Truth Assignments as Conditional Autarkies

Benjamin Kiesl[1], Marijn J.H. Heule[2], and Armin Biere[3]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
[2] Computer Science Department, CMU, Pittsburgh, United States
[3] Institute for Formal Models and Verification, JKU Linz, Austria

Abstract. An autarky for a formula in propositional logic is a truth assignment that satisfies every clause it touches, i.e., every clause for which the autarky assigns at least one variable. In this paper, we present how conditional autarkies, a generalization of autarkies, give rise to novel preprocessing techniques for SAT solving. We show that conditional autarkies correspond to a new type of redundant clauses, termed globally-blocked clauses, and that the elimination of these clauses can simulate existing circuit-simplification techniques on the CNF level.

1 Introduction

Satisfiability (SAT) solvers have been successfully used for a broad spectrum of applications ranging from formal verification [1] over security [2] to classical mathematics [3,4]. This success came as a slight surprise because the translation of problem instances from application domains into propositional logic can lead to a loss of domain-specific information. However, this loss of information is often harmless since many domain-specific reasoning techniques (e.g., for Boolean circuits or number theory) can be simulated by SAT-preprocessing techniques such as blocked-clause elimination [5]. In this paper, we present further evidence of this observation by introducing a new propositional reasoning technique that simulates the removal of redundant inputs from Boolean circuits.

Our reasoning technique, which we call *globally-blocked-clause elimination*, is strongly related to the concept of *conditional autarkies* [6]—a generalization of *autarkies* [7,8]. Given a propositional formula in conjunctive normal form, an *autarky* is a truth assignment that satisfies every clause it touches, that is, it satisfies every clause of which it assigns at least one variable. For example, given the formula $(\overline{a} \vee b) \wedge (a \vee \overline{b} \vee \overline{c} \vee d) \wedge (c \vee \overline{d})$, the (partial) assignment that makes both a and b true is an autarky. In contrast, neither the assignment that makes only a true nor the assignment that makes only b true are autarkies because they touch clauses without satisfying them.

A *conditional autarky* is an assignment that can be split into two parts—the *conditional part* and the *autarky part*—such that the autarky part becomes an autarky after applying the conditional part to the formula. For example, after making a true in the formula $(a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b \vee \overline{d}) \wedge (\overline{a} \vee \overline{b} \vee c) \wedge (\overline{a} \vee d)$ we obtain the formula $(b \vee \overline{d}) \wedge (\overline{b} \vee c) \wedge (d)$ for which the assignment that makes b and c true is an autarky. Hence, the assignment that makes a, b, and c true is a

conditional autarky with conditional part a although it is not an autarky. In fact, every truth assignment is a conditional autarky with an empty autarky part, but we are particularly interested in conditional autarkies with *non-empty* autarky parts. We show that such conditional autarkies help us find redundant clauses (i.e., clauses that can be removed from a formula without affecting satisfiability). More specifically, we present *globally-blocked clauses*, a novel type of redundant clauses that is strongly related to the conditional-autarky concept.

Globally-blocked clauses are a strict generalization of set-blocked clauses [9], which themselves are a strict generalization of blocked clauses [10]. This means that every set-blocked clause (and thus every blocked clause) is a globally-blocked clause but not vice versa. The elimination of blocked clauses can improve the efficiency of SAT solvers [11], first-order theorem provers [12], and solvers for quantified Boolean formulas (QBF) [13,14]. Moreover, it can simulate several reasoning techniques for Boolean circuits on the propositional level. Set-blocked clauses form the basis of the *satisfaction-driven clause learning* [6] paradigm for SAT solving, which can lead to exponential speed-ups on hard formulas compared to traditional *conflict-driven clause learning* [15].

We show how the elimination of globally-blocked clauses simulates circuit-reasoning techniques that could not be performed by SAT solvers so far. In a preprocessing step for the actual solving, our approach takes a formula and tries to find conditional autarkies with large autarky parts. It then uses the resulting conditional autarkies to identify and eliminate globally-blocked clauses from the formula, which results in a simplified formula that is easier to solve. In a more theoretic part of the paper, we present several properties of conditional autarkies and pin down their relationship with globally-blocked clauses and other existing types of redundant clauses.

The main purpose of this invited paper is to discuss the concept of globally-blocked clauses and relate it to conditional autarkies [6]. This concept was partially described in section 2.2.4 of the PhD thesis of the first author [16]. In this paper, we provide more details and also describe an algorithm for fast computation of multiple globally-blocked clauses from one arbitrary total assignment. This algorithm has also been implemented in the SAT solver CaDiCaL. The second purpose of this paper is to describe the history and applications of such notions of redundancy as well as how these are used in clausal proofs.

2 Preprocessing, Redundancy, and Proofs

Preprocessing (simplifying a formula before search) and inprocessing (simplifying a formula during search) [17] are crucial techniques in state-of-the-art SAT solvers. This line of research started with *bounded variable elimination*, which allows to significantly reduce the size of large industrial verification problems [18]. Bounded variable elimination removes variables from a formula—by combining the variable elimination with substitution—if this removal does not increase the size of the formula. More recently, some solvers even allow a small increase of the formula size.

Another popular approach to preprocessing and inprocessing are so-called *clause-elimination* techniques, which remove clauses from a formula without affecting satisfiability. Two particularly well-known clause-elimination techniques are *subsumed-clause elimination* [18] and *blocked-clause elimination* [11]. Blocked-clause elimination simulates many circuit simplification techniques on the CNF level and it allows the simulation of other high-level reasoning techniques, such as the elimination of redundant Pythagorean triples, which was crucial to solving the Pythagorean triples problem [19]. Several generalizations of subsumed clauses and blocked clauses have been proposed to further reduce the size of propositional formulas [20,21,22]. Moreover, clause-elimination techniques boost the effectiveness of variable-elimination techniques since the removal of clauses often enables further variable-elimination steps.

Although preprocessing techniques have contributed significantly to the improvement of SAT solvers in the last decade, they can also be expensive. To deal with this issue, SAT solvers have shifted their focus from preprocessing to inprocessing, meaning that only limited preprocessing is done initially and that later on the solver interleaves additional variable and clause-elimination techniques with the search process. One advantage of this is that inprocessing can simplify a formula after important clauses have been learned, allowing for further simplifications compared to preprocessing. As a matter of fact, inprocessing solvers have been dominating the SAT competitions since 2013.

