Journal of Artificial Intelligence Research 53 (2015) 127-168 Submitted 1/15; published 6/15

Clause Elimination for SAT and QSAT

Marijn Heule MARIJN@CS.UTEXAS.EDU
Department of Computer Science,
The University of Texas at Austin, USA

Matti Järvisalo MATTI.JARVISALO@CS.HELSINKI.FI
HIIT, Department of Computer Science,
University of Helsinki, Finland

Florian Lonsing FLORIAN.LONSING@TUWIEN.AC.AT
Institute of Information Systems,
Vienna University of Technology, Austria

Martina Seidl MARTINA.SEIDL@JKU.AT
Armin Biere BIERE@JKU.AT
Institute for Formal Models and Verification,
Johannes Kepler University Linz, Austria

Abstract

The famous archetypical NP-complete problem of *Boolean satisfiability* (SAT) and its PSPACE-complete generalization of *quantified Boolean satisfiability* (QSAT) have become central declarative programming paradigms through which real-world instances of various computationally hard problems can be efficiently solved. This success has been achieved through several breakthroughs in practical implementations of decision procedures for SAT and QSAT, that is, in SAT and QSAT solvers. Here, simplification techniques for conjunctive normal form (CNF) for SAT and for prenex conjunctive normal form (PCNF) for QSAT—the standard input formats of SAT and QSAT solvers—have recently proven very effective in increasing solver efficiency when applied before (i.e., in *preprocessing*) or during (i.e., in *inprocessing*) satisfiability search.

In this article, we develop and analyze clause elimination procedures for pre- and inprocessing. Clause elimination procedures form a family of (P)CNF formula simplification techniques which remove clauses that have specific (in practice polynomial-time) redundancy properties while maintaining the satisfiability status of the formulas. Extending known procedures such as tautology, subsumption, and blocked clause elimination, we introduce novel elimination procedures based on *asymmetric* variants of these techniques, and also develop a novel family of so-called *covered clause elimination* procedures, as well as natural liftings of the CNF-level procedures to PCNF. We analyze the considered clause elimination procedures from various perspectives. Furthermore, for the variants not preserving logical equivalence under clause elimination, we show how to reconstruct solutions to original CNFs from satisfying assignments to simplified CNFs, which is important for practical applications for the procedures. Complementing the more theoretical analysis, we present results on an empirical evaluation on the practical importance of the clause elimination procedures in terms of the effect on solver runtimes on standard real-world application benchmarks. It turns out that the importance of applying the clause elimination procedures developed in this work is empirically emphasized in the context of state-of-the-art QSAT solving.

1. Introduction

Boolean satisfiability (SAT) is the problem of determining whether a given propositional logic formula has a solution. SAT has become an important declarative approach to formulate and solve various NP-hard problems—a general coverage of modern satisfiability research is provided by Biere, Heule, van Maaren, and Walsh (2009). Contrasting the classical worst-case view on NP-completeness as intractability (Cook, 1971; Garey & Johnson, 1979), central to the success of the SAT-based approach are major advances in robust implementations of decision procedures for SAT, i.e., *SAT solvers*. Modern SAT solvers are routinely used in a vast number of different industrial and artificial intelligence applications (Claessen, Eén, Sheeran, & Sörensson, 2008; Marques-Silva, 2008), giving rise to a high demand for new techniques for further improving the robustness and efficiency of current state-of-the-art SAT solvers.

As SAT is the archetypical problem for NP, the *quantified Boolean satisfiability* (QSAT) problem of evaluating *quantified Boolean formulas* (QBF), the well-known extension of SAT, is archetypical for PSPACE, offering a powerful framework for modelling a very large range of important computational problems in artificial intelligence, knowledge representation, verification, and synthesis (Benedetti & Mangassarian, 2008). During the last decade, much effort has been spent in the development of efficient QSAT solvers. Despite several success stories, much research effort is needed for QSAT solving to reach the level of maturity of modern SAT solvers. Due to the wide range of possible QSAT applications, developing more efficient QSAT solver technology is indeed an important on-going quest. A major part of this quest is to lift techniques proven effective in SAT solving to the more general framework of QSAT solving and to analyze their impact.

Simplification techniques applied both before (i.e., in preprocessing) and during search have proven integral in enabling efficient conjunctive normal form (CNF) level SAT solving for real-world application domains. Indeed, there is a large body of work on preprocessing CNF formulas (Freeman, 1995; Le Berre, 2001; Lynce & Marques-Silva, 2001; Bacchus, 2002; Ostrowski, Grégoire, Mazure, & Saïs, 2002; Brafman, 2004; Subbarayan & Pradhan, 2005; Gershman & Strichman, 2005; Eén & Biere, 2005; Van Gelder, 2005; Fourdrinoy, Grégoire, Mazure, & Saïs, 2007a, 2007b; Jin & Somenzi, 2005; Han & Somenzi, 2007; Piette, Hamadi, & Saïs, 2008; Järvisalo, Biere, & Heule, 2010; Manthey, Heule, & Biere, 2013; Heule, Järvisalo, & Biere, 2013b) based on, for examples, variable elimination and equivalence reasoning. Further, while many SAT solvers rely mainly on Boolean constraint propagation (that is, *unit propagation*) during search, it is possible to improve solving efficiency by applying additional simplification techniques also during search. This dynamic interplay between simplification and search is captured by the *inprocessing* SAT solving paradigm (Järvisalo, Heule, & Biere, 2012b). Inprocessing SAT solvers have been recently shown to push further the efficiency of SAT solving, as witnessed for example by LINGELING (Biere, 2013), one of the most successful SAT solvers in the recent SAT Competitions (Järvisalo, Le Berre, Roussel, & Simon, 2012; SAT Competitions Organizing Committee, 2014). Importantly, when scheduling *combinations* of simplification techniques during search, even quite simple ideas, such as removal of subsumed clauses, can bring additional gains by enabling further simplifications by other techniques.

Motivated by the impact of preprocessing in SAT, some preprocessors for QSAT have started to emerge, and have proven advantageous for the evaluation of representative QSAT benchmarks (Samulowitz, Davies, & Bacchus, 2006; Bubeck & Kleine Büning, 2007; Giunchiglia, Marin, & Narizzano, 2010; Mangassarian, Le, Goultiaeva, Veneris, & Bacchus, 2010; Pigorsch & Scholl, 2010). In fact, there is high promise for achieving further advances to the efficiency of QSAT solvers through adding stronger simplification techniques to the solving flow. Intuitively, this is due to the fact that, in light of simplification and preprocessing techniques, real-world SAT and QSAT instances tend to notably differ in their size characteristics. Real-world application SAT instances still solvable with state-of-the-art SAT solvers today can contain up to tens of millions of variables and clauses (Järvisalo et al., 2012), which restricts the use of theoretically interesting polynomial-time simplification techniques in practical applications due to the shear size of the input CNF formulas the solvers must be able to cope with. In contrast, QSAT instances are often relatively small, because the language of QBF enables more succinct encodings via quantification. Despite their small size, QBFs can be very challenging for state-of-the-art solvers to solve. Hence there is more room for successful applications of computationally intensive (but still polynomial-time) simplification rules. Inprocessing for QSAT has hardly been considered so far; the solver STRUQS (Pulina & Tacchella, 2009) combines search-based solving with variable elimination what may be considered as a step in this direction.

The focus of this article is on preprocessing and simplification techniques for SAT and QSAT solving. This work is motivated on one hand by the possibilities of improving SAT and QSAT solving efficiency further by integrating additional simplification techniques to the solving process before and/or during search, and on the other hand by understanding the relationships between different simplification techniques. Especially, we concentrate on developing and analyzing *clause elimination procedures* for CNF (for SAT) and PCNF formulas (for QSAT)—the standard input formats of SAT and QSAT solvers.

Clause elimination procedures form a specific family of simplification techniques which focus on removing redundant clauses—with respect to specific redundancy properties—from CNF formulas in a satisfiability-preserving way. More precisely, a clause elimination procedure based on a redundancy property P is a procedure which, given a CNF (or PCNF) formula F, removes iteratively until fixpoint from F clauses which have P. For any well-defined redundancy property P, it holds that for any clause C which has P in a (P)CNF formula F, F and F without C are satisfiability-equivalent. In other words, F is satisfiable whenever F without C is satisfiable.

However, the most general redundancy property, simply requiring satisfiability-equivalence under clause elimination, is not applicable in practice, since checking whether a clause is C redundant under this property is co-NP-complete (Liberatore, 2005). In connection to practically relevant clause elimination procedures, in the context of this work of specific interest are clause elimination procedures that are based on polynomial-time checkable redundancy properties. As simple examples in the context of SAT, two such well-known redundancy properties are tautology and subsumption. The corresponding clause elimination procedures are tautology elimination and subsumption elimination (Eén & Biere, 2005). The more sophisticated redundancy property of blocked clauses (Kullmann, 1999) allows for blocked clause elimination (Ostrowski et al., 2002; Järvisalo et al., 2010).

As extensions of the known procedures, in this work we introduce novel elimination procedures based on *asymmetric* variants of the techniques. In the asymmetric variants, a clause in a CNF is first augmented with certain literals so that the satisfiability of the CNF is preserved. The original clause is replaced by the augmented one. If the augmented clause turns out to have a redundancy property, then it is eliminated from the CNF. Otherwise, the original clause is restored. We also develop a novel family of so-called *covered clause elimination* procedures. For applications to the more general setting of QSAT, we develop natural liftings of the CNF-level procedures to PCNF, which turn out—naturally—to be somewhat more involved.

We analyze the resulting clause elimination procedures from various perspectives. One property is *reduction power*, that is, the ability to remove clauses and thus reduce the size of the CNF formula. The relative reduction power of two clause elimination procedures reveals the potential strengths of the procedures, subject to practical realizations of the more powerful procedures being fast enough to speed up the total solving time. Another orthogonal property we consider is *BCP-preservance*, that is, the ability to preserve all possible unit propagations that can also be done on the original CNF. BCP-preservance amounts to the question of whether different clause elimination procedures maintain arc consistency on the clausal level w.r.t. the original CNF formula. A third property we consider, *confluence*, implies that a procedure has a unique fixpoint; for practical realizations, knowledge of whether a simplification procedure is confluent is of interest. For non-confluent procedures, well-working elimination-ordering heuristics have to be developed. The fourth property we consider is whether the procedures maintain *logical equivalence* with respect to the original CNF, that is, preserve the set of satisfying assignments. Maintaining logical equivalence is most often not necessary for applications in which only a single solution is sought for. However, for simplification techniques which maintain only satisfiability but not logical equivalence, it is important to develop algorithms for fast reconstruction of a satisfying assignment to the original CNF from an assignment to the simplified instance. Motivated by this, for the variants which do not preserve logical equivalence, we show how to efficiently reconstruct solutions to original CNFs from satisfying assignments to simplified CNFs.

Complementing the analysis of the properties and relationships between the considered clause elimination procedures, we also provide empirical results on the practical implications of the clause elimination procedures in terms of runtime improvements with state-of-the-art SAT and QSAT solvers on real-world application benchmarks. The empirical results show that the clause elimination procedures developed in this work have a clear positive effect on the performance of various state-of-the-art QSAT solvers, while the impact on the performance of inprocessing SAT solving is less announced.

The rest of this article is organized as follows. After preliminaries on SAT, QSAT, and related necessary concepts (Section 2), we present an overview of the results on the properties of clause elimination procedures (Section 3). Technical analysis of the clause elimination procedures for SAT and QSAT are presented in Sections 4–6, followed by a section on solution reconstruction (Section 7). Before concluding, results from an empirical evaluation of the procedures are presented in Section 8.

This article extends and thoroughly revises work presented earlier at the 17th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2010) (Heule, Järvisalo, & Biere, 2010, 2013a) and at the 23rd International Conference on Automated Deduction (CADE 2011) (Biere, Lonsing, & Seidl, 2011). The variants of quantified covered clause elimination (in Section 6) have not been published previously in detail. Further, model reconstruction for the variants of covered clause elimination (in Section 7) are new. Compared to the earlier publications, the empirical evaluation presented in this article is extended and updated with more recent state-of-the-art solvers and benchmarks. Definitions of some of the clause elimination procedures, and the related analysis, have been updated to better reflect the current insights into these procedures. Furthermore, discussions, examples, and background have been extended.

2. Preliminaries

In this section we review necessary background concepts: Boolean satisfiability, resolution, Boolean constraint propagation, as well as their counterpart in the more general context of quantified Boolean formulas.

2.1 Boolean Satisfiability

For a Boolean variable x, there are two *literals*, the positive literal, denoted by x, and the negative literal, denoted by $\bar{x}$. A *clause* is a disjunction of literals and a CNF formula is a conjunction of clauses. A clause can be seen as a finite set of literals and a CNF formula as a finite set of clauses. The set of literals occurring in a CNF formula F is denoted by $\mathsf{lits}(F)$. A *unit clause* contains exactly one literal. A clause is a *tautology* if it contains both x and $\bar{x}$ for some variable x. Given a CNF formula F, a clause $C_1 \in F$ *subsumes* (another) clause $C_2 \in F$ in F if and only if $C_1 \subset C_2$. Then C_2 is *subsumed by* C_1.

A truth assignment for a CNF formula F is a function τ that maps variables in F to $\{\mathbf{t}, \mathbf{f}\}$. If $\tau(x) = v$, then $\tau(\bar{x}) = \neg v$, where $\neg\mathbf{t} = \mathbf{f}$ and $\neg\mathbf{f} = \mathbf{t}$. A clause C is satisfied by τ if $\tau(l) = \mathbf{t}$ for some $l \in C$. An assignment satisfies F if it satisfies every clause in F. An assignment falsifies a clause C if it assigns all literals that occur in C to $\mathbf{f}$.

Two CNF formulas are *logically equivalent* if they have the same set of satisfying assignments over the common variables.

2.1.1 RESOLUTION AND BCP

The classical resolution proof system (Robinson, 1965) for CNF formulas consists of the *resolution rule*, which states that, given two clauses C_1 and C_2 with $l \in C_1$ and $\bar{l} \in C_2$, the clause $C = (C_1 \setminus \{l\}) \cup (C_2 \setminus \{\bar{l}\})$, called the *resolvent* of C_1 and C_2, can be inferred by *resolving* on the literal l. This is denoted by $C = C_1 \otimes_l C_2$. The resolution rule not only forms a complete proof system for SAT, but it is also important as an inference rule used in preprocessing CNF formulas.

Boolean constraint propagation (BCP) or *unit propagation* is based on applying *unit resolution*, i.e., the special case of the resolution rule in which one of the clauses C_1 and C_2 is a unit clause. BCP is a central propagation mechanism applied within the typical DPLL and CDCL-based SAT solvers. For a CNF formula F, BCP propagates all unit clauses, that is, repeats the following until fixpoint:

> If there is a unit clause $(l) \in F$, remove from $F \setminus \{(l)\}$ all clauses that contain the literal l, and remove the literal $\bar{l}$ from all clauses in F.

The resulting formula is referred to as $\mathrm{BCP}(F)$. It is easy to see that BCP has a unique fixpoint for any CNF formula. In other words, BCP is *confluent*.

