [7].

The following lemma results from the fact that the reconstruction function satisfies $\mathcal{R}_{\sigma\cdot\sigma'}(\tau) = \mathcal{R}_\sigma(\mathcal{R}_{\sigma'}(\tau))$, for any sequences σ, σ' and assignment τ [7].

Lemma 1. *If σ is a reconstruction sequence for a set of clauses S with respect to $F \cup \{C\}$, and σ' is a reconstruction sequence for $\{C\}$ with respect to F, then $\sigma \cdot \sigma'$ is a reconstruction sequence for S with respect to F.*

[1] This improved variant over [7] is due to Christoph Scholl (3rd author of [7]).

3 Covered Clause Elimination

This section reviews covered clause elimination (CCE) and its asymmetric variant (ACCE), introduced by Heule, Järvisalo, and Biere [13], and presents an explicit algorithm implementing the more general ACCE procedure. The definitions as given here differ slightly from the original work, but are generally equivalent. A proof of correctness for the algorithm and its reconstruction sequence are given.

CCE is a clause elimination procedure which iteratively extends a clause by the addition of so-called "covered" literals. If at some point the extended clause becomes blocked, the original clause is redundant and can be eliminated. To make this precise, the set of *resolution candidates* in F of C upon l, written $\mathrm{RC}(F, C, l)$, is defined as the collection of clauses in F with which C has a non-tautological resolvent upon l (where "$\otimes_l$" denotes resolution):

$$\mathrm{RC}(F, C, l) = \{C' \vee \neg l \in F \mid C' \vee \neg l \otimes_l C \not\equiv \top\}.$$

The *resolution intersection* in F of C upon l, written $\mathrm{RI}(F, C, l)$, consists of those literals occurring in each of the resolution candidates, apart from $\neg l$:

$$\mathrm{RI}(F, C, l) = \big(\bigcap \mathrm{RC}(F, C, l)\big) \setminus \{\neg l\}.$$

If $\mathrm{RI}(F, C, l) \neq \emptyset$, its literals are covered by l and can be used to extend C.

Definition 2. *A literal k is* covered *by $l \in C$ with respect to F if $k \in \mathrm{RI}(F, C, l)$. A literal is covered by C if it is covered by some literal in C.*

Covered literals can be added to a clause in a satisfiability-preserving manner, meaning that if the extended clause $C \cup \mathrm{RI}(F, C, l)$ is added to F, then C is redundant. In fact, C is a PR clause.

Proposition 1. *C is* PR *with respect to $F' = F \wedge (C \cup \mathrm{RI}(F, C, l))$ with witness $\omega = \alpha_l$, for $l \in C$ and $\alpha = \neg C$.*

Proof. Consider a clause $D|_\omega \in F'|_\omega$, for some $D \in F'$. We prove that ω is a PR witness by showing that $F'|_\alpha$ implies $D|_\omega$ by unit propagation. First, we know $l \notin D$, since otherwise $D|_\omega = \top$ would vanish in $F|_\omega$. If also $\neg l \notin D$, this means $D|_\omega = D|_\alpha$, and therefore $F'|_\alpha \vdash_1 D|_\omega$. Now, suppose $\neg l \in D$. Notice that D contains no other literal k such that $\neg k \in C$, since otherwise $D|_\omega = \top$ here as well. As a result $D \in \mathrm{RC}(F, C, l)$, so $\mathrm{RI}(F, C, l) \subset D$ and $\mathrm{RI}(F, C, l) \setminus C \subseteq D|_\omega$. Notice $\mathrm{RI}(F, C, l) \setminus C = (C \cup \mathrm{RI}(F, C, l))|_\alpha \in F'|_\alpha$, therefore $F'|_\alpha \vdash_1 D|_\omega$. □

Consequently, C is redundant with respect to $F \cup \{C'\}$ for any $C' \supseteq C$ constructed by iteratively adding covered literals to C. In other words, F and $(F \setminus \{C\}) \cup \{C'\}$ are satisfiability-equivalent, so that C could be replaced by C' in F without affecting the satisfiability of the formula. Thus if some such extension C' would be blocked in F, that C' would be redundant, and therefore C is redundant itself. CCE identifies and removes such clauses from F.

Definition 3. *A clause C is* covered *in F if an extension of C by iteratively adding covered literals is blocked.*

CCE refers to the following procedure: while some clause C in F is covered, remove C (that is, replace F with $F \setminus \{C\}$).

ACCE strengthens this procedure by extending clauses using a combination of covered literals and asymmetric literals. A literal k is *asymmetric* to C with respect to F if there is a clause $C' \vee \neg k \in F$ such that $C' \subseteq C$. The addition of asymmetric literals to a clause is model-preserving, so that the formulas F and $(F \setminus \{C\}) \cup \{C \vee k\}$ are equivalent, for any k which is asymmetric to C [12].

Definition 4. *A clause $C' \supseteq C$ is an* ACC extension *of C with respect to F if C' can be constructed from C by the iterative addition of covered and asymmetric literals. If some* ACC *extension of C is blocked or subsumed in F, then C is an* asymmetric covered clause (ACC)*.*

ACCE performs the following: while some C in F is an ACC, remove C from F. Solvers aiming to eliminate covered clauses more often implement ACCE than plain CCE, since asymmetric literals can easily be found by unit propagation, and ACCE is more powerful than CCE, eliminating more clauses [12,13].