As a drawback, the incorporation of inprocessing has made SAT solvers more complex and thus more prone to implementation errors and conceptual errors [17]. Various communities that use SAT solvers have therefore expressed interest in verifiable SAT solver output. For example, SAT solvers are used in industry to show the correctness of safety-critical hardware and software, or in mathematics to solve long-standing open problems. In both cases, it is crucial that a SAT solver not just returns a simple yes/no answer but instead produces verifiable output that certifies the correctness of its result—a so-called proof.

Constructing such proofs is a non-trivial issue as several inprocessing techniques cannot be expressed succinctly in the resolution proof system, which was commonly used in the past. This led to the search for stronger proof systems that are well-suited for practical SAT solving, and it turned out that clause-redundancy notions that form the theoretical foundation of clause-elimination techniques can also act as ideal building blocks for stronger proof systems. The DRAT proof system [23], which is based on the notion of *resolution asymmetric tautologies*, has since become the de-facto standard proof system in SAT solving. DRAT is now supported by all state-of-the-art SAT solvers and there exist formally-verified tools—in ACL2, Coq, and Isabelle [24,25]—that check the correctness of DRAT proofs. Such tools have not only been used for validating the unsatisfiability results of recent SAT competitions [26] but also for verifying the solutions of some long-standing math problems, including the Erdős discrepancy conjecture, the Pythagorean triples problem, and Schur number five [3,19,4].

To strengthen the DRAT proof system even further, proof systems based on stronger redundancy notions than resolution asymmetric tautologies have

been proposed, leading to the *Propagation Redundancy* (PR) proof system [27]. This proof system is surprisingly strong even without the introduction of new variables, which usually is a key technique to obtaining short proofs. As has been shown, there exist short PR proofs without new variables for pigeon hole formulas, Tseitin formulas, and mutilated chessboard problems [28,29]—these problem families are notorious for admitting only resolution proofs of exponential size. Moreover, the PR proofs for these problems can be found automatically using the *satisfaction-driven clause learning* paradigm (SDCL) [6,30], which is a generalization of *conflict-driven clause learning*.

As DRAT has been shown to polynomially simulate the PR [28] proof system, it is possible to transform PR proofs into DRAT proofs and then check their correctness using a formally-verified checker, meaning that one can have high confidence in the correctness of results obtained by SDCL.

Research on preprocessing, clause redundancy, and proofs has also expanded beyond propositional logic. When solving quantified Boolean formulas, QBF generalizations of blocked-clause elimination have been used successfully [13]. Also, the QRAT proof system [31] (a generalization of DRAT) allows the succinct expression of virtually all QBF inprocessing techniques, and QRAT has given rise to various new QBF preprocessing techniques, such as the elimination of blocked literals [32] as well as of QRAT clauses and their generalizations [33,34]. The research on generalizing redundancy notions has also been extended to first-order logic [35], where especially the elimination of blocked clauses has proven to be a valuable preprocessing technique [12].

In the following, we present globally-blocked clauses, a new kind of redundant clauses that generalizes existing notions of redundancy in propositional logic.

3 Motivating Example

Consider a single-output circuit $F(I)$ (where I is a set of inputs) that can be decomposed syntactically into $F(I) = G(J, H(K))$, where both G and H are single output sub-circuits, and J and K partition the inputs I. If we want to solve the satisfiability problem for F, i.e., the problem of deciding if there exists a set of inputs such that F produces a 1 as its output (CIRCUIT-SAT), we can proceed as follows: We show that $G(J, x)$ is satisfiable, where x is a new variable, and that $H(K)$ can produce both 0 and 1 as its output. In many situations, if the sub-circuit H is given, the second requirement can be shown easily using random simulation. Checking satisfiability of G remains to be shown, but is hopefully easier, since G is smaller than F.

This circuit-level technique is also called "unique sensitization" in the FAN algorithm [36] and H would be called a "headline". However, if we are only given a CNF encoding of F, previously known CNF preprocessing techniques are in general not able to perform such a simplification, whereas globally-blocked-clause elimination in essence allows to remove all clauses of the CNF encoding of H.

To continue the example, assume for simplicity that the top-level gate of H is an AND gate (the same arguments apply to arbitrary top gates of H). After

introducing Tseitin variables x for H, y and z for the AND gate inputs, etc., the Tseitin encoding F' of F has the following structure

$$\begin{aligned} F'(I,S,T,x,y,z) &= G'(J,S,x) \wedge H'(K,x,y,z,T) && \text{with} \\ H'(K,x,y,z,T) &= \underbrace{(\overline{x} \vee y) \wedge (\overline{x} \vee z) \wedge (x \vee \overline{y} \vee \overline{z})}_{\text{Tseitin encoding of top AND gate in } H} \wedge H''(K,y,z,T) \end{aligned}$$

where $S \cup T \cup \{x,y,z\}$ are new Tseitin variables. Note that F', G', H' and H'' are in CNF. Further assume we find two assignments α and β over the variables of H', which both satisfy H', i.e., $\alpha(H') = \beta(H') = 1$, and $\alpha(x) = 1$, $\beta(x) = 0$.

It is not that hard to find such assignments through local search or random decisions and unit propagation. Actually, one can also start with a total assignment with these properties, which will then—by our algorithms—be pruned down to range only over variables in H'. These assignments are conditional autarkies where the conditional part consists of the assignment to x and the autarky part consists of the assignments to the other variables of H'.

It turns out that the first two binary clauses encoding the top AND gate of H contain the negation $\overline{x}$ of the condition in α, and the last ternary clause of the AND gate contains the negation x of the condition in β. Moreover, these three clauses are satisfied by the autarky parts of α and β. As we are going to prove, this situation allows to deduce that the clauses are globally blocked and thus redundant. After removing the three AND gate clauses, both conditional autarkies α and β become autarkies, allowing to remove H' too.

Alternatively, blocked-clause elimination [11] or variable elimination [18] would also remove H', since after removing the clauses of the top gate of H *cone-of-influence reduction* applies, which is simulated by both techniques [5]. Thus, for this example the key aspect of globally-blocked-clause elimination is that it allows to the remove the clauses of the headline gate connecting the two parts of the CNF, in fact simulating unique sensitization on the CNF level.