If $(l) \in \mathrm{BCP}(F)$ for some unit clause $(l) \notin F$, we say that BCP of F *assigns* the literal l to $\mathbf{t}$ (and the literal $\bar{l}$ to $\mathbf{f}$). If $(l), (\bar{l}) \in \mathrm{BCP}(F)$ for some literal $l \notin F$ (or, equivalently, $\emptyset \in \mathrm{BCP}(F)$), we say that BCP *derives a conflict* on F. For notational convenience, for a partial assignment τ over the variables in F, let $\mathrm{BCP}(F, \tau) := \mathrm{BCP}(F \cup T_\tau \cup F_\tau)$, where $T_\tau = \{(x) \mid \tau(x) = \mathbf{t}\}$ and $F_\tau = \{(\bar{x}) \mid \tau(x) = \mathbf{f}\}$. In words, $\mathrm{BCP}(F, \tau)$ denotes the formula obtained by adding to F unit clauses corresponding to the variable assignments in τ.

2.2 Quantified Boolean Formulas

A *quantified Boolean formula G* in *prenex conjunctive normal form* (PCNF) has the structure $\Pi.F$ with quantifier prefix Π and propositional matrix F in conjunctive normal form. The quantifier prefix Π is an ordered partition $Q_1 \dots Q_n$ of the variables in F. The size of the quantifier prefix $\Pi = Q_1 \dots Q_n$, denoted by $|\Pi|$, is $|Q_1| + \dots + |Q_n|$. An element Q_i of Π is called a *scope* or *quantifier block*. The function $\mathsf{quant}(Q_i)$ assigns either a universal quantifier $\forall$ or an existential quantifier $\exists$ to scope Q_i in such a way that $\mathsf{quant}(Q_i) \neq \mathsf{quant}(Q_{i+1})$. For convenience we also write $Qx_1, \dots, x_n$ for a scope $S = \{x_1, \dots, x_n\}$ with $\mathsf{quant}(S) = Q$ and $Q \in \{\forall, \exists\}$. The *quantifier level* of a variable x with $x \in Q_i$ is i, i.e., one plus the number of the preceding scopes. In the following, we assume that each variable of F occurs exactly once in the prefix. We say that a variable x is universal (existential) in QBF $\Pi.F$ if $x \in S$ in Π and $\mathsf{quant}(S) = \forall$ ($\mathsf{quant}(S) = \exists$). The notions of literals, clauses, and tautologies follow those for SAT. The function $\mathsf{var}(l)$ returns x if l is of the form x or $\bar{x}$. If $l = x$ then $\bar{l} = \bar{x}$ else $\bar{l} = x$. For a literal l with $\mathsf{var}(l) \in S$, $\mathsf{quant}(l) = \mathsf{quant}(S)$. For a clause C, its existential and its universal literals are given by $L_Q(C) = \{l \in C \mid \mathsf{quant}(l) = Q\}$ with $Q \in \{\forall, \exists\}$. For literals l, l' with $\mathsf{var}(l) \in Q_i$ and $\mathsf{var}(l') \in Q_j$, $l \leq l'$ if $i \leq j$.

Let $G = \Pi.F$ be a QBF and l a literal. Then $G[l]$ denotes the QBF which is obtained from G by deleting each clause C with $l \in C$, by removing each occurrence of $\bar{l}$, and by substituting the scope Q_i with $\mathsf{var}(l) \in Q_i$ by $Q_i \backslash \{\mathsf{var}(l)\}$.

The truth value of a QBF $G = \Pi.F$ is recursively defined as follows.

- If $F = \emptyset$ then G is satisfiable, if $\emptyset \in F$ then G is unsatisfiable.
- If $\mathsf{quant}(Q_1) = \forall$ and $x \in Q_1$, then G is satisfiable iff $G[x]$ and $G[\bar{x}]$ is satisfiable.
- If $\mathsf{quant}(Q_1) = \exists$ and $x \in Q_1$, then G is satisfiable iff $G[x]$ or $G[\bar{x}]$ is satisfiable.

This definition of QBF semantics indicates that the ordering of the variables in the prefix impacts the truth value of a formula. The prefix ordering introduces an ordering between the variables such that a variable x has to be assigned before a variable y if $x < y$. This restriction is specific to QBF and is does not apply to propositional logic where variables can be assigned in any order. The following example illustrates the consequences of swapping quantifiers.

Example 1. *The QBF $G = \forall x \exists y.((x \vee \bar{y}) \wedge (\bar{x} \vee y))$ is satisfiable, whereas the QBF $G' = \exists y \forall x.((x \vee \bar{y}) \wedge (\bar{x} \vee y))$ obtained from G by swapping $\forall x$ and $\exists y$ in the prefix is unsatisfiable.*

Intuitively, the semantics of QBF can also be considered as a two-player game (Schaefer, 1978) with an existential player and a universal player. The former controls the existentially quantified variables with the goal to satisfy the formula and the latter controls the universally quantified variables with the goal to falsify the formula. The moves are performed according to the order of the variables in the quantifier prefix from the left to the right. Obviously, the universal player takes advantage of the CNF structure, because conflicts can be easily detected. To reduce this bias while preserving the benefits of the CNF, approaches realizing *duality-aware reasoning* have been presented (Zhang, 2006; Klieber, Sapra, Gao, & Clarke, 2010; Goultiaeva & Bacchus, 2013; Goultiaeva, Seidl, & Biere, 2013; Sabharwal, Ansótegui, Gomes, Hart, & Selman, 2006).

2.2.1 QBF MODELS

In the context of QSAT there are different definitions of models (satisfying assignments). The choice of one particular definition is motivated by the actual application and underlying formal framework. In a theoretical setting, *satisfiability models* of QBFs were presented as sets of Skolem functions (Kleine Büning & Bubeck, 2009). A *Skolem function* f_y models the values an existential variable y can take to satisfy the matrix with respect to the universal variables on which y depends. In general, a Skolem function f_y for one particular y is not unique. Replacing y by f_y produces a semantically equivalent formula. This definition of satisfiability models is the theoretical foundation of certificate extraction for QBFs from resolution proofs (Balabanov & Jiang, 2011; Niemetz, Preiner, Lonsing, Seidl, & Biere, 2012). In the context of the game-based view on the QBF semantics, a similar approach for extracting winning strategies was introduced by Goultiaeva, Van Gelder, and Bacchus (2011). An approach to directly produce Skolem functions by symbolic skolemization (Benedetti, 2005a) was implemented to verify results of the solver sKIZZO (Benedetti, 2005b). Skolem function extraction from so-called QRAT proofs was recently proposed by Heule, Seidl, and Biere (2014b).

In certain applications related to preprocessing in QSAT, it is necessary to explicitly distinguish variable assignments which satisfy the matrix. Recursive semantics and satisfiability models are too coarse and hence not suitable for that purpose. Instead, tree-like models for QBFs can be applied (Samer, 2008; Samulowitz et al., 2006). In a tree-like model, every path from the root of the tree to a leaf comprises a variable assignment which satisfies the matrix. Assignments to universal variables are reflected by branches in the tree. Tree-like models are the formal foundation of preprocessing techniques in QSAT such as hyper binary resolution (Samulowitz et al., 2006) and failed literal detection (Van Gelder, Wood, & Lonsing, 2012) and are also relevant for theoretical work such as dependency schemes (Samer, 2008).

The different notions of models give rise to different definitions of *equivalence* in the theory of QSAT. In analogy with SAT, equivalence of two QBFs G_1 and G_2 can be checked by comparing the sets of tree-like models of G_1 and G_2, respectively. Such explicit comparison is impossible if recursive semantics is applied. For example, the definition of recursive semantics (Kleine Büning & Bubeck, 2009) distinguishes only between different assignments to the free variables in the QBF, i.e., variables which occur in the matrix but which are not explicitly quantified in the prefix. In this paper, we only consider *closed QBFs* without free variables.

Two QBFs G and G' are *satisfiability-equivalent* if and only if the following holds: G is satisfiable if and only if G' is satisfiable. For simplicity, in the context of QSAT we write "equivalent" instead of "satisfiability-equivalent".

2.2.2 Q-RESOLUTION

Many techniques used in SAT can be transferred to QSAT with certain adaptions to preserve soundness. In the following, we introduce some of these techniques which are important for the rest of the paper.

For defining *Q-resolution* (Kleine Büning, Karpinski, & Flögel, 1995), a lifting of the resolution rule to QSAT, we first review the concept of *universal reduction* (UR) (Kleine Büning et al., 1995; Cadoli, Giovanardi, & Schaerf, 1998).

Definition 1. *A* universally reduced clause C' *is obtained from a clause* C *by applying universal reduction until fixpoint, i.e.,*

$$C := C \backslash \{l \in C \mid \mathsf{quant}(l) = \forall, \text{ and there is no } l' \in C \text{ with } \mathsf{quant}(l') = \exists \text{ and } l < l'\}.$$

The removal of a universally quantified literal l from a clause which does not contain any existentially quantified literals with a higher level than l is called universal reduction*.*

It can be easily shown that the application of universal reduction is confluent and preserves satisfiability of a formula, provided that it is applied to non-tautological clauses only. Based on the universal reduction rule, Q-resolution is a combination of resolution for propositional logic and universal reduction.

Definition 2. *The* Q-resolvent $C_1 \otimes_l C_2$ *of two non-tautological clauses C_1 and C_2 with $l \in C_1$, $\bar{l} \in C_2$, and* $\mathsf{quant}(l) = \exists$ *is defined as $(C_1' \setminus \{l\}) \cup (C_2' \setminus \{\bar{l}\})$ where C_1' and C_2' are the universally reduced clauses obtained from C_1 and C_2, respectively. The literal l is called the* pivot element*.*

The construction rule of Q-resolvents, enhanced with the universal reduction rule, forms the quantified resolution calculus which is sound and refutationally-complete for QSAT (Kleine Büning et al., 1995): a QBF G is unsatisfiable if and only if the empty clause can be derived from G by Q-resolution and universal reduction. When combining universal reduction (as in Definition 1) and Q-resolution (as in Definition 2), the restriction of Q-resolution to non-tautological clauses is crucial for soundness, as the following example shows.

Example 2. *Consider the satisfiable QBF $G = \forall a \exists x. C_1 \wedge C_2$, where $C_1 = (a \vee \bar{a} \vee x)$ and $C_2 = (\bar{x})$. Universal reduction cannot reduce literals from C_1 and C_2. Furthermore, C_1 and C_2 do not have a Q-resolvent since C_1 is tautological. If the restriction of Q-resolution to non-tautological clauses is ignored, then a Q-resolvent of C_1 and C_2 is $C = C_1 \otimes_x C_2 = (a \vee \bar{a})$. Universal reduction reduces C to the empty clause, which* erroneously *determines G as unsatisfiable.*

2.2.3 UNIT AND PURE LITERALS FOR QSAT

The unit literal rule for QSAT, which is part of the definition of BCP for QSAT (QBCP) given in the following, is obtained by extending from SAT to QSAT as follows. The variable of a unit literal is required to be existentially quantified, because universal reduction immediately reduces a clause containing only a universal literal to the empty clause. By taking universal reduction into account, we arrive at the following rule for unit propagations in QSAT.

Definition 3. *An existentially quantified literal l is* unit *in QBF $G = \Pi.F$ if $\{l, l_1, \ldots, l_m\} \in F$ with* $\mathsf{quant}(l_i) = \forall$ *and $l < l_i$. If l is unit in G, then G is equivalent to $G[l]$.*

Obviously, if a QBF G contains a non-tautological clause with universally quantified literals only, then G is unsatisfiable. Note that unit literal elimination allows to ignore the quantifier ordering during the evaluation, i.e., to assign a variable that is not a member of the outermost quantifier block.

Another important rule allowing to assign a variable not occurring in the outermost quantifier block is *quantified pure literal elimination*.

Definition 4. *A literal l is* pure *in a QBF $G = Q_1 \ldots Q_n.F$ if $l \in \bigcup_{C \in F}$ and $\bar{l} \notin \bigcup_{C \in F}$. Then G is equivalent to $G[l]$ if* $\mathsf{quant}(l) = \exists$ *and equivalent to $G[\bar{l}]$ if* $\mathsf{quant}(l) = \forall$*.*

In addition to unit propagation through which BCP for SAT is defined, the QSAT-specific variant of BCP (QBCP) includes quantified pure literal elimination and universal reduction (UR) applied until fixpoint. Both the elimination of universal pure literals and universal reduction increase the deductive power of QBCP. In other words, the successive application of these rules together with unit propagation allows to identify unit clauses which would be missed if only unit propagation was applied.

3. Overview of Contributions

In this section, we give an overview of our main results. For this overview, we present several important definitions of basic clause elimination techniques. Further, we introduce several measures of comparison, such as the notion of relative reduction power, which allow a detailed analysis of the various techniques.

3.1 Clause Elimination Procedures for SAT and QSAT

Generally speaking, a clause elimination procedure based on a *redundancy property* P is a procedure which, given a CNF (or PCNF) formula F, removes iteratively until fixpoint from F clauses which have P. For any well-defined redundancy property P, it should hold that for any clause C that has P in a (P)CNF formula F, F and $F \setminus \{C\}$ are satisfiability-equivalent. In connection to practically relevant clause elimination procedures, in the context of this work of specific interest are clause elimination procedures which are based on polynomial-time checkable redundancy properties. As simple examples in the context of SAT, two such well-known redundancy properties are *tautology* and *subsumption*, which give the corresponding clause elimination procedures of tautology elimination and subsumption elimination.

Definition 5 (Tautology Elimination)**.** *For a given formula F,* tautology elimination (TE) *repeats the following until fixpoint: If there is a tautological clause $C \in F$, let $F := F \setminus \{C\}$. The CNF formula resulting from applying* TE *on F is denoted by* $\mathrm{TE}(F)$.

Definition 6 (Subsumption Elimination)**.** *For a given formula F,* subsumption elimination (SE) *repeats the following until fixpoint: If there is a subsumed clause $C \in F$, let $F := F \setminus \{C\}$. The CNF formula resulting from applying* SE *on F is denoted by* $\mathrm{SE}(F)$.

A third earlier defined—but somewhat more involved—polynomial-time checkable redundancy property is that of a clause being *blocked*[1] (Kullmann, 1999), which gives the corresponding technique of *blocked clause elimination* (BCE). Blocked clause elimination has been recently shown to be surprisingly effective in simulating various structure-based simplification mechanisms purely on the CNF-level (Järvisalo, Biere, & Heule, 2012a), further motivating the technique both from the theoretical and the practical perspectives.

Definition 7 (Blocked Clause Elimination for SAT)**.** *Given a CNF formula F, a clause C, and a literal $l \in C$, the literal l* blocks *C w.r.t. F if for each clause $C' \in F$ with $\bar{l} \in C'$, $C \cup (C' \setminus \{\bar{l}\})$ is a tautology. Given a CNF formula F, a clause C is blocked w.r.t. F if there is a literal that blocks C w.r.t. F. For a CNF formula F,* blocked clause elimination *(*BCE*) repeats the following until fixpoint: If there is a blocked clause $C \in F$ w.r.t. F, let $F := F \setminus \{C\}$. The CNF formula resulting from applying* BCE *on F is denoted by* $\mathrm{BCE}(F)$.