The procedure $\mathsf{ACC}(F, C)$ in Fig. 1 provides an algorithm identifying whether a clause C is an ACC with respect to a formula F. This procedure differs in some ways, and includes optimizations over the original procedure as implicitly given by the definition of ACCE. Notably, two extensions of the original clause C are maintained: E consists of C and any added covered literals, while α tracks C and all added literals, both covered and asymmetric. The literals in α are kept negated, so that $E \subseteq \neg\alpha$, and the clause represented by $\neg\alpha$ is the ACC extension of the original clause C being computed.

The E and α extensions are maintained separately for two purposes. First, the covered literal addition loop (lines 9–16) needs to iterate only over those literals in E, and can ignore those in $(\neg\alpha) \setminus E$, as argued below.

Lemma 2. *If k is covered by $l \in (\neg\alpha) \setminus E$, then $k \in \neg\alpha$ already.*

Proof. If l belongs to $\neg\alpha$ but not to E, then there is some clause $D \vee \neg l$ in F such that $D \subseteq \neg\alpha$. But then $D \vee \neg l$ occurs in $\mathrm{RC}(F, \neg\alpha, l)$, and consequently $\mathrm{RI}(F, \neg\alpha, l) \subseteq D \subseteq \neg\alpha$. Thus $k \in \mathrm{RI}(F, \neg\alpha, l)$ implies $k \in \neg\alpha$. □

Notice that the computation of the literals covered by $l \in E$ also prevents any of these literals already in $\neg\alpha$ from being added again.

The second reason for separating E and α is as follows. When a covered literal is found, or when the extended clause is blocked, the algorithm appends a new witness-labeled clause to the reconstruction sequence σ (lines 11 and 14). Instead of $(\neg\alpha_l : \alpha)$, the procedure adds the shorter witness-labeled clause $(\neg E_l : E)$. The proof of statement (3) in lemma 3 below shows that this is sufficient.

Certain details are omitted, especially concerning the addition of asymmetric literals (lines 6–7), but notice that it is never necessary to recompute $F|_\alpha$ entirely. Instead the assignment falsifying each u newly added to α can simply be applied to the existing $F|_\alpha$. In contrast, the **for each** loop (lines 9–16) should re-iterate over the entirety of E each time, as added literals may allow new coverings:

Example 1. Let $C = a \vee b \vee c$ and

$$F = (\neg a \vee \neg x_1) \wedge (\neg a \vee x_2) \wedge (\neg b \vee \neg x_1) \wedge (\neg b \vee \neg x_2) \wedge (\neg c \vee x_1)$$

Initially, neither a nor b cover any literals, but c covers x_1, so it can be added to the clause. After extending, a in $C \vee x_1$ covers x_2, and b blocks $C \vee x_1 \vee x_2$.

The following lemma supplies invariants for arguing about $\mathsf{ACC}(F, C)$.

Lemma 3. *After each update to α, for the clauses represented by $\neg\alpha$ and E:*

(1) *$\neg\alpha$ is an* ACC *extension of C,*
(2) *$F \cup \{\neg\alpha\} \vDash F \cup \{E\}$, and*
(3) *σ is a reconstruction sequence for $\{C\}$ with respect to $F \cup \{\neg\alpha\}$.*

Proof. Let α_i, σ_i, and E_i refer to the values of α, σ, and E, respectively, after $i \geq 0$ updates to α (so that $\alpha_i \subsetneq \alpha_{i+1}$ for each i, but possibly $\sigma_i = \sigma_{i+1}$ and $E_i = E_{i+1}$). Initially, (1) and (2) hold as $E = \neg\alpha_0 = C$. Further, $\sigma_0 = \epsilon$ is a reconstruction sequence for $\{C\}$ with respect to $F \cup \{C\}$, so (3) holds as well. Assuming these claims hold after update i, we show that they hold after $i + 1$.

First suppose update $i + 1$ is the result of executing line 7.

(1) $\alpha_{i+1} = \alpha_i \cup U$, where $u \in U$ implies (u) is a unit clause in $F|_{\alpha_i}$. Then $\neg\alpha_{i+1}$ is the extension of $\neg\alpha_i$ by the addition of asymmetric literals $\neg U$. Assuming $\neg\alpha_i$ is an ACC extension of C, then so is $\neg\alpha_{i+1}$.
(2) Asymmetric literal addition is model-preserving, so $F \cup \{\neg\alpha_{i+1}\} \vDash F \cup \{\neg\alpha_i\}$. Since E was not updated, $E_{i+1} = E_i$. Assuming $F \cup \{\neg\alpha_i\} \vDash F \cup \{E_i\}$, we get $F \cup \{\neg\alpha_{i+1}\} \vDash F \cup \{E_{i+1}\}$.
(3) Again, asymmetric literal addition is model-preserving. Assuming σ_i is a reconstruction sequence for $\{C\}$ with respect to $F \cup \{\alpha_i\}$, then lemma 1 implies $\sigma_{i+1} = \sigma_i \cdot \varepsilon = \sigma_i$ reconstructs $\{C\}$ with respect to $F \cup \{\neg\alpha_{i+1}\}$.