4 Preliminaries

Here, we present the background necessary for understanding the rest of the paper. We consider propositional formulas in conjunctive normal form (CNF), which are made up of variables, literals, and clauses, as defined in the following. A *literal* is either a variable x (a *positive literal*) or the negation $\overline{x}$ of a variable x (a *negative literal*). The *complement* $\overline{l}$ of a literal l is defined as $\overline{l} = \overline{x}$ if $l = x$ and as $\overline{l} = x$ if $l = \overline{x}$. For a literal l, we denote the variable of l by $var(l)$. A *clause* is a finite disjunction of the form $(l_1 \vee \cdots \vee l_n)$ where $l_1, \ldots, l_n$ are literals. A *tautology* is a clause that contains both a literal and its complement. If not stated otherwise, we assume that clauses are not tautologies. A *formula* is a finite conjunction of the form $C_1 \wedge \cdots \wedge C_m$ where $C_1, \ldots, C_m$ are clauses. Clauses can be viewed as sets of literals and formulas can be viewed as sets of clauses. For a set L of literals and a formula F, we define $F_L = \{C \in F \mid C \cap L \neq \emptyset\}$. We sometimes write F_l for $F_{\{l\}}$.

A *truth assignment* (or short, *assignment*) is a function from a set of variables to the truth values 1 (*true*) and 0 (*false*). An assignment is *total* with respect to a formula if it assigns a truth value to all variables occurring in the formula. If not stated otherwise, we do not require assignments to be total. We denote the domain of an assignment α by $var(\alpha)$. A literal l is *satisfied* by an assignment α if l is positive and $\alpha(var(l)) = 1$ or if it is negative and $\alpha(var(l)) = 0$. A literal is *falsified* by an assignment if its complement is satisfied by the assignment. An assignment *touches* a clause if it assigns a truth value to at least one of its literals. A clause is satisfied by an assignment α if it contains a literal that is satisfied by α. Finally, a formula is satisfied by an assignment α if all its clauses are satisfied by α. A formula is *satisfiable* if there exists an assignment that satisfies it. Two formulas are *logically equivalent* if they are satisfied by the same total assignments; they are *satisfiability-equivalent* if they are either both satisfiable or both unsatisfiable. We often view assignments as the sets of literals they satisfy and denote them as sequences of literals. For instance, given an assignment α that makes x true and y false, we would denote α by $x\,\overline{y}$ and write things like $x \in \alpha$.

We denote the empty clause by $\bot$ and the satisfied clause by $\top$. Given an assignment α and a clause C, we define $C\,|\,\alpha = \top$ if α satisfies C, otherwise $C\,|\,\alpha$ denotes the result of removing from C all the literals falsified by α. Moreover, for a formula F, we define $F\,|\,\alpha = \{C\,|\,\alpha \mid C \in F \text{ and } C\,|\,\alpha \neq \top\}$.

We consider a clause to be redundant with respect to a formula if the clause can be removed from the formula without affecting the formula's satisfiability or unsatisfiability:

Definition 1. *A clause C is* redundant *with respect to a formula F if F and $F \wedge C$ are satisfiability-equivalent.*

For instance, the clause $C = (a \vee b)$ is redundant with respect to the formula $F = (\overline{a} \vee \overline{b})$ since F and $F \wedge C$ are satisfiability-equivalent (although they are not logically equivalent).

5 Conditional Autarkies

In the following, we discuss the notions of autarkies and conditional autarkies from the literature. We then present new theoretical results for conditional autarkies and use these results to develop an algorithm that identifies particular conditional autarkies. This section provides the basis for our SAT-preprocessing approach. We start with autarkies (remember that we do not require assignments to be total) [7,8]:

Definition 2. *An assignment α is an* autarky *for a formula F if α satisfies every clause $C \in F$ for which $var(\alpha) \cap var(C) \neq \emptyset$.*

In other words, an autarky satisfies every clause it touches.

Example 3. Let $F = (a \vee b \vee \overline{c}) \wedge (\overline{b} \vee c \vee d) \wedge (\overline{a} \vee \overline{d})$ and let $\alpha = bc$. Then, α touches only the first two clauses. Since it satisfies them, it is an autarky for F.

One crucial property of autarkies, which follows easily from the definition, is that they can be applied to a formula without affecting the formula's satisfiability:

Theorem 4. *If an assignment α is an autarky for a formula F, then F and $F|\alpha$ are satisfiability-equivalent.*

Theorem 4 can be viewed as follows: If $\alpha = l_1 \dots l_n$ is an autarky for F, then F and $F \wedge (l_1) \wedge \dots \wedge (l_n)$ are satisfiability-equivalent. This view is useful in the context of *conditional autarkies* [6]. Informally, a conditional autarky is an assignment that can be partitioned into two parts such that one part becomes an autarky as soon as the other part has been applied to the formula:

Definition 5. *An assignment $\alpha_{\mathsf{c}} \cup \alpha_{\mathsf{a}}$ (with $\alpha_{\mathsf{c}} \cap \alpha_{\mathsf{a}} = \emptyset$) is a* conditional autarky *for a formula F if α_{a} is an autarky for $F|\alpha_{\mathsf{c}}$.*

We call α_{c} the *conditional part* and α_{a} the *autarky part* of $\alpha_{\mathsf{c}} \cup \alpha_{\mathsf{a}}$. Observe that every assignment is a conditional autarky with an empty autarky part. We are mainly interested in conditional autarkies with *non-empty* autarky parts:

Example 6. Consider the formula $F = (a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b \vee \overline{d}) \wedge (\overline{a} \vee \overline{b} \vee c) \wedge (\overline{a} \vee d)$ and the assignments $\alpha_{\mathsf{c}} = a$ and $\alpha_{\mathsf{a}} = bc$. The assignment $\alpha_{\mathsf{c}} \cup \alpha_{\mathsf{a}}$ is a conditional autarky for F since α_{a} is an autarky for $F|\alpha_{\mathsf{c}} = (b \vee \overline{d}) \wedge (\overline{b} \vee c) \wedge (d)$. Notice that neither α_{a} alone nor abc (or any subset) are autarkies for F.