1. Kullmann defines a blocking literal $l \in C$ as a literal for which it holds that all resolvents $C \otimes_l D$ with $\bar{l} \in D$ are tautologies. Our definition is slightly different in that it implies that binary tautologies are blocked clauses, in contrast to Kullmann's definition. Apart from that detail, the two definitions are equivalent.

Example 3. *Consider the following CNF formula, having a structure that is often observed in CNF encodings of graph coloring problems. The formula encodes a graph with two vertices v and w and an edge between them using three colors. The variable v_i (or w_i) has the interpretation that vertex v (or w) gets color i.*

$$F_{\mathrm{BCE}} = (v_1 \vee v_2 \vee v_3) \wedge (w_1 \vee w_2 \vee w_3) \wedge (\bar{v}_1 \vee \bar{w}_1) \wedge (\bar{v}_2 \vee \bar{w}_2) \wedge (\bar{v}_3 \vee \bar{w}_3) \wedge \\ (\bar{v}_1 \vee \bar{v}_2) \wedge (\bar{v}_1 \vee \bar{v}_3) \wedge (\bar{v}_2 \vee \bar{v}_3) \wedge (\bar{w}_1 \vee \bar{w}_2) \wedge (\bar{w}_1 \vee \bar{w}_3) \wedge (\bar{w}_2 \vee \bar{w}_3).$$

The first two clauses encode that v and w have at least one color. The next three clauses force that v and w cannot have the same color. The last six clauses denote that v and w have at most one color. It is easy to check that these last six clauses are blocked in F_{BCE}, since for each of these clauses, both of the two literals block the clause. Thus BCE *will remove these last six binary clauses from F_{BCE}. Thus the formula $\mathrm{BCE}(F_{\mathrm{BCE}})$ does not include the at-most-one-color constraints over the nodes v and w, and hence, in contrast to F_{BCE}, $\mathrm{BCE}(F_{\mathrm{BCE}})$ has satisfying assignments in which v or w are assigned multiple colors. However, given such a satisfying assignment, there is a simple linear-time algorithm (see Section 7 for details) for reconstructing a satisfying assignment to F_{BCE}, i.e., an assignment in which v and w are both assigned only a single color.*

In this work, we focus on a total of eight different clause elimination procedures for CNF formulas as well as for PCNF formulas, based on clause elimination techniques that remove *tautological*, *subsumed*, *blocked*, and *covered* clauses. For each of these elimination techniques, we consider the *plain* as well as what we call the *asymmetric* variant. For (plain) tautology elimination (TE), we introduce *asymmetric tautology elimination* (ATE). For (plain) subsumption elimination (SE), we have the *asymmetric* variant ASE, and for (plain) blocked clause elimination (BCE), the *asymmetric* variant ABCE, respectively. Additionally, we develop a novel family (including the plain (CCE) and asymmetric (ACCE) variants) of so-called *covered clause elimination* procedures. For the context of QSAT, we propose natural liftings of these CNF-level clause elimination procedures. While for the redundancy properties *tautology* and *subsumed* the corresponding CNF-level clause elimination procedures are directly applicable by ignoring the quantifier prefix, the PCNF-level procedures based on the properties *blocked* and *covered* require more care as it is necessary to take the quantifier prefix into account.

Due to somewhat involved definitions, we postpone the definitions for clause elimination procedures based on the covered *property until Section 6 in which we analyze there procedures in detail. Similarly, liftings of the blocked and covered clause elimination procedures to QSAT are defined in Sections 5 and 6, respectively.* However, let us already here define the concept of *asymmetric* clause elimination procedures, generalizing any (plain) clause elimination procedure. This is motivated by the fact that, as will be shown, the asymmetric variants can achieve more simplification than the plain procedures.

The asymmetric variant of a clause elimination technique relies on the clause extension rule of *asymmetric literal addition* (ALA).

Definition 8 (Asymmetric Literal Addition)**.** *Given a clause C and a CNF formula F, literal l is an* asymmetric literal *for clause C if there exist a clause $C' \in F_{\bar{l}} \setminus \{C\}$ such that $C' \setminus \{\bar{l}\}$ subsumes C. For a clause C and a CNF formula F, $\mathrm{ALA}(F, C)$ denotes the* unique *clause resulting from repeating the following until fixpoint: If $l_1, \ldots, l_k \in C$ and there is a clause $(l_1 \vee \ldots \vee l_k \vee l) \in F \setminus \{C\}$ for some literal l, let $C := C \cup \{\bar{l}\}$.*

Example 4. *Consider the formula* $F = (a \vee b \vee c) \wedge (a \vee b \vee d) \wedge (a \vee c \vee \bar{d})$. $\mathrm{ALA}(F, (a \vee b \vee c))$ *first adds the asymmetric literals* $\bar{d}$ *and* d *using* $(a \vee b \vee d)$ *and* $(a \vee c \vee \bar{d})$*, respectively. Afterwards, it can add the asymmetric literals* $\bar{a}$ *and* $\bar{b}$ *using* $(a \vee b \vee d)$ *and* $\bar{c}$ *using* $(a \vee c \vee \bar{d})$*. The fixpoint of* $\mathrm{ALA}(F, (a \vee b \vee c))$ *is* $(a \vee \bar{a} \vee b \vee \bar{b} \vee c \vee \bar{c} \vee d \vee \bar{d})$.

It is easy to show that the replacement of a clause C occurring in a CNF F by $\mathrm{ALA}(F, C)$ preserves logical equivalence regardless of whether F is a quantifier free propositional formula or the matrix of a QBF. In other words, ALA is agnostic of the quantifier prefix.

As concrete examples, asymmetric tautology elimination, asymmetric subsumption elimination, and asymmetric blocked clause elimination are defined as follows.

Definition 9. *A clause* C *is an* asymmetric tautology *if and only if* $\mathrm{ALA}(F, C)$ *is a tautology.* Asymmetric tautology elimination *(*ATE*) repeats the following until fixpoint: If there is a clause* $C \in F$ *for which* $\mathrm{ALA}(F, C)$ *is a tautology, let* $F := F \setminus \{C\}$.

Example 5. *Consider the formula* $F = (a \vee b \vee c) \wedge (a \vee b \vee d) \wedge (a \vee c \vee \bar{d})$ *from Example 4. For this* F, $\mathrm{ALA}(F, (a \vee b \vee c)) = (a \vee \bar{a} \vee b \vee \bar{b} \vee c \vee \bar{c} \vee d \vee \bar{d})$ *is a tautology, and hence removed by* ATE *from* F.

As stated by the following lemma, ATE performs what could be called *asymmetric branching* on clauses—referred to as *UP-redundancy* by Fourdrinoy et al. (2007a, 2007a) and Piette et al. (2008)—which is used for example in the technique of *clause distillation* (Jin & Somenzi, 2005). This gives an alternative characterization of ATE in terms of Boolean constraint propagation.

Lemma 1. $\mathrm{ALA}(F, C)$ *is a tautology if and only if* BCP *on* $(F \setminus \{C\}) \cup \bigcup_{l \in C} \{(\bar{l})\})$ *derives a conflict.*

The definitions of asymmetric subsumption elimination and asymmetric blocked clause elimination are analogous to that of asymmetric tautology elimination.

Definition 10. Asymmetric subsumption elimination *(*ASE*) repeats the following until fixpoint: If there is a clause* $C \in F$ *for which* $\mathrm{ALA}(F, C)$ *is subsumed in* F*, let* $F := F \setminus \{C\}$.

Example 6. *Consider the formula* $F_{\mathrm{ASE}} = (a \vee b \vee c) \wedge (a \vee b \vee d) \wedge (a \vee c \vee \bar{d})$. ASE *can remove* $(a \vee b \vee c)$ *from* F*, because* $\mathrm{ALA}(F_{\mathrm{ASE}}, (a \vee b \vee c)) = (a \vee b \vee c \vee d \vee \bar{d})$ *is subsumed by* $(a \vee b \vee d)$ *or* $(a \vee c \vee \bar{d})$.

Definition 11. *For a given CNF formula* F*, a clause* $C \in F$ *is* asymmetric(ally) blocked *if* $\mathrm{ALA}(F, C)$ *is blocked w.r.t.* F*.* Asymmetric blocked clause elimination *(*ABCE*) repeats the following until fixpoint: If there is an asymmetric blocked clause* $C \in F$ *for which* $\mathrm{ALA}(F, C)$ *is blocked w.r.t.* F*, let* $F := F \setminus \{C\}$.

Example 7. *Consider the formula* $F_{\mathrm{ABCE}} = (\bar{a} \vee b \vee c) \wedge (b \vee c \vee \bar{d}) \wedge (a \vee d) \wedge (\bar{b} \vee \bar{d}) \wedge (\bar{c} \vee \bar{d})$. ABCE *can eliminate* $(\bar{a} \vee b \vee c)$*, because* $\mathrm{ALA}(F_{\mathrm{ABCE}}, (\bar{a} \vee b \vee c)) = (\bar{a} \vee b \vee c \vee d)$ *which* b *and* c *are blocking literals. Also,* ABCE *can remove all clauses in* F_{ABCE}.

It turns out that clause elimination procedures that preserve logical equivalence in the case of SAT, do not have to consider any information on quantifier ordering in the case of QSAT, i.e., these rules are the same for SAT and QSAT. Procedures which preserve only satisfiability equivalence in SAT, however, would become unsound if the quantifier ordering were ignored and, therefore, the restrictions imposed by the prefix have to be considered in the definition of such procedures. This results in the quantified variants of blocked clause elimination (QBCE) and asymmetric blocked clause elimination (AQBCE), both detailed in Section 5, as well as the quantified variants of covered clause elimination (QCCE) and asymmetric covered clause elimination (AQCCE) detailed in Section 6).

3.2 Analyzing Clause Elimination Procedures

We will present detailed analysis on the relationships between the considered clause elimination procedures, in terms of the achieved level of simplification (relative reduction power), the level of equivalence maintained by the procedures (in terms of the sets of models of original and simplified formulas), confluence (i.e., whether the procedures have a unique fixpoint), as well as the level of constraint propagation maintained when applying the procedures. We will now formally define these concepts and give an overview of the results. Detailed proofs for these results are presented in Sections 4–6. Analysis of the procedures in terms of these properties are later (in Section 8) complemented with an empirical evaluation on the effect of the clause elimination procedures on runtimes of state-of-the-art SAT and QSAT solvers.

A relevant aspect of simplification techniques is the question of how much a specific technique reduces the size of CNF (and PCNF) formulas. In this paper we analyze the *relative reduction power* of the considered clause elimination procedures based on the clauses removed by the procedures. For this we apply the following natural definition of reduction power.

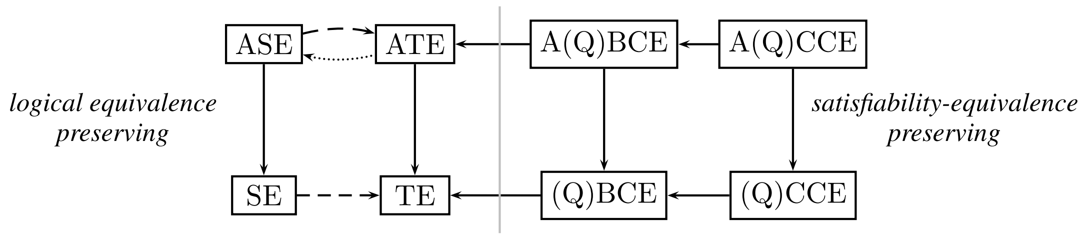
Definition 12 (Relative reduction power)**.** *Assume two clause elimination procedures S_1 and S_2 that take as input an arbitrary CNF formula F and each produce as output a CNF formula that consists of a subset of F that is satisfiability-equivalent to F.*

- S_1 at least as powerful as S_2 *if, for any F and any output $S_1(F)$ and $S_2(F)$ of S_1 and S_2 on input F, respectively, we have that $S_1(F) \subseteq S_2(F)$;*
- S_2 *is* not as powerful as S_1 *if there is an F for which there are outputs $S_1(F)$ and $S_2(F)$ of S_1 and S_2, respectively, such that $S_1(F) \subset S_2(F)$;*
- S_1 *is* more powerful than S_2 *if*
  - *(i) S_1 is at least as powerful as S_2, and*
  - *(ii) S_2 is not as powerful as S_1.*

Our definition of relative reduction power takes into account *non-confluent* elimination procedures, that is, procedures that do not generally have a unique fixpoint and that may thus have more than one possible output for a given input. It should be noted that the result of a non-confluent simplification procedure can be very unpredictable due to the non-uniqueness of results.

The definition of relative reduction power extends naturally to QSAT by considering the size reduction of the matrix of a QBF. Hence, for natural liftings[2] QS_1 and QS_2 of two CNF-level clause elimination procedures S_1 and S_2 to QBFs, it will hold that if S_1 is more powerful than S_2, then QS_1 is more powerful than QS_2.

2. A lifting QS_1 of S_1 is considered natural if QS_1 behaves exactly like S_1 when restricted to QBFs without universally quantified variables.



SE: Subsumption elimination (same for SAT and QSAT).
TE: Tautology elimination (same for SAT and QSAT).
ASE: Asymmetric subsumption elimination (same for SAT and QSAT).
ATE: Asymmetric tautology elimination (same for SAT and QSAT).
(Q)BCE: (Quantified) blocked clause elimination.
(Q)CCE: (Quantified) covered clause elimination.
A(Q)BCE: Asymmetric (quantified) blocked clause elimination.
A(Q)CCE: Asymmetric (quantified) covered clause elimination.

Figure 1: Relative reduction power hierarchy of clause elimination procedures. An edge from X to Y means that X is more powerful than Y. A solid edge means no corner cases. A dashed edge means that the property does not hold for the corner case that the formula contains only tautologies. A dotted edge means that the property does not hold for the corner case that the formula contains the empty clause. A missing edge from X to Y means that X is not as powerful as Y. However, notice that transitive edges are missing from the figure for clarity. The Q in the prefix of a clause elimination technique indicates that there are differences in the QSAT and SAT variant.

Our analysis results in a relative reduction power hierarchy (Figure 1) for the considered elimination procedures. For example, we show that for each of the known *plain* techniques, the *asymmetric* variants are more powerful. In this sense, the novel variants are proper generalizations of the known plain techniques. It also turns out that the most powerful technique is the asymmetric variant of covered clause elimination. The figure is slightly different from earlier work (Heule et al., 2010). The changes are based on our renewed view of subsumption: a tautology is subsumed by any clause.[3] This view is justified by the fact that if $C \in F$ subsumes another clause $C' \in F$ in a CNF F due to $C \subset C'$, then C' is logically entailed by F. Since a tautological clause $C' \in F$ is trivially entailed by F, we regard C' to be subsumed by any other clause $C \in F$. We note that there is only one single corner case: if a formula contains only tautologies, then tautology elimination can remove all tautologies, while subsumption elimination can remove all, but one. Using this definition, subsumption elimination techniques are as powerful as tautology elimination techniques.