Now, suppose instead update $i + 1$ is executed in line 16.

(1) $\alpha_{i+1} = \alpha_i \cup \Phi$, for some set of literals $\Phi \neq \emptyset$ constructed for $l \in E \subseteq \neg\alpha$. Notice for $k \in \Phi$ that $k \in \mathrm{RI}(F, \neg\alpha, l)$, so k is covered by $\neg\alpha$. Thus assuming $\neg\alpha_i$ is a ACC extension of C, then $\neg\alpha_{i+1}$ is as well.
(2) Consider an assignment τ satisfying $F \cup \{\neg\alpha_{i+1}\}$. If τ satisfies $\neg\alpha_i \subset \neg\alpha_{i+1}$ then τ satisfies $F \cup \{\neg\alpha_i\}$ and by assumption, $F \cup \{E_i\}$. Since $E_i \subset E_{i+1}$ in this case, τ satisfies $F \cup \{E_{i+1}\}$. If instead τ satisfies some literal in $\neg\alpha_{i+1} \setminus \neg\alpha_i$ then τ satisfies $\Phi \subseteq E_{i+1}$, so τ satisfies $F \cup \{E_{i+1}\}$. Thus $F \cup \{\neg\alpha_{i+1}\} \vDash F \cup \{E_{i+1}\}$ in this case as well.

```
   ACC(F, C)
1    σ := ε
2    E :=  C
3    α := ¬C
4    repeat
5      if ⊥ ∈ F|α then return (true, σ)
6      if there are unit clauses in F|α then
7        α := α ∪ {u} for each unit u
8      else
9        for each l ∈ E
10         G := {D|α | (D ∨ ¬l) ∈ F and D|α ≠ ⊤}
11         if G = ∅ then return (true, σ · (¬E_l : E))
12         Φ := ⋂G
13         if Φ ≠ ∅ then
14           σ := σ · (¬E_l : E)
15           E := E ∪  Φ
16           α := α ∪ ¬Φ
17   until no updates to α
18   return (false, ε)
```

Fig. 1. Asymmetric Covered Clause (ACC) Identification. The procedure $\mathsf{ACC}(F, C)$ maintains a sequence σ of witness-labeled clauses, and two sets of literals E and α. The main loop iteratively searches for literals which could be used to extend C and adds their negations to α, so that the clause represented by $\neg\alpha$ is an ACC extension of C. The set E records only those which could be added as covered literals. If C is an ACC, then $\mathsf{ACC}(F, C)$ returns $(\mathbf{true}, \sigma)$: in line 5 if the extension $\neg\alpha$ becomes subsumed in F, or in line 11 if it becomes blocked. In either case, the witness-labeled clauses in σ form a reconstruction sequence for the clause C. Note that lines 5–7 implement Boolean constraint propagation (over the partial assignment α) and can make use of efficient watched clause data structures, while line 10 has to collect all clauses containing $\neg l$, which are still unsatisfied by α, and thus requires full occurrence lists.

(3) Proposition 1 implies $((\alpha_i)_l : \neg\alpha_i)$ is a reconstruction sequence for $\{\neg\alpha_i\}$ with respect to $F \cup \{\neg\alpha_{i+1}\}$. As $E_i \subseteq \neg\alpha_i$, and $F \cup \{\neg\alpha_i\} \vDash F \cup \{E_i\}$ by assumption, then any τ falsifies $\neg\alpha_i$ if and only if τ falsifies E_i. Since $l \in E_i$ as well, then $((\neg E_i)_l : E_i)$ is, in fact, also a reconstruction sequence for $\{\neg\alpha_i\}$ with respect to $F \cup \{\neg\alpha_{i+1}\}$. Finally, with the assumption σ_i is a reconstruction sequence for C with respect to $F \cup \{\neg\alpha_i\}$ and lemma 1, then $\sigma_{i+1} = \sigma_i \cdot ((\neg E_i)_l : E_i)$ is a reconstruction sequence for $\{C\}$ in $F \cup \{\neg\alpha_{i+1}\}$.

Thus both updates maintain invariants (1)–(3). □

With the help of this lemma we can now prove the following theorem, which shows the correctness of $\mathsf{ACC}(F, C)$.

Theorem 2. *For a formula F and a clause C, the procedure* $\mathsf{ACC}(F, C)$ *returns* $(\textbf{\textit{true}}, \sigma)$ *if and only if C is an* ACC *with respect to F. Further, if* $\mathsf{ACC}(F, C)$ *returns* $(\textbf{\textit{true}}, \sigma)$*, then σ is a reconstruction sequence for $\{C\}$ with respect to F.*

Proof. ($\Rightarrow$) Suppose $(\textbf{true}, \sigma)$ is returned in line 5. Then $\bot \in F|_\alpha$, so there is some $D \in F$ such that $D \subseteq \neg\alpha$; that is, $\neg\alpha$ is subsumed by D. By lemma 3 then an ACC extension of C is subsumed in F, so C is an ACC with respect to F. Further, subsumption elimination is model-preserving, so that lemmas 1 and 3 imply σ is a reconstruction sequence for C with respect to F.