Theorem 4 above tells us that the application of an autarky to a formula does not affect the formula's satisfiability. The following statement, which is a simple consequence of Theorem 4 and the fact that α_{a} is an autarky for $F|\alpha_{\mathsf{c}}$, generalizes this statement for conditional autarkies:

Theorem 7. *Let F be a formula and $\alpha_{\mathsf{c}} \cup \alpha_{\mathsf{a}}$ a conditional autarky for F with conditional part α_{c} and autarky part α_{a}. Then, $F|\alpha_{\mathsf{c}}$ and $F|\alpha_{\mathsf{a}} \cup \alpha_{\mathsf{c}}$ are satisfiability-equivalent.*

As for ordinary autarkies, where we can add all unit clauses $l \in \alpha$ of an autarky α to a formula F, we get a similar result for conditional autarkies:

Given a conditional autarky $c_1 \dots c_m a_1 \dots a_n$ (with conditional part $c_1 \dots c_m$) for a formula F, we can safely add to F the clause form of the implication

$$c_1 \wedge \dots \wedge c_m \;\rightarrow\; a_1 \wedge \dots \wedge a_n.$$

This will later allow us to prove the redundancy of globally-blocked clauses:

Theorem 8. *Let $c_1 \dots c_m\, a_1 \dots a_n$ be a conditional autarky (with conditional part $c_1 \dots c_m$) for a formula F. Then, F and $F \wedge \bigwedge_{1 \le i \le n} (\overline{c}_1 \vee \dots \vee \overline{c}_m \vee a_i)$ are satisfiability-equivalent.*

Proof. We have to show that the satisfiability of F implies the satisfiability of $F \wedge \bigwedge_{1 \leq i \leq n} (\overline{c}_1 \vee \cdots \vee \overline{c}_m \vee a_i)$. Assume that F is satisfiable and let τ be a satisfying assignment of F. If τ falsifies one of the literals $c_1, \ldots, c_m$, the statement trivially holds. Assume thus that τ satisfies all of $c_1, \ldots, c_m$ and define $\tau'(a_i) = 1$ for $1 \leq i \leq n$ and $\tau'(l) = \tau(l)$ for each remaining literal l. Since $c_1 \ldots c_m\, a_1 \ldots a_n$ is a conditional autarky for F with conditional part $c_1 \ldots c_m$, we know that $a_1 \ldots a_n$ is an autarky for $F|_{c_1 \ldots c_m}$. Hence, since τ satisfies F and all of $c_1 \ldots c_m$, the clauses that were affected by making $a_1, \ldots, a_n$ true must also be satisfied by τ'. We conclude that τ' satisfies $F \wedge \bigwedge_{1 \leq i \leq n} (\overline{c}_1 \vee \cdots \vee \overline{c}_m \vee a_i)$.

We already mentioned that we are interested in conditional autarkies with non-empty autarky parts. In fact, for our preprocessing approach, we try to find the smallest conditional parts (and thus the largest autarky parts) for given assignments. As we show next, the smallest conditional part of a given assignment is unique and we can find it efficiently. For this, we need the notion of the *least* conditional part of an assignment:

Definition 9. *Given an assignment α and a formula F, the* least conditional part *of α on F is the assignment α_c such that (1) α is a conditional autarky for F with conditional part α_c and (2) for all assignments α'_c such that α is a conditional autarky for F with conditional part α'_c, it holds that $\alpha_c \subseteq \alpha'_c$.*

The least conditional part of an assignment is unique. To see this, assume α_1 and α_2 are least conditional parts for α on a formula F. Then, $\alpha_1 \subseteq \alpha_2$ and $\alpha_2 \subseteq \alpha_1$ and thus $\alpha_1 = \alpha_2$.

The algorithm LeastConditionalPart in Fig. 1 computes the least conditional part of a given assignment for a formula. In a greedy fashion, the algorithm iterates over all clauses of the formula and whenever it encouters a clause that is touched but not satisfied by the given assignment, it adds all the touched literals of the clause to the conditional part.

```
   LeastConditionalPart(assignment α, formula F)
1    α_c := ∅
2    for C ∈ F do
3      if α touches C without satisfying C then
4        α_c := α_c ∪ (α ∩ C̄)
5    return α_c
```

Fig. 1. Compute the least conditional part of an assignment.

Theorem 10. *Let $\alpha_c = \mathsf{LeastConditionalPart}(\alpha, F)$ given a CNF formula F and an assignment α. Then, α_c is the least conditional part of α on F.*

Proof. Clearly, α is a conditional autarky for F with conditional part α_c: Whenever α touches a clause without satisfying it, all the touched literals are added to α_c (in line 4). Thus a clause in $F|_{\alpha_c}$ touched by $\alpha \setminus \alpha_c$ is also satisfied by it.

It remains to show that for every assignment α'_{c} such that α is a conditional autarky with conditional part α'_{c}, it holds that $\alpha_{\mathsf{c}} \subseteq \alpha'_{\mathsf{c}}$. Let $l \in \alpha_{\mathsf{c}}$. Then, l occurs in a clause C that is touched but not satisfied by α. Now, assume that l is not contained in α'_c. It follows that $l \in \alpha \setminus \alpha'_c$. But then $\alpha \setminus \alpha'_c$ touches $C|\alpha'_{\mathsf{c}}$ without satisfying it and so it is not an autarky for $F|\alpha'_{\mathsf{c}}$. It follows that $\alpha_{\mathsf{c}} \subseteq \alpha'_{\mathsf{c}}$.

6 Globally-Blocked Clauses

We now have an algorithm that identifies the least conditional part of a given conditional autarky. In the following, we use this algorithm to find redundant clauses in a propositional formula. To this end, we introduce *globally-blocked clauses*—a type of redundant clauses that generalizes the existing notion of *blocked clauses* [10] (note that in our notation, the set operators have precedence over logical operators, i.e., $D \setminus \{\bar{l}\} \vee C$ means $(D \setminus \{\bar{l}\}) \vee C$):

Definition 11. *A clause C is* blocked *by a literal $l \in C$ in a formula F if for every clause $D \in F_{\bar{l}}$, the clause $D \setminus \{\bar{l}\} \vee C$ is a tautology.*

Example 12. Let $F = (a \vee b) \wedge (\overline{a} \vee c) \wedge (\overline{b} \vee \overline{a})$ and $C = a \vee b$. The literal b blocks C in F since the only clause in $F_{\overline{b}}$ is the clause $D = \overline{b} \vee \overline{a}$, and $D \setminus \{\overline{b}\} \vee C = \overline{a} \vee a \vee b$ is a tautology.