Additionally, we consider the properties listed in Table 1 for further analysing the clause elimination procedures for SAT. It is easy to see that TE and SE are confluent and BCP-preserving, and also that for any CNF formula F, $\mathrm{TE}(F)$ and $\mathrm{SE}(F)$ are logically equivalent to F. Furthermore, for any QBF $\Pi.F$, tautology elimination and subsumption elimination, as well as their asymmetric

3. Donald Knuth convinced us of this view in personal communication on July 21, 2014.

Table 1: Properties of clause elimination procedures for SAT.

	Preserves logical eq.	BCP-preserving	Confluent
Subsumption-based			
SE	yes	yes	yes
ASE	yes	no	no
Tautology-based			
TE	yes	yes	yes
ATE	yes	no	no
Blocked clause			
BCE	no	no	yes
ABCE	no	no	no
Covered clause			
CCE	no	no	yes
ACCE	no	no	no

variants, do not use any information on the variable ordering, i.e., $S(\Pi.F) := \Pi.S(F)$ for any $S \in \{\mathrm{TE}, \mathrm{SE}, \mathrm{ATE}, \mathrm{ASE}\}$. It also holds that BCE is confluent (Järvisalo et al., 2010).

While each of the techniques preserves satisfiability (and are thus sound), it turns out that the variants of blocked clause elimination and covered clause elimination do not preserve logical equivalence; this is the motivation for demonstrating in Section 7 how one can efficiently reconstruct original solutions based on satisfying assignments for CNFs simplified using these variants. A further property of simplification techniques is BCP-preservance, which implies that relevant unit propagation (restricted to the remaining variables in the simplified CNF formula) possible in the original CNF is also possible in the simplified CNF under any partial assignment. This property is solver-related and very much practically relevant, since BCP is an integral part of a vast majority of SAT solvers today.

Definition 13 (BCP-preserving)**.** *For a formula F, a preprocessing procedure S* preserves BCP on F *if under any partial assignment τ over the variables in F and for any formula $S(F)$ resulting from applying S on F, we have that*

(i) for any literal l occurring in $S(F)$, $(l) \in \mathrm{BCP}(F, \tau)$ implies $(l) \in \mathrm{BCP}(S(F), \tau)$

(ii) $\emptyset \in \mathrm{BCP}(F, \tau)$ implies $\emptyset \in \mathrm{BCP}(S(F), \tau)$ (the empty clause is obtained, that is, BCP *derives a conflict).*

S is BCP-preserving *if S preserves* BCP *on every CNF formula.*

Notice that our definition is similar to *deductive power* as defined by Han and Somenzi (2007). Also notice that BCP-preserving implies that logical equivalence is also preserved. Interestingly, it turns out that BCP-preserving is quite a strict property, as only the plain SE and TE have it.

We note that if a procedure S is not BCP-preserving, and a procedure S' is more powerful than S, then we immediately have that S' is not BCP-preserving. Similarly, if S does not preserve logical equivalence, then S' does not preserve logical equivalence either. Furthermore, by viewing CNF formulas as PCNF formulas where all variables are existentially quantified, showing that a CNF-level procedure S is not BCP-preserving or is not confluent typically directly implies the same negative result for the lifting of S to PCNF formulas. Indeed, this is the case for the procedures considered in this article. Furthermore, since the quantifier structure does not impose restrictions on the behavior of $S \in \{\mathrm{TE}, \mathrm{SE}, \mathrm{ATE}, \mathrm{ASE}\}$, the positive results on BCP-preservance and confluence also directly translate to the level of PCNF formulas. Hence we focus the analysis of the procedures from these perspectives on the CNF-level.

In the following, we proceed by giving detailed analysis of each of the variants of tautology, subsumption, blocked clause, and covered clause based elimination procedures by considering equivalence preserving techniques first followed by a discussion of satisfiability preserving techniques. Finally, experimental results on the practical effectiveness of the procedures are presented in Section 8.

4. Logical Equivalence Preserving Clause Elimination Techniques

We start our analysis by shortly considering clause elimination procedures which preserve logical equivalence, namely, the well-known tautology and subsumption elimination, and their asymmetric variants.

Lemma 2. ATE *is more powerful than* TE.

Proof. ATE is at least as powerful as TE due to $C \subseteq \mathrm{ALA}(F, C)$: if C is a tautology, so is $\mathrm{ALA}(F, C)$. Moreover, let $F = (a \vee b) \wedge (\bar{b} \vee c) \wedge (a \vee c)$. Since $\mathrm{ALA}(F, (a \vee c)) = (a \vee \bar{a} \vee b \vee \bar{b} \vee c \vee \bar{c})$, ATE can remove $(a \vee c)$ from F, in contrast to TE. $\square$

Proposition 1. ATE *is not confluent.*

Proof. Consider the formula $F = (\bar{a} \vee b) \wedge (\bar{a} \vee c) \wedge (a \vee \bar{c}) \wedge (\bar{b} \vee c) \wedge (b \vee \bar{c})$. Now, $\mathrm{ALA}(F, (\bar{a} \vee b)) = \mathrm{ALA}(F, (\bar{a} \vee c)) = \mathrm{ALA}(F, (b \vee \bar{c})) = (a \vee \bar{a} \vee b \vee \bar{b} \vee c \vee \bar{c})$. ATE can remove either $(\bar{a} \vee b)$ or both $(\bar{a} \vee c), (b \vee \bar{c})$. $\square$

Proposition 2. *For any CNF formula F,* $\mathrm{ATE}(F)$ *is logically equivalent to F.*

Proof. For any clause C removed by ATE, $(F \setminus \{C\}) \cup \bigcup_{l \in C} \{(\bar{l})\}$ is unsatisfiable. This implies that $F \setminus \{C\} \models C$, that is, $F \setminus \{C\}$ logically entails C. $\square$

Proposition 3. ATE *is not* BCP*-preserving.*

Proof. Consider the “standard” CNF translation of x = If-Then-Else(c, t, e) as the formula

$$(\bar{x} \vee \bar{c} \vee t) \wedge (x \vee \bar{c} \vee \bar{t}) \wedge (\bar{x} \vee c \vee e) \wedge (x \vee c \vee \bar{e}) \wedge (x \vee \bar{e} \vee \bar{t}) \wedge (\bar{x} \vee e \vee t).$$

Notice that ATE can remove $(x \vee \bar{e} \vee \bar{t})$ and $(\bar{x} \vee e \vee t)$. However, after removing these clauses, BCP will no longer assign x to **t** under the truth assignment $\tau(e) = \tau(t) =$ **f**. Also, BCP will no longer assign x to **f** under the truth assignment $\tau(e) = \tau(t) =$ **t**. $\square$

Next we consider the asymmetric variants of tautology elimination and subsumption elimination.

Proposition 4. ASE *is more powerful than* SE.

Proof. ASE is at least as powerful as SE since for any CNF formula F, (i) for every clause $C \in F$, $C \subseteq \mathrm{ALA}(F, C)$, and (ii) if C is subsumed then any clause $C' \supseteq C$ is subsumed. Moreover, consider the formula $F = (a \vee b \vee c) \wedge (a \vee b \vee d) \wedge (b \vee \bar{c})$. In contrast to SE, ASE can remove $(a \vee b \vee d)$, because $\mathrm{ALA}(F, (a \vee b \vee d)) = (a \vee b \vee c \vee d)$ is subsumed by $(a \vee b \vee c)$. $\square$

In connection with Proposition 4, we note that, under the *UP-redundancy* terminology, it was equivalently observed by Fourdrinoy et al. (2007a) that, essentially, removing asymmetric tautologies until fixpoint produces a formula which is closed under subsumption elimination.

Lemma 3. ATE *is at least as powerful as* ASE*, except in the corner case that the formula contains the empty clause.*

Proof. Consider the corner case that a formula contains the empty clause. In the case, ASE can remove all clauses other than the empty clause. However, let $F := \emptyset \wedge C$ such that C is not a tautology. ATE cannot remove C, while ASE can.

To see that ATE is at least as powerful as ASE in the other cases, consider the following. If there is a clause $C \in F$ for which $\mathrm{ALA}(F, C)$ is subsumed by $C' \in F \setminus \{C\}$, then $\mathrm{ALA}(F, C)$ is a tautology: say $\mathrm{ALA}(F, C)$ is subsumed by $C' = (l_1 \vee \ldots \vee l_k)$. By the definition of ALA, $\bar{l}_1, \ldots, \bar{l}_k \in \mathrm{ALA}(F, C)$. $\square$

Lemma 4. ASE *is at least as powerful as* ATE*, except in the corner case that the formula consists of only tautologies.*

Proof. Consider the corner case that a formula consists of only tautologies. In this case, ATE can remove all clauses. However, ASE can remove at most all but one clause.

To see that ASE is at least as powerful as ATE, consider the following. Any tautology is subsumed by any other clause. That also holds for asymmetric tautologies. Hence as long as there is at least one clause available for subsumption (which is the case if the formula contains at least one non-tautological clause), ASE is at least as powerful as ATE. $\square$

Proposition 5. ASE *is not confluent.*

Proof. By replacing ATE with ASE in the proof of Proposition 1. $\square$

Proposition 6. *For any CNF formula F,* $\mathrm{ASE}(F)$ *is logically equivalent to F.*

Proof. For any clause C removed by ASE, $(F \setminus \{C\}) \cup \bigcup_{l \in C}\{(\bar{l})\}$ is unsatisfiable. This implies that $F \setminus \{C\} \models C$, that is, $F \setminus \{C\}$ logically entails C. $\square$

By replacing ATE with ASE in the proof of Lemma 3 we have the following.

Proposition 7. ASE *is not* BCP*-preserving.*

5. Clause Elimination Procedures based on Blocked Clauses

As a third family of clause elimination procedures considered in this paper, we now analyze procedures that eliminate blocked clauses (Kullmann, 1999) and present generalizations thereof to QSAT.

5.1 Blocked Clause Elimination for SAT

We start with the "plain" variant of blocked clause elimination, BCE.

Proposition 8. BCE *is more powerful than* TE.

Proof. To see that BCE is at least as powerful as TE, notice that for any tautology C, $D \supseteq C$ is also a tautology. In this case $D = C \cup C' \setminus \{\bar{l}\}$ with C' and $\bar{l}$ from Definition 7 of BCE. Moreover, consider the formula $F := (a)$. BCE can remove (a) from F, in contrast to TE. $\square$

Proposition 9. *For some CNF formula F,* $\mathrm{BCE}(F)$ *is not logically equivalent to F.*

Proof. Recall the formula F_{BCE} from Example 3. Consider the truth assignment τ with $\tau(v_1) = \tau(v_2) = \tau(w_3) = \mathbf{t}$ and $\tau(v_3) = \tau(w_1) = \tau(w_2) = \mathbf{f}$. Although τ satisfies $\mathrm{BCE}(F_{\mathrm{BCE}})$, the clause $(\bar{v}_1 \vee \bar{v}_2)$ in F_{BCE} is falsified by τ. $\square$

Proposition 10. $\mathrm{BCE}(F)$ *is not BCP-preserving.*

Proof. Follows from the fact that BCE does not preserve logical equivalence (Proposition 9). $\square$

We now turn to the asymmetric variant of blocked clause elimination which turns out to be more powerful that BCE.

Proposition 11. *Removal of an asymmetric blocked clause preserves satisfiability.*

Proof. Follows from the facts that F is logically equivalent to $(F \setminus \{C\}) \cup \{\mathrm{ALA}(F, C)\}$ and that BCE preserves satisfiability. $\square$

Lemma 5. ABCE *is more powerful than (i)* BCE*, and (ii)* ATE.

Proof. ABCE is at least as powerful as BCE due to $C \subseteq \mathrm{ALA}(F, C)$: if C is a tautology, then $\mathrm{ALA}(F, C)$ is a tautology. ABCE is at least as powerful as ATE since tautologies are blocked clauses. Moreover, recall that in Example 7, ABCE could eliminate *all* clauses from F_{ABCE}. Neither BCE nor ATE can remove any clause from F_{ABCE}. $\square$

Proposition 12. ABCE *is not confluent.*

Proof. Let $F = (\bar{a} \vee b) \wedge (\bar{a} \vee c) \wedge (a \vee \bar{d}) \wedge (\bar{b} \vee d) \wedge (\bar{c} \vee d)$. F contains four asymmetric blocked clauses: $\mathrm{ALA}(F, (\bar{a} \vee b)) = (\bar{a} \vee b \vee \bar{c} \vee \bar{d})$ with blocking literal b, $\mathrm{ALA}(F, (\bar{a} \vee c)) = (\bar{a} \vee \bar{b} \vee c \vee \bar{d})$ with blocking literal c, $\mathrm{ALA}(F, (\bar{b} \vee d)) = (a \vee \bar{b} \vee c \vee d)$ with blocking literal $\bar{b}$, and $\mathrm{ALA}(F, (\bar{c} \vee d)) = (a \vee b \vee \bar{c} \vee d)$ with blocking literal $\bar{c}$. ABCE removes either $(\bar{a} \vee b)$ and $(\bar{b} \vee d)$, or $(\bar{a} \vee c)$ or $(\bar{c} \vee d)$ from F. $\square$

Replacing BCE with ABCE in the proof of Proposition 9, we have the following.

Proposition 13. *For some CNF formula F,* $\mathrm{ABCE}(F)$ *is not logically equivalent to F.*

5.2 Quantified Blocked Clause Elimination

In the following, we generalize the notion of blocked clauses and blocked clause elimination (BCE) for QSAT. We prove that blocked clauses can be removed also in the case of QSAT, and discuss differences to propositional logic.

Definition 14. *A literal l with* quant*(l) = $\exists$ in a clause $C \in F$ of a QBF $G = Q_1 \dots Q_n.F$ is called a* quantified blocking literal *if for all $C' \in F$ with $\bar{l} \in C'$, a literal l' with $l' \leq l$ exists such that $l', \bar{l}' \in C \cup (C' \setminus \{\bar{l}\})$. A clause is* quantified blocked *if it contains a quantified blocking literal.*

For a QBF G in PCNF, *quantified blocked clause elimination* repeats the removal of quantified blocked clauses from G until fixpoint. The resulting QBF is denoted by $\mathrm{QBCE}(G)$. Our definition of quantified blocking literals slightly differs from the original definition (Biere et al., 2011) which uses $l', \bar{l}' \in C \otimes_l C'$ instead of $l', \bar{l}' \in C \cup (C' \setminus \{\bar{l}\})$. Our definition includes the case that either C or C' is a tautology, i.e., when $C \otimes_l C'$ is undefined. Apart from this, the definitions are equivalent.

In contrast to propositional logic, there are two restrictions on the selection of quantified blocking literals. A blocking literal has to be existential and the literals responsible for the tautology in a resolvent have to be of smaller level than the blocking literal. Without these restrictions, quantified blocked clause elimination would not be sound, as illustrated by the following examples.