Suppose now that $(\textbf{true}, \sigma)$ is returned in line 11. Then for α and some $l \in E$, all clauses in F with $\neg l$ are satisfied by α. Since $E \subseteq \neg\alpha$, then $\neg\alpha$ is blocked by l. By lemma 3 then C is an ACC with respect to F. Now, α_l is a witness for $\neg\alpha$ with respect to F, and $(\alpha_l : \neg\alpha)$ is a reconstruction sequence for $\{\neg\alpha\}$ in F. Further, $E \subseteq \neg\alpha$, and lemma 3 gives $F \cup \{\neg\alpha\} \vDash F \cup \{E\}$, therefore $((\neg E_i)_l : E_i)$ is a reconstruction sequence for $\{\neg\alpha\}$ in F as well. Then lemma 1 implies $\sigma \cdot (\neg E_l : E)$ is a reconstruction sequence for C with respect to F.

($\Leftarrow$) Suppose C is an ACC; that is, some $C' = C \vee k_1 \vee \cdots \vee k_n$ is blocked or subsumed in F, where k_1 is an asymmetric or covered literal for C, and k_i is an asymmetric or covered literal for $C \vee k_1 \vee \cdots \vee k_{i-1}$ for $i > 1$. Towards a contradiction, assume $\mathsf{ACC}(F, C)$ returns $(\textbf{false}, \varepsilon)$. Then for the final value of α, the clause represented by $\neg\alpha$ is not blocked nor subsumed in F, and hence, $C' \not\subseteq \neg\alpha$. As $C \subseteq \neg\alpha$, there must be some values of i such that $\neg k_i \notin \alpha$.

Let m refer to the least such i; that is, $\neg k_m \notin \alpha$, but $\neg k_i \in \alpha$ for all $1 \leq i < m$. Thus k_m is asymmetric, or covered by, $C_{m-1} = C \vee k_1 \vee \cdots \vee k_{m-1}$.

If k_m is asymmetric to C_{m-1}, there is some clause $D \vee \neg k_m$ in F such that $D \subseteq C_{m-1}$. By assumption, $\neg k_m \notin \alpha$ but $\neg C_{m-1} \subseteq \alpha$. Further, $k_m \notin \alpha$, as otherwise $(D \vee \neg k_m)|_\alpha = \bot$ and $\mathsf{ACC}(F, C)$ would have returned **true**. But $(D \vee \neg k_m)|_\alpha = \neg k_m$ would be a unit in $F|_\alpha$ and added to α by line 7.

If instead k_m is covered by C_{m-1}, then $k_m \in \mathrm{RI}(F, C_{m-1}, l)$ for some literal $l \in C_{m-1} \subseteq \neg\alpha$. In fact $l \in E$, by lemma 2. During the l^{th} iteration of the **for each** loop, then $k_m \in \Phi$, and $\neg k_m$ would be added to α by line 16. □

$\mathsf{ACC}(F, C)$ produces, for any asymmetric covered clause C in F, a reconstruction sequence σ for C with respect to F. This allows ACCE to be used during preprocessing or inprocessing like other clause elimination procedures, appending this σ to the solver's main reconstruction stack whenever an ACC is removed. However, the algorithm does not produce redundancy witnesses for the clauses it removes. Instead, σ consists of possibly many witness-labeled clauses, starting with the redundant clause C, and reconstructs solutions for C in multiple steps.

In contrast, most clause elimination procedures produce a single witness-labeled clause $(\omega : C)$ for each removed clause C. In practice, only the part of ω which differs from $\neg C$ must be recorded; for most procedures this difference includes only literals in C, so that reconstruction for $\{C\}$ needs only linear space in the size of C. In contrast, the size of σ produced by $\mathsf{ACC}(F, C)$ to reconstruct $\{C\}$ can be quadratic in the length of the extended clause.

Example 2. Consider $C = x_0$ and

$$F_n = (\neg x_{n-2} \vee x_{n-1} \vee x_n) \wedge (\neg x_{n-1} \vee \neg x_n) \wedge \bigwedge_{i=1}^{n-2} (\neg x_{i-1} \vee x_i).$$

The extended clause $\neg\alpha = x_0 \vee x_1 \vee \cdots \vee x_n$ is blocked in F_n by x_{n-1}. Then $\mathsf{ACC}(F_n, C)$ returns the pair with **true** and the reconstruction sequence[2]

$$\sigma = (x_0 \wr x_0) \cdot (x_1 \wr x_0 \vee x_1) \cdots (x_{n-2} \wr x_0 \vee x_1 \vee \cdots \vee x_{n-2}) \cdot (x_{n-1} \wr x_0 \vee x_1 \vee \cdots \vee x_n).$$

The extended clause includes n literals, and the size of σ is $O(n^2)$.

4 Witnesses for Covered Clauses

In this section, we consider the specific problem of finding witnesses for (asymmetric) covered clauses. As these clauses are redundant, such witnesses are guaranteed to exist by theorem 1, though they are not produced by $\mathsf{ACC}(F, C)$. More precisely, we are interested in the *witness problem* for covered clauses.