Blocked clauses are redundant clauses and according to [6] and are related to conditional autarkies as follows:

Theorem 13. *A clause $C = c_1 \vee \cdots \vee c_n \vee l$ is blocked by l in F iff the assignment $\overline{c}_1 \dots \overline{c}_n l$ is a conditional autarky (with conditional part $\overline{c}_1 \dots \overline{c}_n$) for F.*

Globally-blocked clauses generalize blocked clauses by not only considering a single literal $l \in C$, but a set L of literals with a common literal:

Definition 14. *A clause C is* globally blocked *by a set L of literals in a formula F if $L \cap C \neq \emptyset$ and all $D \in F_{\overline{L}} \setminus F_L$, the clause $D \setminus \overline{L} \vee C$ is a tautology.*

We say a clause is globally blocked if there exists some set L of literals by which the clause is globally blocked.

Example 15. Consider $F = (a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b \vee \overline{d}) \wedge (\overline{a} \vee \overline{b} \vee c) \wedge (\overline{a} \vee d)$ from Example 6. The clause $C = \overline{a} \vee c$ is globally blocked in F. To see this, consider the set $L = \{b, c\}$ and the formulas $F_{\overline{L}} = (a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b \vee \overline{d}) \wedge (\overline{a} \vee \overline{b} \vee c)$ and $F_L = (\overline{a} \vee b \vee \overline{d}) \wedge (\overline{a} \vee \overline{b} \vee c)$. We then have $F_{\overline{L}} \setminus F_L = (a \vee \overline{b})$. Let $D = (a \vee \overline{b} \vee \overline{c})$ then $D \setminus \overline{L} \vee C = a \vee \overline{a} \vee c$ is a tautology and so C is globally blocked by L in F. Note, C is not blocked in F. In a similar manner $(\overline{a} \vee b)$ is globally blocked.

Remember that we showed in the previous section (Example 6) that *abcd* is a conditional autarky for F with conditional part a and autarky part *bcd*. Now in Example 15, to demonstrate that C is globally blocked, we used the literals of the autarky part as the set L and we could observe that the literal a of the conditional part together with its complement $\overline{a}$ caused the clause $D \setminus \overline{L} \vee C$ to be a tautology. This is a consequence of the following statement, which will help us with finding globally-blocked clauses using conditional autarkies:

Theorem 16. *Let F be a formula, let C be a clause, let L be a set of literals such that $L \cap C \neq \emptyset$, and define the assignments $\alpha_{\mathsf{c}} = \overline{C \setminus L}$ and $\alpha_{\mathsf{a}} = L$. Then, C is globally blocked by L in F if and only if $\alpha_{\mathsf{c}} \cup \alpha_{\mathsf{a}}$ is a conditional autarky (with conditional part α_{c}) for F.*

Proof. For the "only if" direction, assume C is globally blocked by L in F. We show that α_{a} is an autarky for $F|\alpha_{\mathsf{c}}$. Let $D|\alpha_{\mathsf{c}} \in F|\alpha_{\mathsf{c}}$. Then, D is not satisfied by α_{c}. Since α_{c} falsifies exactly the literals of $C \setminus L$, it follows that D cannot contain the complement of a literal in $C \setminus L$. This implies that C cannot contain the complement of a literal in $D \setminus \overline{L}$ and so $D \setminus \overline{L} \vee C$ is not a tautology. But then D cannot be contained in $F_{\overline{L}} \setminus F_L$, meaning that if D is touched by α_{a} (which satisfies exactly the literals of L), D is also satisfied by α_{a}. Hence, since α_{a} assigns only variables that are not assigned by α_{c}, it cannot be the case that α_{a} touches $D|\alpha_{\mathsf{c}}$ without satisfying it. We thus conclude that α_{a} is an autarky for $F|\alpha_{\mathsf{c}}$.

For the "if" direction, assume $\alpha_{\mathsf{c}} \cup \alpha_{\mathsf{a}}$ is a conditional autarky for F with conditional part α_{c}. We show that for every clause $D \in F_{\overline{L}} \setminus F_L$, the clause $D \setminus \overline{L} \vee C$ is a tautology. Let $D \in F_{\overline{L}} \setminus F_L$. Then, D is a clause that is touched but not satisfied by α_{a}. Hence, α_{c} must satisfy a literal l of D, for otherwise $D|\alpha_{\mathsf{c}}$ would be touched but not satisfied by α_{a}. Moreover, since α_{c} assigns no literals of $\overline{L}$, it must actually be the case that $l \in D \setminus \overline{L}$. But then, since α_{c} falsifies only literals of C, it follows that $\bar{l} \in C$ and so $C \vee D \setminus \overline{L}$ is a tautology. It follows that C is globally blocked by L in F.

Before we focus on finding and removing globally-blocked clauses, we have to show that they are indeed redundant:

Theorem 17. *If a clause C is globally blocked in a formula F, it is redundant with respect to F.*

Proof. Assume that C is globally blocked by some set $L = \{l_1, \ldots, l_n\}$ in F and that F is satisfiable. We show that the formula $F \wedge C$ is satisfiable. First, observe that C is of the form $c_1 \vee \ldots c_m \vee l_1 \vee \cdots \vee l_k$ where $\{l_1, \ldots, l_k\} \subseteq L$ and $k \geq 1$. By Theorem 16, we know that the assignment $\alpha_{\mathsf{c}} \cup \alpha_{\mathsf{a}}$ with $\alpha_{\mathsf{c}} = \bar{c}_1 \ldots \bar{c}_m$ and $\alpha_{\mathsf{a}} = l_1 \ldots l_n$ is a conditional autarky (with conditional part α_{c}) for F. Hence, by Theorem 8, F and $F' = F \wedge \bigwedge_{1 \leq i \leq n}(c_1 \vee \cdots \vee c_m \vee l_i)$ are satisfiability-equivalent and so F' must be satisfiable. But then, as C is subsumed by each clause $c_1 \vee \cdots \vee c_m \vee l_i$ with $i \in 1, \ldots, k$, every satisfying assignment of F' must also satisfy $F \wedge C$. It follows that C is redundant with respect to F.