Example 8. *Both clauses in the satisfiable QBF $G = \forall x \exists y.((x \vee \bar{y}) \wedge (\bar{x} \vee y))$ are quantified blocked clauses since the existential literals $\bar{y}$ and y are quantified blocking literals in the first and second clause, respectively. Note that $x < y$ in the prefix ordering. Let $G' = \exists x \forall y.((x \vee \bar{y}) \wedge (\bar{x} \vee y))$ be obtained from G by changing the quantifiers of x and y in the prefix. The QBF G' is unsatisfiable. No clause in G' is quantified blocked since x is existential and $x < y$. If we ignored the condition on the level of a quantified blocking literal, then erroneously both clauses in G' would be considered as quantified blocked, and removing any clause of G' results in a satisfiable QBF.*

Example 9. *Consider the unsatisfiable QBF $\exists y \forall x \exists z.((y \vee \bar{x} \vee z) \wedge (\bar{y} \vee x \vee z) \wedge (y) \wedge (\bar{z}))$. By definition, literals of the universal variable x cannot be quantified blocking. If we ignored the quantifier type of x then erroneously $\bar{x}$ and x in the first and second clause, respectively, would be considered as quantified blocking literals. Note that the condition on the quantifier level holds, that is, $y < x$ where y is responsible for the tautological resolvent of the two clauses containing x. However, removing the second clause, which is erroneously considered as quantified blocked, results in a satisfiable QBF.*

As the following theorem shows, quantified blocked clauses contain redundant information only, and may therefore be removed from the formula.

Theorem 1. *Let $G = Q_1 \dots Q_n.(F \cup \{C\})$ be a QBF and let C be a quantified blocked clause in G with blocking literal l. Then G and $Q_1 \dots Q_n.F$ are equivalent.*

Proof. Let C be a quantified blocked clause with the quantified blocking literal l with $\mathsf{var}(l) \in Q_i$, $i \leq n$. The direction $G \Rightarrow Q_1 \dots Q_n.F$ trivially holds. We show $Q_1 \dots Q_n.F \Rightarrow G$ by induction over $q = |Q_1 \dots Q_{i-1}|$.

In the base case, we have $q = 0$, i.e., $\mathsf{var}(l) \in Q_1$ with $\mathsf{quant}(Q_1) = \exists$. The same argument as in SAT (Kullmann, 1999) applies: let τ be a satisfying assignment for F, i.e., for each $C' \in F$ there exists a literal l' such that $\tau(l') = \mathbf{t}$. If τ satisfies C, the implication $Q_1.F \Rightarrow G$ holds, otherwise we construct a satisfying assignment τ' for $F \cup \{C\}$ as follows. Let $\tau'(l') = \tau(l')$ for $l' \neq l$ and $\tau'(l) = \mathbf{t}$. Then τ' satisfies not only C but also all other clauses $C' \in F$. If $\bar{l} \in C'$, there exists a literal $l'' \neq l$ such that $l'' \in C$ and $\bar{l}'' \in C'$, with $\tau(l'') = \tau(C) = \tau'(l'') = \mathbf{f}$ and thus $\tau'(C') = \tau'(\bar{l}'') = \mathbf{t}$. Note that $l'' \in Q_1$ due to the restriction $l'' \leq l$.

For the induction step, assume $q > 0$. Let h be a literal with $\mathsf{var}(h) = y$ and $y \in Q_1$. Note that $\mathsf{var}(l) \neq y$. We show that $Q_1 \backslash \{y\} \ldots Q_n.F[h] \Rightarrow G[h]$. The rest follows from lifting the implication over the conjunction that defines the semantics of universal quantification if $\mathsf{quant}(Q_1) = \forall$, and, respectively, over the disjunction that defines the semantics of the existential quantification if $\mathsf{quant}(Q_1) = \exists$. Three cases have to be considered for showing that $C[h]$ is a blocked clause or removed in $F[h]$.

1. $h \in C$. Then C is removed from $G[h]$.
2. $h \notin C$ and $\bar{h} \notin C$. Consequently, $C[h] = C$. Furthermore, C is still a quantified blocked clause in $G[h]$, since h was not used to make a resolvent on l tautological. Then the induction hypothesis is applicable.
3. $\bar{h} \in C$. Consequently, $C[h] = C \backslash \{\bar{h}\}$ which is a quantified blocked clause in $G[h]$, because each clause C' with $h, \bar{h} \in C \otimes_l C'$ is removed from $G[h]$, and other clauses C' with $k, \bar{k} \in C \otimes_l C'$ and $y \neq \mathsf{var}(k)$ still produce tautological resolvents with C on l. Note that $l \in C[h]$ since $l \neq \bar{h}$.

□

Theorem 2. *The application of* $\mathrm{QBCE}(G)$ *on a QBF G is confluent.*

Proof. The argument is similar as for propositional logic (see Section 5). □

For the soundness of quantified blocked clause elimination for QSAT, the level of the blocking literal must be equal or higher than the level of the literal making the resolvent tautological, as the following example illustrates.

Example 10. *An extended example related to Example 8 where universal reduction is not applicable to any clause is given by the unsatisfiable QBF*

$$G' = \exists x \forall y \exists z.((x \vee \bar{z}) \wedge (\bar{x} \vee z) \wedge (y \vee \bar{z}) \wedge (\bar{y} \vee z)).$$

The first two clauses in G' encode that x and z are equivalent, and the last two clauses encode that y and z are equivalent. The variable z prohibits the application of universal reduction. In the first two clauses x and $\bar{x}$, respectively, are not quantified blocking literals because $x < z$. If these clauses were erroneously considered to be blocked and removed, then the resulting QBF would be satisfiable.

Quantified blocked clauses may be eliminated from a formula without changing its truth value, because they contain redundant information only. Hence, quantified blocked clause elimination is applied in order to remove clauses from a QBF which may reduce the number of variables occurring in the formula too. The following properties established for SAT (Kullmann, 1999; Heule et al., 2010), also hold for QSAT. For the sake of compactness, we omit the prefix "quantified" if no confusion arises.

1. Formulas which are smaller with respect to their number of clauses potentially contain more blocked clauses. If the matrix of a QBF G_1 is a subset of the matrix of the QBF G_2 then there might be clauses which are blocked in G_1, but not in G_2. If there is a clause C which is blocked in G_2, but not in G_1, then $C \notin G_1$.

2. From the statement above, it follows immediately that QBCE has a unique fixpoint. If a clause C is blocked in a QBF G, then any clause C' with $C \neq C'$ blocked in G is also blocked in $G \backslash \{C\}$.

3. If a clause C is subsumed by a blocked clause C', i.e., $C' \subseteq C$, then C is also a blocked clause. Obviously, the other direction does not hold.

4. Clauses containing an existential pure literal are blocked. The pure literal is the blocking literal. In fact, QBCE may be considered as a generalization of pure literal elimination rule for existentially quantified variables. However, the elimination of pure literals which are universally quantified is not simulated by QBCE. In general, pure literal elimination of universal literals does not eliminate whole clauses, but only single literals.

5. If the clauses $C_1 \ldots C_n$ are the only clauses of a QBF G which contain the literal l, then a clause C with $\bar{l} \in C$ is blocked if for each clause C_i, the clause C contains a literal l_i with $\bar{l}_i \in C_i$ and $l_i < l$. In particular, if a QBF G contains an equivalence of the form $(l, \bar{l}_1, \ldots, \bar{l}_n), (\bar{l}, l_1), \ldots, (\bar{l}, l_n)$ and l occurs in no other than these clauses, then the equivalence may be removed due to QBCE.

The fifth property indicates that QBCE eliminates equivalences under certain conditions. In fact, like BCE in SAT (Järvisalo et al., 2012a), QBCE achieves structure-based simplifications defined for circuit-based representations purely on the PCNF-level, without explicit knowledge on the structure of the original representation.

6. Covered Clause Elimination Procedures

As the final family of clause elimination procedures considered in this paper, we now introduce and analyze CNF and PCNF-level procedures that eliminate what we call *covered clauses*.

Covered clause elimination is based on successively adding certain literals to a clause C in a CNF $F' = F \wedge \{C\}$ in a satisfiability-preserving way. Adding a literal l to C produces the extended clause $C' = C \cup \{l\}$ which replaces C in F' to obtain $F' = F \wedge \{C'\}$. If C' becomes blocked due to adding a literal l then C' can be removed from F' by BCE, effectively eliminating one clause of F'. The literals added to a clause C by extension steps are called *covered literals*. These literals are determined by inspecting all the clauses which, when resolved with C, result in a non-tautological resolvent. A clause that becomes blocked by adding covered literals is called a covered clause. For the application of covered clause elimination in practice, a clause C is extended by covered literals only tentatively. If the extended clause C' does not become blocked eventually, then the added literals are discarded and the original clause C is restored.

In the following, we formally define the set of covered literals, which can safely be used to extend clauses in a CNF. Then we introduce covered clause elimination for SAT and QSAT and analyze its properties. Similar to BCE, covered clause elimination does not preserve logical equivalence. Therefore, in Section 7 we present an algorithm to reconstruct solutions to CNFs where covered clauses have been eliminated.

6.1 Covered Clause Elimination Procedures for SAT

Given a CNF formula F, a clause C, and a literal $l \in C$, the set of *resolution candidates* of C w.r.t. l is

$$\mathrm{RC}(F, C, l) := \{C' \mid C' \in F_{\bar{l}} \text{ and } C \otimes_l C' \text{ is not a tautology}\}.$$

In words, the set $\mathrm{RC}(F, C, l)$ of resolution candidates consists of clauses C' that contain the opposite literal of l, meaning that these clauses can be resolved with C on the literal l. Notice that every clause in $\mathrm{RC}(F, C, l)$ contains the literal $\bar{l}$. If $\mathrm{RC}(F, C, l) = \emptyset$, then C is blocked w.r.t. F. The literals apart from $\bar{l}$ which occur in all clauses of $\mathrm{RC}(F, C, l)$ form the *resolution intersection* $\mathrm{RI}(F, C, l)$ of l and C w.r.t. F, formally defined as

$$\mathrm{RI}(F, C, l) := \left(\bigcap \mathrm{RC}(F, C, l)\right) \setminus \{\bar{l}\}.$$

In words, the resolution intersection $\mathrm{RI}(F, C, l)$ is the set of literals (apart from $\bar{l}$) which occur in each clause in the resolution candidate set $\mathrm{RC}(F, C, l)$. Given a CNF formula F, a clause $C \in F$, and a literal $l \in C$, we say that l *covers* the literals in $\mathrm{RI}(F, C, l)$ (w.r.t. F and C). A literal l' is *covered* by $l \in C$ if $l' \in \mathrm{RI}(F, C, l)$. A literal $l \in C$ is *covering* w.r.t. F and C if l covers at least one literal, that is, $\mathrm{RI}(F, C, l) \neq \emptyset$.

Example 11. *Consider the formula*

$$F_{\mathrm{CLA}} = (a \vee b \vee c) \wedge (a \vee \bar{b} \vee d) \wedge (a \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee b \vee \bar{d}) \wedge (\bar{a} \vee c \vee d)$$

which is also visualized as a resolution graph in Figure 2. The resolution graph is constructed as follows. The clauses are the vertices and vertices are connected if and only if resolution between the two clauses results in a non-tautological resolvent. In other words, vertices are only connected if they have exactly one clashing literal. The edges have a label which shows the corresponding variable of the clashing literal pair.

BCE *cannot remove a clause* F_{CLA} *because for each literal occurrence in* F_{CLA} *there exists a non-tautological resolvent. Figure 2 illustrates this by having for each literal of each clause at least one edge. Now,* $\mathrm{RC}(F_{\mathrm{CLA}}, (a \vee b \vee c), b) = \{(a, \bar{b}, d)\}$*, so* $\mathrm{RI}(F_{\mathrm{CLA}}, (a \vee b \vee c), b) = \{a, d\}$*. In other words, literal* b *in* $(a \vee b \vee c)$ *covers the literals* a *and* d*. As discussed in the following, adding* d *to* $(a \vee b \vee c)$ *preserves satisfiability.* [4]

After the addition of d *to* $(a \vee b \vee c)$*, several edges disappear. It no longer holds that each literal occurrence has a corresponding edge. All literals that do not have an edge, (for example,* $\bar{c}$ *in* $(a \vee \bar{c} \vee \bar{d})$*), have become blocking literals.*

Lemma 6. *For any CNF formula* F*, clause* $C \in F$*, and literal* $l \in C$*, it holds that replacing* C *by* $C \cup \mathrm{RI}(F, C, l)$ *in* F *preserves satisfiability.*

Proof. For any literal $l \in C$ it holds that $\mathrm{VE}(F, l) = \mathrm{VE}((F \setminus \{C\}) \cup \{C \cup \mathrm{RI}(F, C, l)\}, l)$, where $\mathrm{VE}(F, l)$ denotes the CNF formula resulting from variable eliminating the variable of the literal l from F (more formally, $\mathrm{VE}(F, l) = (F_l \otimes F_{\bar{l}}) \cup (F \setminus (F_l \cup F_{\bar{l}}))$, where F_l and $F_{\bar{l}}$ consist of the clauses in F that contain l and $\bar{l}$, respectively, and $F_l \otimes F_{\bar{l}} = \{C \otimes_l C' \mid C \in F_l, C' \in F_{\bar{l}}$, and $C \otimes_l C'$ is not a tautology$\}$). $\square$

4. In fact, based on the same reasoning, one could also eliminate literal a from $(a \vee b \vee c)$.

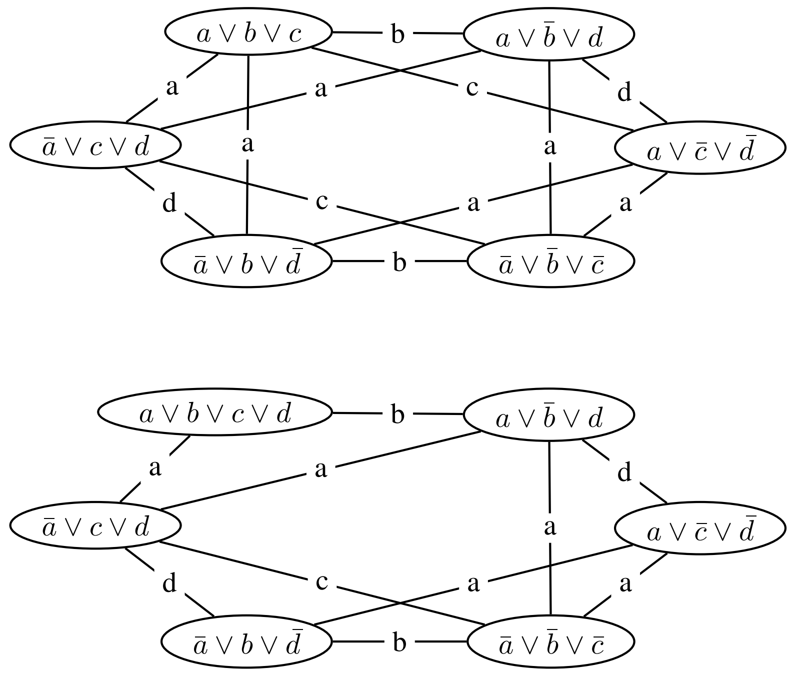


Figure 2: Two resolution graphs of F_{CLA}: both graphs have for each clause in F_{CLA} one vertex and vertices are connected with an edge if the have exactly one pair of complementary literals (hence the resolvent in non-tautological). The edges are labeled with the variable of the complementary literals. The top figure shows F_{CLA} before adding covered literal d to $(a \vee b \vee c)$ and the bottom figure shows F_{CLA} after the addition. Notice that in the top figure there is an edge for each literal. Literals in the bottom figure that have no edge associated with them, such as c and d in $(a \vee b \vee c \vee d)$ are blocking literals.