Definition 5. *The* witness problem *for a redundancy property P is as follows: given a formula F and a clause C, if P is met by C with respect to F then return a witness for C, or decide that P is not met by C.*

For instance, the witness problem for blocked clauses is solved as follows: test each $l \in C$ to see if l blocks C in F. As soon as a blocking literal l is found then α_l is a witness for C, where $\alpha = \neg C$. If no blocking literal is found, then C is not blocked. For blocked clauses, this polynomial procedure decides whether C is blocked or not and also determines a witness $\omega = \alpha_l$ for C.

Solving the witness problem for covered clauses is not as straightforward, as it is not clear how a witness could be produced when deciding a clause is covered, or from a sequence σ constructed by $\mathsf{ACC}(F, C)$. The following theorem shows that this problem is as difficult as producing a satisfying assignment for an arbitrary formula, if one exists. In particular, we present a polynomial time reduction from the search analog of the SAT problem: given a formula F, return a satisfying assignment of F, or decide that F is unsatisfiable.

Specifically, given a formula G, we construct a pair (F, C) as an instance to the witness problem for covered clauses. In this construction, C is covered in F and has some witness ω. Moreover, any witness ω for this C necessarily provides a satisfying assignment to G, if there is one.

[2] In order to simplify the presentation, only the part of the witness differing from the negated clause is written, so that $(l \wr C)$ actually stands for $(\neg C_l : C)$. The former is in essence the original notation used in [21], while set, super or globally blocked, as well as PR clauses [15,22,23] require the more general one used in this paper.

Proposition 2. *Given a formula $G = D_1 \wedge \cdots \wedge D_n$, let $G' = D'_1 \wedge \cdots \wedge D'_n$ refer to a variable-renamed copy of G, containing v' everywhere G contains v, so that $var(G) \cap var(G') = \emptyset$. Further, let $C = k \vee l$ and construct the formula:*

$$\begin{aligned} F \;=\; & (x \vee \neg k) \wedge (\neg x \;\vee \neg y) \wedge (y \vee \neg l) \;\wedge \\ & (x \vee D_1) \wedge \quad \cdots \quad \wedge (x \vee D_n) \wedge \\ & (y \vee D'_1) \wedge \quad \cdots \quad \wedge (y \vee D'_n) \end{aligned}$$

for variables $x, y, k, l \notin var(G) \cup var(G)'$. Finally, let ω be a witness for C with respect to F. Either ω satisfies at least one of G or G', or G is unsatisfiable.

Proof. First notice for C that x is covered by k and y is covered by l, so that the extension $(k \vee l \vee x \vee y)$ is blocked in F (with blocking literal x or y). Thus C is redundant in F, so a witness ω exists.

We show that ω satisfies G or G' if and only if G is satisfiable.

($\Rightarrow$) If ω satisfies G then surely G is satisfiable. If ω satisfies G' but not G then the assignment $\omega_G = \{x \in var(G) \mid x' \in G' \text{ and } x \in \omega\}$ satisfies G.

($\Leftarrow$) Assume G is satisfiable, and without loss of generality[3] further assume $\omega = \{k\} \circ \omega'$ for some ω' not assigning $var(k)$. Then $F|_\alpha \vDash (F|_{\{k\}})|_{\omega'}$; that is,

$$F|_\alpha \;\vDash\; \big((x) \wedge (\neg x \vee \neg y) \wedge (y \vee \neg l) \wedge (x \vee D_1) \wedge \cdots \wedge (y \vee D'_n)\big)|_{\omega'}.$$

G is satisfiable, so there are models of $F|_\alpha$ in which $\neg x$ is true. However, x occurs as a unit clause in $F|_{\{k\}}$, so it must be the case that $x \in \omega'$. Therefore $\omega = \{k, x\} \circ \omega''$ for some ω'' assigning neither $var(k)$ nor $var(x)$ such that

$$F|_\alpha \;\vDash\; \big((\neg y) \wedge (y \vee \neg l) \wedge (y \vee D'_1) \wedge \cdots \wedge (y \wedge D'_n)\big)|_{\omega''}.$$

By similar reasoning, ω'' must assign y to false, so now $\omega = \{k, x, \neg y\} \circ \omega'''$ for some ω''', assigning none of $var(k)$, $var(x)$, or $var(y)$, such that

$$F|_\alpha \;\vDash\; \big((\neg l) \wedge (D'_1) \wedge \cdots \wedge (D'_n)\big)|_{\omega'''}.$$

Finally, consider any clause $D'_i \in G'$. We show that ω satisfies D'_i. As ω is a witness, $F|_\alpha \vDash F|_\omega$, so that $(D'_i)|_\omega$ is true in all models of $F|_\alpha$, including models which assign y to true. In particular, let τ be a model of G; then $(D'_i)|_\omega$ is satisfied by $\tau \cup \{\neg x, y\} \cup \nu$, for every assignment ν over $var(G')$. Because $var(D'_i) \subseteq var(G')$, then $(D'_i)|_\omega \equiv \top$. Therefore $G'|_\omega \equiv \top$. □

Proposition 2 suggests there is likely no polynomial procedure for computing witnesses for covered clauses. The existence of witnesses is the basis for solution reconstruction, but witnesses which cannot be efficiently computed make the use of non-model-preserving clause elimination procedures more challenging; that is, we are not aware of any polynomial algorithm for generating a compact (sub-quadratic) reconstruction sequence (see also example 2).