Finally, it can be shown that globally-blocked clauses are a subclass of propagation-redundant clauses (for details, see [27]) but we omit the proof here [16]:

Theorem 18. *If a clause C is globally blocked in a formula F, it is propagation-redundant with respect to F.*

7 Detecting Globally-Blocked Clauses

We have seen that globally-blocked clauses are redundant and that they correspond closely to conditional autarkies. In the next step, we use this correspondence to find globally-blocked clauses in a formula. The idea is as follows: We take an assignment (we will see later how this assignment can be obtained) and then check for all clauses whether the assignment *witnesses* that the clause is globally blocked. We start with a formal notion of a witness:

Definition 19. *Given a clause C and a formula F, a conditional autarky $\alpha_\mathsf{c} \cup \alpha_\mathsf{a}$ (with conditional part α_c) for F* witnesses *that C is globally blocked in F if $\alpha_\mathsf{a} \cap C \neq \emptyset$ and $\alpha_\mathsf{c} \subseteq \overline{C}$.*

Suppose we have a conditional autarky $\alpha_\mathsf{c} \cup \alpha_\mathsf{a}$ with conditional part α_c for F and we want to use for checking if a clause C is globally blocked. We know from Theorem 16 that C is globally blocked by $L = \alpha_\mathsf{a}$ in F if $\alpha_\mathsf{a} \cap C \neq \emptyset$ and $\alpha_\mathsf{c} = \overline{C \setminus L}$. However, a closer look reveals that the requirement $\alpha_\mathsf{c} = \overline{C \setminus L}$ is needlessly restrictive for our purpose: Theorem 20 below implies that it suffices if α_c is a subset of $\overline{C \setminus L}$ (and thus of $\overline{C}$, since α_c assigns no variables of $L = \alpha_\mathsf{a}$) to guarantee that C is globally blocked. Hence, if we have a conditional autarky which witnesses (as defined above) that C is globally blocked, we can be sure that the clause is indeed globally blocked.

Theorem 20. *Let F be a formula, α a conditional autarky for F with autarky part α_a, and τ an assignment such that $\alpha \subseteq \tau$. Then, τ is a conditional autarky for F with autarky part α_a.*

Proof. Let $\alpha_\mathsf{c} = \alpha \setminus \alpha_\mathsf{a}$ and $\tau_c = \tau \setminus \alpha_\mathsf{a}$. We know that α_a is an autarky for $F|\alpha_\mathsf{c}$. Since $\alpha \subseteq \tau$, it follows that $\alpha_\mathsf{c} \subseteq \tau_c$. Now, let $D|\tau_c \in F|\tau_c$. If D is not satisfied by τ_c, then it is also not satisfied by α_c. Thus, if $D|\tau_c$ is touched by α_a, then it must be satisfied by α_a, for otherwise α_a is not an autarky for $F|\alpha_\mathsf{c}$. It follows that α_a is an autarky for $F|\tau_c$.

We can now present the algorithm (Fig. 2) for finding globally-blocked clauses. The algorithm repeatedly computes the least conditional part α_c of the given assignment (line 1) and then removes from α_c all literals that are not in $\overline{C}$ (line 2) because of the requirement $\alpha_\mathsf{c} \subseteq \overline{C}$. If the algorithm finally reaches a fixpoint, meaning that $\alpha_\mathsf{c} \subseteq \overline{C}$, it returns whether the autarky part has a non-empty intersection with C (line 3), which is necessary to guarantee that the assignment witnesses that C is globally blocked.

Theorem 21. *Let C be a clause, F a formula, and α an assignment. Then,* $\mathsf{IsGloballyBlocked}(C, F, \alpha) = \mathsf{TRUE}$ *if and only if a subassignment of α witnesses that C is globally blocked in F.*

Proof. In the rest of the proof, we denote by α^i the assignment passed to IsGloballyBlocked at the i-th recursive call (we denote the initial call as the 0-th recursive call, i.e., $\alpha^0 = \alpha$). The assignments α_c^i and α_a^i are defined accordingly.

```
IsGloballyBlocked(clause C, formula F, assignment α)
1   α_c := LeastConditionalPart(α, F), α_a := α \ α_c
2   α' := α_a ∪ (α_c ∩ C̄)
3   if (α' = α) then return α_a ∩ C ≠ ∅
4   return IsGloballyBlocked(C, F, α')
```

Fig. 2. Algorithm for detecting globally-blocked clauses.

For the "only if" direction, assume that $\mathsf{IsGloballyBlocked}(C, F, \alpha) = \mathsf{TRUE}$ and let α^n be the assignment to the last recursive call (i.e., α^n is the assignment for which the algorithm returns if $\alpha^n_a \cap C \neq \emptyset$). Then, since the algorithm only modifies the initial assignment α by unassigning variables, α^n is a subassignment of α. Now, since $\alpha^n_c = \mathsf{LeastConditionalPart}(\alpha, F)$, we know that α^n is a conditional autarky for F with (least) conditional part α^n_c. Moreover, all literals of α^n_c are contained in $\overline{C}$ due to line 2 of the algorithm. Finally, since α^n_a and C have a non-empty intersection, α^n witnesses that C is globally blocked in F.

For the "if" direction, suppose some subassignment $\tau = \tau_c \cup \tau_\alpha$ of α witnesses that C is globally blocked in F. Below, we show by induction on i that $\tau \subseteq \alpha^i$ and $\tau_a \subseteq \alpha^i_a$. From this, the statement follows then easily: Denote by α^n the assignment passed to the final recursive call (it can be easily seen that the algorithm terminates). Since $\tau_a \subseteq \alpha^n_\mathsf{a}$ and since $\tau_a \cap C \neq \emptyset$, it must then be the case that $\alpha_\mathsf{a} \cap C \neq \emptyset$. We conclude with the induction proof of the mentioned statement:

Induction start ($i = 0$): In this case, $\alpha^0 = \alpha$. By assumption $\tau \subseteq \alpha$. Thus, by Theorem 20, we know that τ_a is an autarky for $F|\alpha \setminus \tau_a$. Hence, as α_c is the least conditional part of α, it follows that $\alpha_\mathsf{c} \subseteq \alpha \setminus \tau_a$ and thus $\tau_a \subseteq \alpha \setminus \alpha_\mathsf{c} = \alpha_\mathsf{a}$.

Induction step ($i > 0$): We assume that $\tau \subseteq \alpha^{i-1}$ and $\tau_a \subseteq \alpha^{i-1}_a$. We first show that $\tau \subseteq \alpha^i$. The assignment α_i is obtained as $\alpha' = \alpha^{i-1}_a \cup \{l \mid l \in \alpha^{i-1}_c$ and $l \in \overline{C}\}$ in the $(i-1)$-th recursive call. Thus, since $\tau_a \subseteq \alpha^{i-1}_a$, we know that $\tau_a \subseteq \alpha_i$. Therefore, the only literals that are contained in α^i but not in α_{i-1} are literals of α^{i-1}_c that are not in $\overline{C}$. But such literals cannot be contained in τ_c since $\tau_c \subseteq \overline{C}$. It follows that $\tau \subseteq \alpha^i$. Hence, by Theorem 20, it follows that $\tau_a \subseteq \alpha^i_a$.