For a given clause C in a CNF formula F, we denote by (*covered literal addition*) $\text{CLA}(F, C)$ the clause resulting from repeating the following until fixpoint:

If there is a literal $l \in C$ such that $\text{RI}(F, C, l) \setminus C \neq \emptyset$, let $C := C \cup \text{RI}(F, C, l)$.

Lemma 7. *Replacing a clause $C \in F$ by $\text{CLA}(F, C)$ preserves satisfiability.*

Proof. The clause $\text{CLA}(F, C)$ is obtained by iteratively applying Lemma 6 on clause C. □

Lemma 8. *Assume two clauses C, D with $l \in C \subseteq D$ and two sets of clauses F, G with $F \subseteq G$. Further assume that D is not blocked w.r.t. F and hence C is not blocked w.r.t. G. Then $\text{RC}(G, C, l) \supseteq \text{RC}(F, D, l) \neq \emptyset$ and hence $\text{RI}(G, C, l) \subseteq \text{RI}(F, D, l)$.*

Proof. Monotonicity of RC w.r.t. its first argument and anti-monotonicity w.r.t. its second argument follows directly from its definition. For RI, note that intersection is anti-monotonic for non-empty sets of sets. □

Theorem 3. *Given a CNF formula F and a clause $C \in F$, $\mathrm{CLA}(F, C)$ is blocked or uniquely defined.*

Proof. Assume C is not blocked w.r.t. F and contains two literals l_1, l_2, which cover the literals $L'_i = \mathrm{RI}(F, C, l_i)$ respectively. Consider the clauses $C_1 = C \cup L'_1$ and $C_2 = C \cup L'_2$. Now assume that both of C_1, C_2 are not blocked w.r.t. F. Then all clauses $D \in \mathrm{RC}(F, C_1, l_2) \subseteq \mathrm{RC}(F, C, l_2)$ contain all literals in L'_2. Since C_1 is not blocked and thus $\mathrm{RC}(F, C_1, l_2)$ is not empty, we obtain $L'_2 \subseteq \mathrm{RI}(F, C_1, l_2)$. The case where the indices are exchanged (i.e., $L'_1 \subseteq \mathrm{RI}(F, C_2, l_1)$) is symmetric. Thus as long clauses do not become blocked, covered literals can be added independently. The case that both of C_1, C_2 are blocked is trivial.

What remains (by symmetry) is the case that C_2 is blocked but C_1 is not. Again, we get $L'_2 \subseteq \mathrm{RI}(F, C_1, l_2)$. For $C'_1 = C_1 \cup \mathrm{RI}(F, C_1, l_2)$ we have $C'_1 = C \cup L'_1 \cup \mathrm{RI}(F, C_1, l_2) \supseteq L'_1 \cup (C \cup L'_2) \supseteq C'_2$ which is also blocked. This generalizes to the following observation: For any non-deterministic choice of adding covered literals to C, the literal l_2 remains covering. Further, if in this process the clause did not become blocked, it will eventually become blocked if the covered literals of l_2 are added. □

With the needed preliminaries in place, we are ready to introduce covered clause elimination procedures. The first one is the plain variant, simply called covered clause elimination.

Definition 15. *Given a CNF formula F, a clause $C \in F$ is* covered *if $\mathrm{CLA}(F, C)$ is blocked w.r.t. F.*

Example 12. *Back to Example 11. Recall that $\mathrm{RI}(F_{\mathrm{CLA}}, (a \vee b \vee c), b) = \{a, d\}$. Also, $\mathrm{RI}(F_{\mathrm{CLA}}, (a \vee b \vee c), c) = \{a, \bar{d}\}$. Therefore, depending on the order of addition, $\mathrm{CLA}(F_{\mathrm{CLA}}, (a \vee b \vee c))$ is either $(a \vee b \vee c \vee d)$ when starting with covering literal b, or $(a \vee b \vee c \vee \bar{d})$ when starting with covering literal c. In both cases $\mathrm{CLA}(F_{\mathrm{CLA}}, (a \vee b \vee c))$ is blocked. After replacing $(a \vee b \vee c)$ by $(a \vee b \vee c \vee d)$, the truth assignment τ with $\tau(a) = \tau(b) = \tau(c) = \mathbf{f}$ and $\tau(d) = \mathbf{t}$ satisfies the new formula, while falsifying $(a \vee b \vee c) \in F_{\mathrm{CLA}}$. In fact, F_{CLA} witnesses the fact that none of the clause elimination procedures based on covered clauses, as introduced next, preserve logical equivalence in general.*

An illustration of the above sis show in Figure 2. It shows the resolution graph of F_{CLA} before and after adding a covered literal.

Lemma 9. *Removal of an arbitrary covered clause preserves satisfiability.*

Proof. Clause C can be replaced by $\mathrm{CLA}(F, C)$ (Lemma 7), and C can be removed as $\mathrm{CLA}(F, C)$ is blocked. □

Definition 16 (Covered Clause Elimination)**.** *For a given formula F,* covered clause elimination *(*CCE*) repeats the following until fixpoint: If there is a covered clause $C \in F$, let $F := F \setminus \{C\}$. The resulting* unique *formula is denoted by* $\mathrm{CCE}(F)$*.*

Confluence of CCE follows from the following lemma.

Lemma 10. *The following holds for any CNF formula F, clause $C \in F$, and set of clauses $S \subseteq F$ such that $C \notin S$. If C is covered w.r.t. F, then C is covered w.r.t. $F \setminus S$.*

Proof. Let $\mathrm{CLA}(F, C) = C_k$, where $C_0 := C$, and $C_{i+1} := C_i \cup \mathrm{RI}(F, C_i, l_i)$ for each $i = 0..k-1$ and $l_i \in C_i$. Now define $D_0 := C$ and, for each $i = 0..k-1$, $D_{i+1} := D_i$ if D_i is blocked w.r.t. $F \setminus S$ and $D_{i+1} := D_i \cup \mathrm{RI}(F \setminus S, D_i, l_i)$ otherwise. Using Lemma 8, one can show by induction that for each i we have either (i) D_i is blocked w.r.t. $F \setminus S$, or (ii) $\mathrm{RI}(F \setminus S, D_i, l_i) \supseteq \mathrm{RI}(F, C_i, l_i)$. If (i) holds for some i, then $\mathrm{CLA}(F \setminus S, C)$ is blocked w.r.t. $F \setminus C$. If D_i is not blocked w.r.t. $F \setminus S$ for any i, then $\mathrm{CLA}(F \setminus S, C) \supseteq \mathrm{CLA}(F, C)$. □

Theorem 4. CCE *is confluent.*

Proof. Follows directly from Lemma 10: if two clauses C and D are both covered w.r.t. F, then C is covered w.r.t. $F \setminus \{D\}$. □

Lemma 11. CCE *is more powerful than* BCE.

Proof. CCE is at least as powerful as BCE follows from the fact that $C \subseteq \mathrm{CLA}(C)$: if C is blocked, so is $\mathrm{CLA}(C)$. Moreover, in F_{CLA} no clause is blocked. However, all clauses are covered. Hence BCE will not remove a single clause, while CCE removes all of them. □

In fact, in cases it is possible to add covered literals to $\mathrm{ALA}(F, \mathrm{CLA}(F, C))$: there are cases when adding asymmetric literals to a clause can increase $|\mathrm{RI}(F, C, l)|$ for a non-asymmetric literal $l \in C$.

Example 13. *Consider the CNF formula* $F = (a \vee b) \wedge (b \vee c) \wedge (\bar{a} \vee c) \wedge (\bar{a} \vee d)$. *Notice that* $\mathrm{CLA}(F, (a \vee b), a) = (a \vee b)$. *Literal* $\bar{c}$ *is asymmetric w.r.t.* $(a \vee b)$ *due to* $(b \vee c)$. *After adding* $\bar{c}$*,* a *in the extended clause* $(a \vee b \vee \bar{c})$ *covers* d*.*

This observation motivates the following definition for *asymmetric covered* clauses, based on extending clauses iteratively by both CLA and ALA until a fixpoint is reached. More formally, for a given CNF formula F, a clause $C \in F$ is asymmetric covered if the clause resulting from repeating

1. $C := \mathrm{CLA}(F, C)$.
2. $C := \mathrm{ALA}(F, C)$.

until fixpoint is blocked w.r.t. F. Based on this definition, we arrive at asymmetric covered clause elimination, ACCE.

Definition 17. Asymmetric covered clause elimination *(*ACCE*) repeats the following until fixpoint: if there is asymmetric covered clause* C *in* F*, let* $F := F \setminus \{C\}$*.*

Lemma 12. *Removal of an arbitrary asymmetric covered clause preserves satisfiability.*

Proof. Follows from the facts that (i) F is satisfiability-equivalent to $(F \setminus \{C\}) \cup \{\mathrm{CLA}(F, C)\}$; (ii) F is satisfiability-equivalent to $(F \setminus \{C\}) \cup \{\mathrm{ALA}(F, C)\}$; and (iii) BCE preserves satisfiability. □

Lemma 13. ACCE *is more powerful than* (*i*) ABCE*, and* (*ii*) CCE*.*

Proof. (i) By replacing CCE and BCE by ACCE and ABCE in the proof of Lemma 11. (ii) Consider the formula

$$\begin{aligned} F_{\mathrm{ACCE}} \quad = \quad & (a \vee b \vee c) \wedge (a \vee b \vee \bar{c}) \wedge (a \vee \bar{b} \vee c) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge \\ & (\bar{a} \vee b \vee c) \wedge (\bar{a} \vee b \vee \bar{c}) \wedge (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c}) \wedge \\ & (a \vee b \vee d) \wedge (a \vee b \vee \bar{d}) \wedge (a \vee \bar{b} \vee d) \wedge (a \vee \bar{b} \vee \bar{d}). \end{aligned}$$

CLA cannot add literals to any clause in F_{ACCE}. However, ALA can add d and $\bar{d}$ to $(a \vee b \vee c)$ using $(a \vee b \vee \bar{d})$ and $(a \vee b \vee d)$, respectively. After ALA, $(a \vee b \vee c)$ is a tautology, so ACCE can remove it. □

6.2 Quantified Covered Clause Elimination

We now introduce a lifting of covered clause elimination for QSAT. Thereby, covered clauses are clauses which are blocked when they are enriched with literals contained in any resolvent with pivot element l, the covering literal. As with QBCE, the prefix ordering has to be taken into account.

Definition 18. *Let* QRC *denote the set of* resolution candidates *with*

$$\mathrm{QRC}(G, C, l) := \{C' \mid C' \in F, \bar{l} \in C', \nexists l' : \{l', \bar{l'}\} \subseteq C \otimes_l C'\},$$

where G is a QBF with matrix F, $C \in F$, $l \in C$. Then the resolution intersection $\mathrm{QRI}(G, C, l)$ *of l and C w.r.t. G is given by*

$$\mathrm{QRI}(G, C, l) := \Big(\bigcap \{C'' \mid C' \in \mathrm{QRC}(G, C, l), C'' \subseteq C', \forall l' \in C'' : l' \leq l\}\Big) \setminus \{\bar{l}\}.$$

A literal l is called covering literal *if* $\mathrm{QRI}(G, C, l) \neq \emptyset$*, i.e., l covers the literals in* $\mathrm{QRI}(G, C, l)$.

Lemma 14. *The replacement of a clause C in a QBF G by $C \cup \mathrm{QRI}(G, C, l)$ preserves unsatisfiability.*

Proof. We will show that for each QBF $G = \Pi.(F \cup \{C\})$ it holds that if G is unsatisfiable, so is $G' = \Pi.(F \cup \mathrm{QRI}(G, C, l))$. Assume that G is unsatisfiable, but G' is not. Then there has to be at least one assignment τ such that $\tau(C) = \mathbf{f}$ and $\tau(\mathrm{QRI}(G, C, l)) = \mathbf{t}$. In consequence, there is at least one $l' \in \mathrm{QRI}(G, C, l)$ with $\tau(l') = \mathbf{t}$. Due to the construction of QRI, it holds that $\forall C' \in F$ with $\bar{l} \in C'$, $\tau(C') = \mathbf{t}$. This means that at latest, after assigning all variables v where $v < l$ with value $\tau(v)$, l is pure. This means that F is satisfiable by an assignment τ' with $\tau'(k) = \tau(k)$ for $k \neq l$ and $\tau'(l) = \neg(\tau(l))$, which is in contradiction with the assumption that G is unsatisfiable. □

In the following, $\mathrm{QRI}(G, C)$ denotes the clause C extended with all quantified covered literals, i.e., for all $l \in \mathrm{QRI}(G, C)$ it holds that $\mathrm{QRI}(G, C, l) \subseteq \mathrm{QRI}(G, C)$.

Lemma 15 (Quantified Covered Literal Addition)**.** *The replacement of a clause C in a QBF G by* $\mathrm{QRI}(G, C)$ *preserves unsatisfiability.*

Proof. Iterative application of Lemma 14. □

Definition 19 (Quantified Covered Clause)**.** *A clause C in a QBF G is* covered *if* $\mathrm{QRI}(G, C)$ *is blocked w.r.t. G.*

Theorem 5. *The removal of a covered clause preserves unsatisfiability.*

Proof. According to Lemma 15, each clause may be replaced by the clause $\mathrm{QRI}(G, C)$. If this clause is blocked, it may be removed according to Theorem 1. If it is a tautology, it can be removed due to standard rewriting rules. □

Example 14. *In the QBF $\forall a, b, c\, \exists x, y.((x \vee \bar{a}) \wedge (\bar{x} \vee y \vee b) \wedge (\bar{x} \vee y \vee c) \wedge (\bar{y} \vee a))$ the literal x of the clause $(x \vee \bar{a})$ covers the literal y. Therefore, we can replace this clause by $(x \vee \bar{a} \vee y)$ which is blocked due to the blocking literal y. Consequently, the clause $(x \vee \bar{a} \vee y)$ can be eliminated.*

The restriction that the literals in the resolution candidates are of smaller level than the according pivot is necessary for the correctness of quantified covered clause elimination, as shown in the following example.

Example 15. *Consider the unsatisfiable QBF $\exists x \forall a \exists y.((x \vee y) \wedge (\bar{x} \vee a) \wedge (\bar{y} \vee \bar{a}))$. If we do not give any restriction on the selection of the resolution candidates, we would obtain the clause $(x \vee y \vee a)$ which is blocked with blocking literal y. If we remove this clause, the QBF becomes satisfiable.*

Lemma 16. *Covered clause elimination for QSAT is confluent.*

Proof. Assume we have to add literal l to a clause C in order to make C a covered clause which is then removed by blocked clause elimination. Assume that $l \in \mathrm{QRI}(C, l')$ for a literal $l' \in C$. We have to show that if clause C' with $l \in C'$ is removed due to QCCE, then either l may be still added to C or C is blocked and hence may be removed. If C' was the only clause containing l, then there is no other clause C'' with $\bar{l}' \in C''$. Then l' is pure and C may be removed. Otherwise, l is in the intersection of the resolvents with pivot literal l' and hence l may be added to C.