[3] If $k \notin \omega$, then $\omega = \{l\} \circ \omega'$ for some ω' not assigning $var(l)$ and the argument is symmetric, ending with $G|_\omega = \top$. Note that by definition ω satisfies C thus k or l.

As PR clauses are defined by witnesses, procedures deciding PR generally solve the witness problem for PR. For example, the PR reduct [16] provides a formula whose satisfying assignments encode PR witnesses, if they exist. However, this does not produce witnesses for covered clauses, which are not encompassed by PR. In other words, although any clause extended by a single covered literal addition is a PR clause by proposition 1, this is not true for covered clauses.

Theorem 3. *Covered clauses are not all propagation redundant.*

Proof. By counterexample. Consider the clause $C = k \vee l$ and the formula

$$\begin{aligned} F \;=\; & (x \vee \neg k) \;\wedge\; (\neg x \vee \neg y) \;\wedge (y \vee \neg l) \;\wedge \\ & (x \vee a \vee b) \wedge (x \vee a \vee \neg b) \wedge (x \vee \neg a \vee b) \wedge (x \vee \neg a \vee \neg b) \wedge \\ & (y \vee c \vee d) \wedge (y \vee c \vee \neg d) \wedge (y \vee \neg c \vee d) \wedge (y \vee \neg c \vee \neg d). \end{aligned}$$

The extension $C \vee x \vee y$ is blocked with respect to F, so C is covered. However, C is not PR with respect to F. To see this, suppose to the contrary that ω is a PR witness for C. Similar to the reasoning in the proof of theorem 2, assume, without loss of generality, that $\omega = \{k\} \circ \omega'$ for some ω' not assigning k. Notice that $(x) \in F|_k$, but unit propagation on $\neg x \wedge F|_\alpha$ stops without producing $\bot$. Therefore $x \in \omega'$, and $\omega = \{k, x\} \circ \omega''$ for some ω'' assigning neither k nor x. By similar reasoning, it must be the case that $\neg y \in \omega''$, so that $\omega = \{k, x, \neg y\} \circ \omega'''$. Now, $(c \vee d) \in F|_{\{k,x,\neg y\}}$, but once more, unit propagation on $F|_\alpha \wedge \neg c \wedge \neg d$ does not produce $\bot$, so either c or d belongs to ω'''. Without loss of generality, assume $c \in \omega'''$ so that $\omega = \{k, x, \neg y, c\} \circ \omega''''$. Finally, both d and $\neg d$ are clauses in $F'|_{\{k,x,\neg y,c\}}$, but neither are implied by $F|_\alpha$ by unit propagation. However, if either d or $\neg d$ belongs to ω, then $\bot \in F|_\omega$. As unit propagation on $F|_\alpha$ alone does not produce $\bot$, this is a contradiction. □

Notice the formula in theorem 3 can be seen as an instance of the formula in proposition 2, with G as $(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b)$. In fact, as long as unit propagation on G does not derive $\bot$, then G could be any, arbitrarily hard, unsatisfiable formula (such as an instance of the pigeonhole principle).

5 Complexity of Redundancy

In the previous section we introduced the witness problem for a redundancy property (definition 5) and showed that it is not trivial, even when the redundancy property itself can be efficiently decided. Further, the witness problem for PR clauses is solvable by encoding it into SAT [16].

Note that PR is considered to be a very general redundancy property. The proof of theorem 1 in [15,17] shows that if F is satisfiable and C redundant, then $F \wedge C$ is satisfiable by definition. In addition, any satisfying assignment τ of $F \wedge C$ is a PR witness for C with respect to F. This yields the following:

Proposition 3. *Let F be a satisfiable formula. A clause C is redundant with respect to F if and only if it is a* PR *clause with respect to F.*

While not all covered clauses are PR, this motivates the question of whether witnesses for all redundancy properties can be encoded as an instance to SAT, and solved similarly. In this section we show that this is likely not the case by demonstrating the complexity of the *redundancy problem*: given a clause C and a formula F, is C redundant with respect to F?

Deciding whether a clause is PR belongs to NP: assignments can be chosen non-deterministically and efficiently verified as PR witnesses, since the relation $\vdash_1$ is polynomially decidable [18]. For clause redundancy in general, it is not clear that this holds, as the corresponding problem is co-NP-complete.