8 Implementation

The abstract algorithm presented in the previous section connects well to the presented theory of globally-blocked clauses, but is hard to implement efficiently. Figure 3 describes a refinement of the algorithm and further discusses implementation details which are crucial for efficiency. Without giving a detailed analysis, it is easy to see that for each candidate clause, the running time of the algorithm is similar and thus bounded by the time it would take to propagate all conditional variables obtained during the first step of the algorithm.

1. Split the assignment into a conditional part α_{c} and an autarky part α_{a} (one initial call to LeastConditionalPart in Fig. 1). Mark the resulting literals of α_{c} and save them on a *conditional stack*, gather *candidate clauses* (those with a literal that is true but not yet in the conditional part) and watch a true literal in all clauses with a true literal.
2. For each candidate clause C:
3. If C contains no literal from α_{a}, continue with next clause (goto 2).
4. Watch one literal l_{a} of α_{a} in C and mark all literals in C to be part of C. Actually have a variable pointing to the literal l_{a}.
5. For each unprocessed literal l_{c} on the conditional stack:
6. If $\bar{l}_{\mathsf{c}} \in C$ (cheap check since literals in C are marked) continue (goto 5).
7. Unassign $\bar{l}_{\mathsf{c}} \in C$ and push it on an *unassigned* stack.
8. For each unassigned literal u on the unassigned stack not processed yet:
9. For each clause D watched by u (through watches initialized in step 1):
10. Search for a replacement literal $r \in D$ which satisfies D. If such r is found, stop watching D with u, watch it with r instead, and continue with next clause D watched by u (goto 9).
11. Otherwise no replacement is found.
12. If there is no literal $k \in \alpha_{\mathsf{a}}$ with $\overline{k} \in D$, continue with next clause D watched by u (goto 9).
13. For each literal $k \in \alpha_{\mathsf{a}}$ with $\overline{k} \in D$:
14. Put k into the conditional part α_{c} by using another mark bit and push it onto the conditional stack.
15. If k is different from the watched literal $l_{\mathsf{a}} \in C$ (see step 4), continue with the next unassigned and unprocessed literal u on the unassigned stack.
16. Otherwise, search for a replacement of l_{a} in C.
17. If no replacement is found, C is not a globally-blocked clause; continue with next candidate clause (goto 2 – thus jump out of four loops).
18. If there are no unprocessed literals, neither on the conditional nor on the unassigned stack, and we still watch a literal of α_{a} in the candidate clause C, then we now reached a fix-point and C is globally blocked.
19. Eliminate C and put the autarky part as witness (found by traversing the assignment trail) and C on the extension stack for witness reconstruction.
20. Pop literals from unassigned stack and reassign them to their original value.
21. Pop literals from conditional stack pushed after initialization in step 1 and unmark their conditional bit.
22. Now we are back to the initial assignment after step 1, with the initial literals of the conditional part α_{c} marked as such and the literals of α_{a} unmarked.
23. Unmark literals marked in step 4 and continue with next clause (goto 2).

Fig. 3. Algorithm to extract globally-blocked clauses from a given assignment.

This variant has been implemented in C++ in CaDiCaL [37] and is available at `http://fmv.jku.at/globalblocking` (see "`condition.cpp`"). We experimented on SAT Competition benchmarks and also in an incremental bounded model checking setting [38]. Our algorithm in Fig. 3 does find non-trivial globally blocked clauses, but at this point we have not found an instance or application where the removal of globally-blocked clauses results in an overall improvement in running time. It thus remains to be seen whether or not the idea of removing globally-blocked clauses can be beneficial in practice.

One issue is particularly problematic: For some instances with many globally-blocked clauses, the large number of literals on the reconstruction stack requires too much memory, particularly if the autarky part—which serves as witness—contains a substantial fraction of all variables.

9 Conclusion

We introduced globally-blocked clauses, a new kind of redundant clauses that generalizes the existing notions of blocked clauses and set-blocked clauses. As we have shown, globally-blocked clauses correspond closely to conditional autarkies, which are special assignments that can be partitioned into two parts such that one part becomes an autarky once the other part has been applied to the formula.

Since finding globally-blocked clauses is non-trivial, we presented an algorithm that takes as input a formula and a clause together with a candidate assignment and then checks if the assignment (or a subassignment thereof) can witness that the clause is globally blocked in the formula.

Our algorithm simulates a well-known circuit preprocessing technique, known as unique sensitization, on the CNF level. Although our algorithm is conceptually simple, implementing it efficiently is far from straight-forward. We thus presented an implementation of our algorithm, which we ran on a range of formulas to evaluate its effectiveness in practice.

Acknowledgment This work has been supported by the National Science Foundation under grant CCF-1618574 and by the Austrian Science Fund (FWF) under project W1255 (LogiCS) and S11409-N23 (RiSE).

We want to thank Oliver Kullmann, who explained to the second author an (as far as we know unpublished) algorithm to compute the maximal autarky of a total assignment, which is a special case of LeastConditionalPart in Fig. 1.

This work was triggered by Gianpiero Cabodi who asked the last author whether there is a CNF level version of unique sensitization as explained in Sect. 3 with potential applications in SAT-based model checking.

We would finally also like to thank the organizers of ATVA'19 for inviting the last author to present these ideas as invited talk and include this invited paper in the proceedings.