□

7. Reconstructing Solutions

Since the elimination procedures based on blocked clauses and covered clauses do not preserve logical equivalence, a truth assignment τ satisfying, for example, $\mathrm{BCE}(F)$ may not satisfy F. In this section we shown how solutions to the original CNF formulas can be reconstructed based on solutions to the CNF formulas resulting from applying variations of blocked clause and covered clause elimination.

7.1 Procedures Based on Blocked Clauses

Järvisalo and Biere (2010) showed how, given any CNF formula F and truth assignment τ that satisfies $\mathrm{BCE}(F)$, one can construct a satisfying assignment for F: Add the clauses $C \in F \setminus \mathrm{BCE}(F)$ back in the opposite order of their elimination. In case C is satisfied by τ, do nothing. Otherwise, assuming that $l \in C$ is blocking C, flip the truth value of l in τ to **t**. After all clauses have been added, the modified τ satisfies F.

We now show that this procedure can be used to reconstruct solutions for formulas simplified using ABCE.

Lemma 17. *Given a clause $C \in F$, if $\mathrm{ALA}(F, C)$ is blocked and not a tautology, then there is a literal $l \in C$ blocking it.*

Proof. By construction, for each literal $l \in \mathrm{ALA}(F, C) \setminus C$, here is a clause $C' \in F$ that contains $\bar{l}$ and $C' \setminus \{\bar{l}\} \subseteq \mathrm{ALA}(F, C)$. Therefore, because $\mathrm{ALA}(F, C)$ is not a tautology, $C' \otimes_l \mathrm{ALA}(F, C) = \mathrm{ALA}(F, C) \setminus \{l\}$ is not a tautology either. Hence l is not blocking $\mathrm{ALA}(F, C)$. □

Lemma 18. *Given a CNF formula F and a truth assignment τ satisfying F, if $C \notin F$ is falsified by τ, then $\mathrm{ALA}(F, C)$ is falsified by τ.*

Proof. From Lemma 1 follows that $F \cup \{\mathrm{ALA}(F, C)\}$ is logically equivalent to $F \cup \{C\}$. Therefore, $\mathrm{ALA}(F, C)$ is satisfied by τ if and only if τ satisfies C. □

Lemma 19. *Given a CNF formula F and a truth assignment τ satisfying F, if $C \notin F$ is falsified by τ and $\mathrm{ALA}(F, C)$ is blocked w.r.t. F with blocking literal $l \in C$, then τ satisfies at least two literals in each clause $C' \in F$ with $\bar{l} \in C'$.*

Proof. First, such $C' \in F$ contain a literal $\bar{l}$ which is satisfied by τ. Second, because l is blocking, each clause C' must contain one more literal $l' \neq \bar{l}$ such that $\bar{l}' \in \mathrm{ALA}(F, C)$. Since all literals in $\mathrm{ALA}(F, C)$ are falsified by τ, l' must be satisfied by τ. □

Combining these three lemmas, we can reconstruct a solution for F if we have a satisfying assignment τ for any $\mathrm{ABCE}(F)$. The clauses $C \in F \setminus \mathrm{ABCE}(F)$ are added back in reverse order of elimination to ensure that $\mathrm{ALA}(F, C)$ is blocked. If C is satisfied by F do nothing. Otherwise, we know that there is a literal $l \in C$ blocking $\mathrm{ALA}(F, C)$; recall Lemma 17. Furthermore, all literals in $\mathrm{ALA}(F, C)$ are falsified; recall Lemma 18. However, any $C' \in F$ containing $\bar{l}$ has two satisfied literals; recall Lemma 19. Therefore, by flipping the truth assignment for l to **t**, C becomes satisfied, while no such C' becomes falsified.

Theorem 6. *The following holds for an arbitrary CNF formula F and truth assignment τ satisfying F. For any clause $C \notin F$ for which C, $\mathrm{ALA}(F, C)$ is blocked w.r.t. F with blocking literal l, either (i) τ satisfies $F \cup \{C\}$, or (ii) τ', which is a copy of τ except for $\tau'(l) =$ **t**, satisfies $F \cup \{C\}$.*

The reconstruction proof provides several useful elements that can be used to implement ABCE more efficiently. First, since only original literals $l \in C$ can be blocking $\mathrm{ALA}(F, C)$, we can avoid a blocking literal check for all literals $l \in \mathrm{ALA}(F, C) \setminus C$. Second, it is enough to save each removed *original* clause C. None of the additional literals in the *extended* clause $\mathrm{ALA}(F, C)$ not occurring in C have to be flipped.

We implemented reconstruction as follows. When a clause C is eliminated by a procedure based on blocking literals (BCE and ABCE), C together with a blocking literal $l \in C$ is pushed on a reconstruction stack S: i.e., $S := S, \langle l{:}C \rangle$. During reconstruction, we examine the eliminated clauses in reverse order. If the clause on the top of the stack is falsified, the truth value of the blocking literal is flipped. Figure 3 shows the pseudo-code of the algorithm.

7.2 Procedures Based on Covered Clauses

When covered clauses are eliminated, reconstruction of solutions becomes more tricky. This is due to the following. As shown in the previous section, given any τ satisfying a formula F, and a clause C which is falsified by τ, by Lemma 18 $\mathrm{ALA}(F, C)$ is also falsified by τ. However, the analogous claim for $\mathrm{CLA}(F, C)$ is not true in general.

```
reconstruction (CNF formula F, truth assignment τ, elimination sequence S)
  while S is not empty
  |   let ⟨l:C⟩ := S.pop()
  |   if C is falsified by τ then
  |   |   flip the truth value of l in τ to t
  |   F := F ∪ {C}
  return τ
```

Figure 3: Pseudo-code for reconstructing a solution with F a reduced formula, τ a satisfying assignment for F and S a set of eliminated clauses ordered by last eliminated.

Proposition 14. *There is a CNF formula F and a satisfying truth assignment τ for F such that the following holds. There is a clause $C \in F$ that is falsified by τ, while $\mathrm{CLA}(F, C)$ is satisfied by τ.*

Proof. Let $F = F_{\mathrm{CLA}} \setminus \{(a \vee b \vee c)\}$ and assume that the truth assignment τ assigns $\tau(a) = \tau(b) = \tau(c) = \tau(d) = \mathbf{f}$. Let $C = (a \vee b \vee c)$. Now, τ satisfies F, but falsifies C. Since $c \in C$ covers $\bar{d}$, τ satisfies $\mathrm{CLA}(F, C) = (a \vee b \vee c \vee \bar{d})$. □

Additionally, for any formula F and clause $C \in F$, if $\mathrm{ALA}(F, C)$ is blocked w.r.t. F, then there is a literal $l \in C$ blocking it by definition. However, in case $\mathrm{CLA}(F, C)$ is blocked, there might be a literal in $\mathrm{CLA}(F, C) \setminus C$ blocking it.

Proposition 15. *There is a CNF formula F for which the following holds. There exist a clause $C \in F$ such that C is not blocked w.r.t. F but $\mathrm{CLA}(F, C)$ is blocked (due to a blocking literal $l \in \mathrm{CLA}(F, C) \setminus C$).*

Proof. Recall F_{CLA} from Section 6. In F_{CLA}, $(a \vee b \vee c)$ is not blocked. However, the extended clause $\mathrm{CLA}(F_{\mathrm{CLA}}, C) = (a \vee b \vee c \vee \bar{d})$ is blocked with blocking literal $\bar{d}$. □

Due to the properties stated as Propositions 14 and 15, given a truth assignment τ satisfying a formula F and a covered clause $C \in F$, one may be required to flip the truth values of multiple variables in τ in order to construct a satisfying assignments for $F \cup \{C\}$ based on τ. We will next show how this can be achieved.

Theorem 7. *Given a CNF formula F and a truth assignment τ satisfying F. Let clause $C \notin F$ be falsified by τ, while there is a literal $l \in C$ such that τ satisfies $C \cup \mathrm{RI}(F, C, l)$, then τ', being a copy of τ with l assigned to* **t***, satisfies F and C.*

Proof. We need to show that after flipping l to **t**, all clauses in $F_{\bar{l}}$ are still satisfied. There are two cases. First, consider a clause $C' \in F_{\bar{l}}$ such that $C \otimes_l C'$ is a tautology. Then there must be a $x \in C$ such that $\bar{x} \in C'$. Since x is falsified by τ, because C is falsified by τ, $\bar{x}$ is satisfied by τ and so is C'. The second case is $C'' \in F_{\bar{l}}$ with $C \otimes_l C''$ not being a tautology. By definition, $\mathrm{RI}(F, C, l) \subset C''$. Since C is falsified by τ while $C \cup \mathrm{RI}(F, C, l)$ is satisfied by τ, $\mathrm{RI}(F, C, l)$ is satisfied by τ and so is C''. □

Given a CNF formula F, $C \notin F$ with $\mathrm{CLA}(F, C)$ being blocked w.r.t. F, and a truth assignment τ satisfying F. We can use the observation in Theorem 7 to compute, given a truth assignment τ' that satisfies $F \cup \{C\}$. In case τ satisfies C it is trivial ($\tau' = \tau$). Otherwise, there is a sequence $C_0, C_1, \ldots, C_c$ such that $C_0 := C$, $C_c := \mathrm{CLA}(F, C)$, and $C_{i+1} := C_i \cup \mathrm{RI}(F, C_i, l_i)$ with $l_i \in C_i$. Now, let the reconstruction stack S contain the sequence $\langle l_0{:}C_0\rangle, \langle l_1{:}C_1\rangle, \ldots, \langle l_c{:}C_c\rangle$. Applying $\tau' = \mathsf{reconstruction}(F, \tau, S)$ using the algorithm in Figure 3 produces τ' that satisfies F and C.

7.3 Solution Reconstruction for QSAT

We briefly review approaches to obtaining satisfiability models from preprocessed formulas in the context of QSAT. These approaches can be classified into two categories depending on whether full or partial satisfiability models are generated.

First, for some QSAT applications, it is sufficient to extract a partial model of a formula out of a preprocessed one. Often only the values of the leftmost existential block of variables in a QBF are of interest. In this case, the partial model represents a single assignment to these variables, that is a Skolem function of zero arity. To generate partial models in practice, the preprocessor is restricted to apply only those preprocessing rules which do not affect any variables from the leftmost quantifier block. These variables are declared as *don't touch* variables (Seidl & Könighofer, 2014). This approach originates from incremental bounded model checking based on SAT (Kupferschmid, Lewis, Schubert, & Becker, 2011) and QSAT (Marin, Miller, & Becker, 2012).

As an alternative to don't touch variables, the preprocessor can be equipped with partial tracing capabilities (Heyman, Smith, Mahajan, Leong, & Abu-Haimed, 2014). Thereby, the application of the preprocessing rules is not restricted. Instead, information necessary to reconstruct a partial model in terms of an assignment to the leftmost existential variables is collected during preprocessing.

The second category comprises methods to extract full satisfiability models. Reconstruction steps for common preprocessing rules except the expansion of universal variables have been presented by Janota, Grigore, and Marques-Silva (2013). The effect of universal expansion cannot be expressed solely by Q-resolution, in contrast to equivalence literal substitution, for example, and hence causes complications. The QRAT proof system (Heule, Seidl, & Biere, 2014a; Heule et al., 2014b) is the first framework to allow the extraction of full satisfiability models from formulas preprocessed using all currently implemented preprocessing techniques, including universal expansion.

8. Experimental Evaluation

Complementing the more theoretical analysis on the relationships and properties of the considered clause elimination procedures, we now present results on an empirical evaluation of the effect of applying clause elimination on the runtimes of state-of-the-art SAT and QSAT solvers. As benchmarks, we used standard competition benchmark sets from the most recent SAT Competition (SAT Competitions Organizing Committee, 2014) and QSAT solver evaluation (Jordan & Seidl, 2014), focusing on real-world application instances. As an overview of the results, it turned out that the clause elimination procedures developed in this work, applied within preprocessing, have a clear positive effect on the performance of various state-of-the-art QSAT solvers. On pure CNF SAT formulas, the effects—while still positive or non-negative on the whole—are more modest, both when applying clause elimination as preprocessing, as well as within inprocessing. This difference can partly be explained as follows: SAT benchmarks are typically large and relatively easy considering their size, while—due to the more succinct representation form enabled with the use of quantifiers—QSAT benchmarks are relatively small in terms of how hard they are to solve in practice. Although the presented clause elimination techniques require polynomial time—while the solving procedures are exponential—they can be at times expensive in practice, especially on very large CNF formulas with millions of variables and clauses.[5]

All experiments were performed on a cluster of 2.8-GHz Intel Core 2 Quad machines each equipped with 8-GB memory and running Ubuntu 9.04.

8.1 Effectiveness of Clause Elimination in the Context of SAT

We evaluated the effectiveness of clause elimination procedures in the context of state-of-the-art SAT solvers, more specifically LINGELING version *aqw* (Biere, 2013), which won the application track in the SAT competition 2013. LINGELING heavily relies on the concept of inprocessing (Järvisalo et al., 2012b). As already discussed in the introduction, inprocessing is based on the idea of interleaving preprocessing, including clause elimination procedures, with search. The inprocessing paradigm enables the use of facts learned during search, such as learned unit clauses, in subsequent inprocessing phases, and the then simplified clauses again during search. Beside this synergistic effect, inprocessing allows preprocessing algorithms to be pre-empted by search and then resumed in the next inprocessing phase, to avoid getting stuck in a too-costly preprocessing stages.

There is a difference in cost, in terms of running time, of specific inprocessing algorithms, as well as the issue that certain simplification steps can actually be achieved with different inprocessing techniques. As a consequence, it is extremely difficult to evaluate the effect of individual inprocessing techniques in isolation precisely. As an alternative, we investigated how the performance of the competition version of LINGELING is affected when disabling (i) all clause elimination procedures, (ii) all pre- and inprocessing techniques other than clause elimination procedures, and (iii) *all* pre- and inprocessing techniques. As the benchmark set we used the same instances as in the application track of the SAT 2013 competition, with the time limit of 5000 seconds per benchmark, running on hardware with almost identical speed as in the competition. In essence, the results show how LINGELING would have performed in the competition without clause elimination in comparison to using other or none of the pre- and inprocessing techniques applied within the solver.

We note that we conducted the same experiment also using the application track instances from the SAT Competition 2014. While modest improvements (as shown in the following) on the 2013 instances can be observed, on the 2014 instances clause elimination procedures did not noticeably improve (nor degrade) the performance of LINGELING.[6] Hence here we present more details on the results for the 2013 instances. In general, it appears that clause elimination procedures provide only modest improvements for SAT solvers, but much more substantial improvement for QSAT solvers—as we will demonstrate in the following.