Proposition 4. *Deciding whether an assignment ω is a witness for a clause C with respect to a formula F is complete for* co-NP*.*

Proof. The problem belongs to co-NP since $F|_\alpha \vDash F|_\omega$ whenever $\neg(F|_\alpha) \vee F|_\omega$ is a tautology. In the following we show a reduction from the tautology problem. Given a formula F, construct the formula F' as below, for $x \notin \mathit{var}(F)$. Further, let $C' = x$, so that $\alpha = \neg x$, and let also $\omega = x$.

$$F' = \bigwedge_{C \in F} (C \vee \neg x)$$

Then $F'|_\alpha = \top$ and $F'|_\omega = F$. Therefore $F'|_\alpha \vDash F'|_\omega$ if and only if $\top \vDash F$. □

Theorem 4 below shows that the *irredundancy problem*, the complement of the redundancy problem, is complete for the class $\mathrm{D^P}$, the class of languages that are the intersection of a language in NP and a language in co-NP [28]:

$$\mathrm{D^P} = \{L_1 \cap L_2 \mid L_1 \in \text{NP and } L_2 \in \text{co-NP}\}.$$

This class was originally introduced to classify certain problems which are hard for both NP and co-NP, but do not seem to be complete for either, and it characterizes a variety of optimization problems. It is the second level of the Boolean hierarchy over NP, which is the completion of NP under Boolean operations [5,29]. We provide a reduction from the canonical $\mathrm{D^P}$-complete problem, SAT-UNSAT: given formulas F and G, is F satisfiable and G unsatisfiable?

Theorem 4. *The irredundancy problem is* $\mathrm{D^P}$*-complete.*

Proof. Notice that the irredundancy problem can be expressed as

$$\begin{aligned}\mathrm{IRR} &= \{\,(F, C) \mid F \text{ is satisfiable, and } F \wedge C \text{ is unsatisfiable}\} \\ &= \{\,(F, C) \mid F \in \mathrm{SAT}\} \cap \{(F, C) \mid F \wedge C \in \mathrm{UNSAT}\}.\end{aligned}$$

That is, IRR is the intersection of a language in NP and a language in co-NP, and so the irredundancy problem belongs to $\mathrm{D^P}$.

Now, let (F, G) be an instance to SAT-UNSAT. Construct the formula F' as follows, for $x \notin \mathit{var}(F) \cup \mathit{var}(G)$:

$$F' = \bigwedge_{C \in F} (C \vee x) \wedge \bigwedge_{D \in G} (D \vee \neg x).$$

Further, let $C' = x$. We demonstrate that $(F, G) \in$ SAT-UNSAT if and only if C' is irredundant with respect to F'.

($\Leftarrow$) Suppose C' is irredundant with respect to F'. In other words, F' is satisfiable but $F' \wedge C'$ is unsatisfiable. Since $F' \wedge C'$ is unsatisfiable, it must be the case that $F'|_{\{x\}}$ is unsatisfiable; however, F' is satisfiable, therefore $F'|_{\{\neg x\}}$ must be satisfiable. Since $F'|_{\{\neg x\}} = F$ and $F'|_{\{x\}} = G$, then $(F, G) \in$ SAT-UNSAT.

($\Rightarrow$) Now, suppose F is satisfiable and G is unsatisfiable. Then some assignment τ over $var(F)$ satisfies F. As a result, $\tau \cup \{\neg x\}$ satisfies F'. Because G is unsatisfiable, there is no assignment satisfying $F'|_{\{x\}} = G$. This means there is no σ satisfying both F' and $C' = x$, and so $F' \wedge C'$ is unsatisfiable as well. Therefore C' is irredundant with respect to F'. □

Consequently the redundancy problem is complete for co-D^{P}. This suggests that sufficient SAT encodings of the clause redundancy problem, and its corresponding witness problem, are not possible.

6 Conclusion

We revisit a strong clause elimination procedure, covered clause elimination, and provide an explicit algorithm for both deciding its redundancy property and reconstructing solutions after its use. Covered clause elimination is unique in that it does not produce redundancy witnesses for clauses it eliminates, and uses a complex, multi-step reconstruction strategy. We prove that while witnesses exist for covered clauses, computing such a witness is as hard as finding a satisfying assignment for an arbitrary formula.

For PR, a very general redundancy property used by strong proof systems, witnesses can be found through encodings into SAT. We show that covered clauses are not described by PR, and SAT encodings for finding general redundancy witnesses likely do not exist, as deciding clause redundancy is hard for the class D^{P}, the second level of the Boolean hierarchy over NP.

Directions for future work include the development of redundancy properties beyond PR, and investigating their use for solution reconstruction after clause elimination, as well as in proof systems. Extending redundancy notions by using a structure for witnesses other than partial assignments may provide more generality while remaining polynomially verifiable.

We are also interested in developing notions of redundancy for adding or removing more than a single clause at a time, and exploring proof systems and simplification techniques which make use of non-clausal redundancy properties.