References

1. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design **19**(1) (2001) 7–34
2. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In: Proc. of the 37th International Cryptology Conference (CRYPTO 2017), Springer (2017) 570–596
3. Konev, B., Lisitsa, A.: A SAT attack on the Erdős discrepancy conjecture. In: Proc. of the 17th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2014). Volume 8561 of LNCS., Springer (2014) 219–226
4. Heule, M.J.H.: Schur number five. In: Proc. of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018), AAAI Press (2018)
5. Järvisalo, M., Biere, A., Heule, M.J.H.: Simulating circuit-level simplifications on CNF. Journal on Automated Reasoning **49**(4) (2012) 583–619
6. Heule, M.J.H., Kiesl, B., Seidl, M., Biere, A.: PRuning through satisfaction. In: Proc. of the 13th International Haifa Verification Conference (HVC 2017). Volume 10629 of LNCS., Springer (2017) 179–194
7. Monien, B., Speckenmeyer, E.: Solving satisfiability in less than 2^n steps. Discrete Applied Mathematics **10**(3) (1985) 287 – 295
8. Kleine Büning, H., Kullmann, O.: Minimal unsatisfiability and autarkies. In Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. IOS Press (2009) 339–401
9. Kiesl, B., Seidl, M., Tompits, H., Biere, A.: Super-blocked clauses. In: Proc. of the 8th Int. Joint Conference on Automated Reasoning (IJCAR 2016). Volume 9706 of LNCS., Springer (2016) 45–61
10. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics **96-97** (1999) 149–176
11. Järvisalo, M., Biere, A., Heule, M.J.H.: Blocked clause elimination. In: Proc. of the 16th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010). Volume 6015 of LNCS., Springer (2010) 129–144
12. Kiesl, B., Suda, M., Seidl, M., Tompits, H., Biere, A.: Blocked clauses in first-order logic. In: Proc. of the 21st Int. Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-21). Volume 46 of EPiC Series in Computing., EasyChair (2017) 31–48
13. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: Proc. of the 23rd Int. Conf. on Automated Deduction (CADE-23). Volume 6803 of LNCS., Springer (2011) 101–115
14. Lonsing, F., Bacchus, F., Biere, A., Egly, U., Seidl, M.: Enhancing search-based QBF solving by dynamic blocked clause elimination. In: Proc. of the 20th Int. Conf. on Logic for Programming, Artificial Intelligence (LPAR-20). Volume 9450 of LNCS., Springer (2015) 418–433
15. Marques Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers **48**(5) (1999) 506–521
16. Kiesl, B.: Structural Reasoning Methods for Satisfiability Solving and Beyond. PhD thesis, TU Wien (2019)
17. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Proc. of the 6th Int. Joint Conference on Automated Reasoning (IJCAR 2012). Volume 7364 of LNCS., Springer (2012) 355–370

18. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Proc. of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005). Volume 3569 of LNCS., Springer (2005) 61–75
19. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the Boolean Pythagorean Triples problem via Cube-and-Conquer. In: Proc. of the 19th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2016). Volume 9710 of LNCS., Springer (2016) 228–245
20. Heule, M., Järvisalo, M., Biere, A.: Clause elimination procedures for CNF formulas. In: Proc. of the 17th Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-17). Volume 6397 of LNCS., Springer (2010) 357–371
21. Heule, M., Järvisalo, M., Biere, A.: Covered clause elimination. In: Short papers for the 17th Int. Conf. on Logic for Programming, Artificial intelligence, and Reasoning (LPAR-17-short). Volume 13 of EPiC Series., EasyChair (2010) 41–46
22. Heule, M.J.H., Järvisalo, M., Lonsing, F., Seidl, M., Biere, A.: Clause elimination for SAT and QSAT. Journal of Artifical Intelligence Research **53** (2015) 127–168
23. Wetzler, N.D., Heule, M.J.H., Hunt Jr., W.A.: DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In: Proc. of the 17th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2014). Volume 8561 of LNCS., Springer (2014) 422–429
24. Cruz-Filipe, L., Heule, M.J.H., Jr., W.A.H., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In de Moura, L., ed.: Proc. of the 26th Int. Conference on Automated Deduction (CADE-26). Volume 10395 of LNCS., Springer (2017) 220–236
25. Lammich, P.: Efficient verified (UN)SAT certificate checking. In: Proc. of the 26th Int. Conference on Automated Deduction (CADE-26). Volume 10395 of LNCS., Springer (2017) 237–254
26. Heule, M.J.H., Järvisalo, M., Suda, M.: SAT competition 2018. (2019)
27. Heule, M.J.H., Kiesl, B., Biere, A.: Short proofs without new variables. In: Proc. of the 26th Int. Conference on Automated Deduction (CADE-26). Volume 10395 of LNCS., Springer (2017) 130–147
28. Heule, M.J.H., Biere, A.: What a difference a variable makes. In: Proc. of the 24th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2018). Volume 10806 of LNCS., Springer (2018) 75–92
29. Heule, M.J.H., Kiesl, B., Biere, A.: Clausal proofs of mutilated chessboards. In: Proc. of the 11th Int. NASA Formal Methods Symposium (NFM 2019). Volume 11460 of LNCS., Springer (2019) 204–210
30. Heule, M.J.H., Kiesl, B., Biere, A.: Encoding redundancy for satisfaction-driven clause learning. In: Proc. of the 25th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2019). Volume 11427 of LNCS., Springer (2019) 41–58
31. Heule, M.J.H., Seidl, M., Biere, A.: Solution validation and extraction for QBF preprocessing. Journal of Automated Reasoning (2016) 1–29
32. Heule, M., Seidl, M., Biere, A.: Blocked literals are universal. In: Proc. of the 7th Int. NASA Formal Methods Symposium (NFM 2015). Volume 9058 of LNCS., Springer (2015) 436–442
33. Lonsing, F., Egly, U.: QRAT+: generalizing QRAT by a more powerful QBF redundancy property. In: Proc. of the 9th Int. Joint Conference on Automated Reasoning (IJCAR 2018). Volume 10900 of LNCS., Springer (2018) 161–177

34. Lonsing, F., Egly, U.: QRATPre+: Effective QBF preprocessing via strong redundancy properties. In: Proc. of the 22nd Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2019). Volume 11628 of LNCS., Springer (2019) 203–210
35. Kiesl, B., Suda, M.: A unifying principle for clause elimination in first-order logic. In: Proc. of the 26th Int. Conference on Automated Deduction (CADE-26). Volume 10395 of LNCS., Springer (2017) 274–290
36. Fujiwara, H.: FAN: A fanout-oriented test pattern generation algorithm. In: Proc. of the IEEE Int. Symposium on Circuits and Systems (ISCAS 85). (1985) 671–674
37. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2018. In Heule, M., Järvisalo, M., Suda, M., eds.: Proc. of SAT Competition 2018 – Solver and Benchmark Descriptions. Volume B-2018-1 of Department of Computer Science Series of Publications B., University of Helsinki (2018) 13–14
38. Fazekas, K., Biere, A., Scholl, C.: Incremental inprocessing in SAT solving. In: Proc. of the 22nd Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2019). Volume 11628 of LNCS., Springer (2019) 136–154