References

1. Ajtai, M.: The complexity of the pigeonhole principle. Combinatorica **14**(4), 417–433 (1994)

2. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT competition 2018. In: Heule, M.J.H., Järvisalo, M., Suda, M. (eds.) Proceedings of SAT Competition 2018. pp. 13–14. Department of Computer Science Series of Publications B, University of Helsinki (2018)
3. Biere, A.: CaDiCaL at the SAT race 2019. In: Heule, M.J.H., Järvisalo, M., Suda, M. (eds.) Proceedings of SAT Race 2019. pp. 8–9. Department of Computer Science Series of Publications B, University of Helsinki (2019)
4. Buss, S., Thapen, N.: DRAT proofs, propagation redundancy, and extended resolution. In: Janota, M., Lynce, I. (eds.) Theory and Applications of Satisfiability Testing – SAT 2019. LNCS, vol. 11628, pp. 71–89. Springer (2019)
5. Cai, J.Y., Hemachandra, L.: The Boolean hierarchy: hardware over NP. In: Structure in complexity theory, LNCS, vol. 223, pp. 105–124. Springer (1986)
6. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design **19**(1), 7–34 (2001)
7. Fazekas, K., Biere, A., Scholl, C.: Incremental inprocessing in SAT solving. In: Janota, M., Lynce, I. (eds.) Theory and Applications of Satisfiability Testing – SAT 2019. LNCS, vol. 11628, pp. 136–154. Springer (2019)
8. Gableske, O.: BossLS preprocessing and stochastic local search. In: Balint, A., Belov, A., Diepold, D., Gerber, S., Järvisalo, M., Sinz, C. (eds.) Proceedings of SAT Challenge 2012. pp. 10–11. Department of Computer Science Series of Publications B, University of Helsinki (2012)
9. Haken, A.: The intractability of resolution. Theoretical Computer Science **39**(2-3), 297–308 (1985)
10. Heule, M.J.H.: Schur number five. In: Proc. of the 32nd AAAI Conference on Artificial Intelligence, (AAAI-18). pp. 6598–6606 (2018)
11. Heule, M.J.H., Biere, A.: What a difference a variable makes. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems – TACAS 2018. LNCS, vol. 10806, pp. 75–92. Springer (2018)
12. Heule, M.J.H., Järvisalo, M., Biere, A.: Clause elimination procedures for CNF formulas. In: Fermüller, C.G., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning – LPAR 17. LNCS, vol. 6397, pp. 357–371. Springer (2010)
13. Heule, M.J.H., Järvisalo, M., Biere, A.: Covered clause elimination. In: Voronkov, A., Sutcliffe, G., Baaz, M., Fermüller, C. (eds.) Logic for Programming, Artificial Intelligence and Reasoning – LPAR 17 (short paper). EPiC Series in Computing, vol. 13, pp. 41–46. EasyChair (2010)
14. Heule, M.J.H., Järvisalo, M., Lonsing, F., Seidl, M., Biere, A.: Clause elimination for SAT and QSAT. Journal of Artificial Intelligence Research **53**(1), 127–168 (2015)
15. Heule, M.J.H., Kiesl, B., Biere, A.: Short proofs without new variables. In: de Moura, L. (ed.) Automated Deduction – CADE 26. LNCS, vol. 10395, pp. 130–147. Springer (2017)
16. Heule, M.J.H., Kiesl, B., Biere, A.: Encoding redundancy for satisfaction-driven clause learning. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference – TACAS 2019. LNCS, vol. 11427, pp. 41–58. Springer (2019)
17. Heule, M.J.H., Kiesl, B., Biere, A.: Strong extension-free proof systems. Journal of Automated Reasoning **64**, 533–554 (2020)
18. Heule, M.J.H., Kiesl, B., Seidl, M., Biere, A.: PRuning through satisfaction. In: Strichman, O., Tzoref-Brill, R. (eds.) Haifa Verification Conference – HVC 2017. LNCS, vol. 10629, pp. 179–194. Springer (2017)

19. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In: Creignou, N., Le Berre, D. (eds.) Theory and Applications of Satisfiability Testing – SAT 2016. LNCS, vol. 9710, pp. 228–245. Springer (2016)
20. Järvisalo, M., Biere, A., Heule, M.J.H.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) Tools and Algorithms for the Construction and Analysis of Systems – TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer (2010)
21. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) International Joint Conference on Automated Reasoning – IJCAR 2012. pp. 355–370. Springer (2012)
22. Kiesl, B., Heule, M.J.H., Biere, A.: Truth assignments as conditional autarkies. In: Chen, Y., Cheng, C., Esparza, J. (eds.) Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11781, pp. 48–64. Springer (2019)
23. Kiesl, B., Seidl, M., Tompits, H., Biere, A.: Super-blocked clauses. In: Olivetti, N., Tiwari, A. (eds.) International Joint Conference on Automated Reasoning – IJCAR 2016. LNCS, vol. 9706, pp. 45–61. Springer (2016)
24. Konev, B., Lisitsa, A.: Computer-aided proof of Erdős discrepancy properties. Artificial Intelligence **224**, 103–118 (2015)
25. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics **96/97**, 149–176 (1999)
26. Manthey, N.: Coprocessor 2.0 – a flexible CNF simplifier. In: Cimatti, A., Sebastiani, R. (eds.) Theory and Applications of Satisfiability Testing – SAT 2012. LNCS, vol. 7317, pp. 436–441. Springer (2012)
27. Mironov, I., Zhang, L.: Applications of SAT solvers to cryptanalysis of hash functions. In: Biere, A., Gomes, C.P. (eds.) Theory and Applications of Satisfiability Testing – SAT 2006. pp. 102–115. Springer (2006)
28. Papadimitriou, C., Yannakakis, M.: The complexity of facets (and some facets of complexity). Journal of Computer and System Sciences **28**(2), 244–259 (1984)
29. Wechsung, G.: On the boolean closure of NP. In: Budach, L. (ed.) Fundamentals of Computation Theory – FCT 1985. LNCS, vol. 199, pp. 485–493. Springer (1985)