5. In fact, in terms of worst-case complexity, it has recently been shown that, conditional to the so-called strong exponential time hypothesis (SETH) (Impagliazzo, Paturi, & Zane, 2001; Calabro, Impagliazzo, & Paturi, 2009) being true, checking whether a given CNF formula contains a clause having AT cannot be done in sub-quadratic time, even when restricting to Horn-3-CNF formulas (Järvisalo & Korhonen, 2014).
6. We suspect that the differences in the 2013 and 2014 benchmarks are due to the benchmark selection procedure applied by the competition organizers, which to an extent balance out performance differences between a set of top-performing solvers from the previous year. This benchmark selection procedure used in the main SAT competitions during 2012–2014 is described by Balint, Belov, Järvisalo, and Sinz (2015).

configuration	#sat	#unsat	#total	avg*	total**
pre- & inprocessing disabled	108	83	191	1254	784440
only clause elimination enabled	112	86	198	1209	749324
base line without clause elimination	111	112	223	**1111**	632852
LINGELING version *aqw* (base line)	**119**	**113**	**232**	1124	**600691**

$*$ over solved formulas

$**$ over all formulas

Table 2: LINGELING configurations on application instances from SAT Competition 2013.

Among other inprocessing algorithms this competition version of LINGELING implements the following clause elimination procedures. A separate BCE inprocessor is scheduled during inprocessing if bounded variable elimination (VE) (Eén & Biere, 2005) was run-to-completion at least once. Note that BCE can be implemented much faster than variable elimination. However, since the latter has often a more pronounced effect, we run VE until completion first, before applying BCE. Further, as proposed by Han and Somenzi (2007), BCE can partially be performed on-the-fly during VE. In the configuration without clause elimination we disable both variants of BCE.

The special case of on-the-fly *subsumption* of on-the-fly *strengthening* during conflict clause learning (Han & Somenzi, 2009), can also be considered as a clause elimination procedure, but is performed during search. Thus we keep it enabled in each of the configurations of LINGELING used in this experiment. The same applies to subsumption elimination (SE) during VE (Eén & Biere, 2005) and subsumed clauses found during lazy hyper binary resolution (Heule et al., 2013b). Another separate inprocessor performs transitive reduction of the binary implication graph. In practice, transitive reduction is actually quite fast, but unfortunately in our experience does not give much benefit in terms of solving times.

LINGELING *aqw* also contains an implementation of ACCE, which is rather costly and usually never runs until completion. There is also another simple and partial variant of ATE, called basic ATE (BATE). It is computationally inexpensive to detect some asymmetric tautologies (AT) during (two-sided) literal probing (Le Berre, 2001), which is used as basic probing technique in various inprocessing algorithms.

In the default configuration of LINGELING *aqw*, all the discussed clause elimination procedures not only "wait" until VE was completed at least once, as BCE does, but also require BCE to be completed at least once (except for BCE obviously). Concretely, the configuration of LINGELING with all the clause elimination procedures disabled is called with the following command line options: `--no-bate --no-block --no-cce --no-transred`. By specifying `--plain` we further compare a configuration where pre- and inprocessing is disabled with a configuration which only uses clause elimination procedures during pre- and inprocessing, using `--plain --bate --block --cce=3 --transred`, which first disables all preprocessing and then selectively enables all clause elimination procedure, combined with `--batewait=0 --blockwait=0 --ccewait=1`, which makes sure that CCE is only started after BCE has run until completion (as in the default configuration) and BCE is not delayed.

Results from the experiment are shown in Figure 4 and Table 2. The net result of this evaluation is that the default configuration of LINGELING, as entered to the competition, with all clause elimination procedures enabled, solves 9 more instances; base line version solves 232 benchmarks but only 223 without clause elimination. Out of the nine instances, eight are satisfiable, which seems to

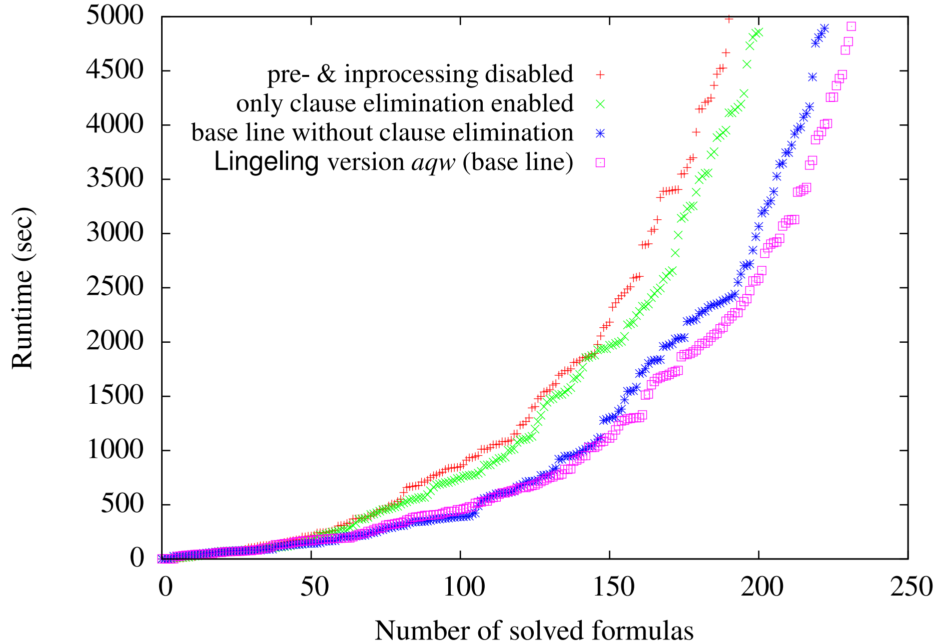


Figure 4: LINGELING version *aqw* without clause elimination procedures solves 9 instances less on the application track benchmark set of the SAT 2013 Competition with a time limit of 5000 seconds.

suggest that additionally applying clause elimination is beneficial especially on satisfiable instances. The results suggest that there are in cases benefits to using clause elimination procedures, but the improvements are not on the same scale as enabling or disabling VE (Eén & Biere, 2005). Here we note that, as explained in detail in (Järvisalo et al., 2012a), while VE and BCE are to some extent orthogonal in terms of the simplifications achieved on CNF formulas, they can perform various types of similar simplifications on their own. These include, for examples, various circuit-level optimizations, such as cone-of-influence and monotone input gate reductions, as well as CNF level techniques such as pure literal eliminations.

8.2 Effectiveness of Clause Elimination in the Context of QSAT

In the following, we empirically investigate the impact of PCNF-level clause elimination procedures when applied as preprocessing in QSAT. To this end, we implemented those techniques in the QSAT preprocessor BLOQQER (version 35) (Seidl & Biere, 2015). The core of BLOQQER is based on the concept of "resolve and expand" realized in the QSAT solver QUANTOR (Biere, 2005). Basically, QUANTOR is a complete solver using variable elimination to remove existential variables from the innermost quantifier block and universal expansion to remove variables from the innermost universal quantifier block. Depending on the benchmark family, this approach proved to be either extremely efficient such that formulas hard to other solvers could be solved within a few seconds or to be extremely memory-consuming. To overcome this limitation, we developed the preprocessor

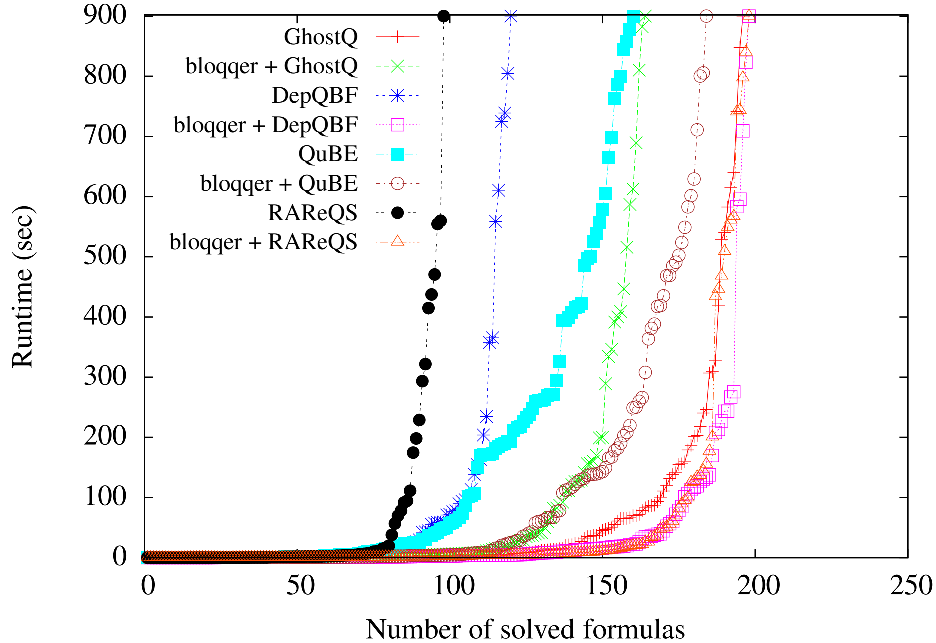


Figure 5: Runtimes on the QBFLib Track Benchmarks of QBF Gallery 2014.

BLOQQER that applies the "resolve and expand" approach in a bounded manner. In BLOQQER, the preprocessed formula is rewritten in a way that a complete solver can then benefit from the careful application of variable elimination and universal expansion which are repeatedly applied until the formula either does not change any more or until some limits are reached. (Note that a formula might already be solved in the preprocessing phase.) We integrated the clause elimination techniques in BLOQQER such that they are applied during each cycle of variable elimination and universal expansion.

For evaluating our implementation, we considered the benchmarks of the QBFLib track and of the Application track of the QBF Gallery 2014 (Jordan & Seidl, 2014). The formulas included in the QBFLib track are a selection of the QBFLib, the QBF community platform. This set contains 345 formulas from various benchmark families. In the competitive evaluation of the QBF Gallery 2014 only a subset of 276 formulas not directly solved by preprocessors was used. The benchmark set of the Application track consists of 735 formulas from recently presented encodings to QSAT. None of these formulas is directly solved by BLOQQER.

The time and memory limits were set to 900 seconds and 7 GB, respectively. Time spent on preprocessing is included in the time limit for experiments that involve QSAT solvers. If the preprocessor does not terminate after 900 seconds, then preprocessing is aborted and the formula is considered to be unsolved. We consider four participants of the QBF Gallery, which are publicly available: the CDCL-based solver DEPQBF (Lonsing & Biere, 2010; Egly, Lonsing, & Widl, 2013); the CEGAR-based solver RAREQS (Janota, Klieber, Marques-Silva, & Clarke, 2012); the GHOSTQ solver (Klieber et al., 2010; Klieber, 2014) implementing a CEGAR-based approach in combination with so-called ghost variables, allowing for duality-aware reasoning on the CNF level; and the solver QUBE that includes the preprocessor SQUEEZEBF (Giunchiglia et al., 2010). This

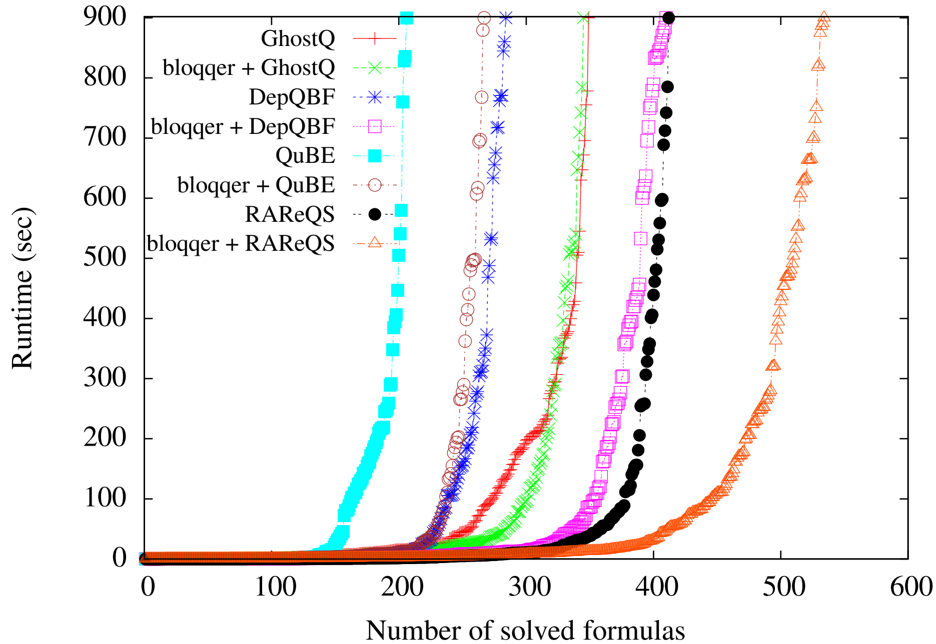


Figure 6: Runtimes on the Application Track Benchmarks of QBF Gallery 2014.

configuration	**#sat**	**#unsat**	**#total**	**avg***	**total****	vars	cls.
no preprocessing	45	74	119	56	210K	32925	77710
no VE/Expansion	70 (2)	72 (8)	142 (10)	66	192K	32928	36863
no QBCE	76 (27)	91 (34)	167 (61)	47	168K	33306	31776
no QCCE	98 (32)	**98** (**36**)	196 (68)	42	142K	33342	28012
no asymmetric CE	98 (28)	94 (33)	192 (61)	53	148K	33310	31642
full preprocessing	**99** (**33**)	**98** (**37**)	**197** (**70**)	**38**	**141K**	33381	27858

$*$ average runtime over solved formulas

$**$ total runtime over all formulas

Table 3: Different BLOQQER configurations with solver DEPQBF on QBFLib benchmarks.

preprocessor implements several techniques also included in BLOQQER as well as a special kind of equivalence substitution, but no clause elimination procedures like BCE and CCE.

Each of these four solvers was run in its standard configuration with and without BLOQQER. The results are shown in Figure 5 and Figure 6. For all solvers except GHOSTQ preprocessing was beneficial. While in the Application track preprocessing hardly had any effect on the runtime of GHOSTQ, in the QBFLib track the performance of GHOSTQ was decreased by preprocessing. GHOSTQ relies on structural patterns in the CNF and the application of the preprocessor seems to destroy these patterns.

Since BLOQQER implements many preprocessing techniques, the just-described experiment does not directly indicate the power of PCNF-level clause elimination procedures. In order to evaluate the impact of the various techniques, we run different configurations of BLOQQER in com-

configuration	**#sat**	**#unsat**	**#total**	**avg***	**total****	vars	cls.
no preprocessing	155	128	283	62	424K	11262	227312
no VE/Expansion	174	**219**	393	**36**	**322K**	11283	173487
no QBCE	189	202	391	60	333K	11340	186860
no QCCE	**196**	213	**409**	69	**322K**	11356	173454
no asymmetric CE	175	213	388	41	328K	11342	174992
full preprocessing	192	217	**409**	70	**322K**	11358	173402

∗ average runtime over solved formulas

∗∗ total runtime over all formulas

Table 4: Different BLOQQER configurations with solver DEPQBF on Application benchmarks.

bination with our solver DEPQBF. The results are summarized in Table 3 and Table 4. The tables give the number of solved satisfiable and unsatisfiable formulas and well as average runtimes for the solved formulas and the total runtime for the complete benchmark set. The last two columns show the average number of variables and clauses of the original formulas for the *no preprocessing* configuration. For the other configurations, the average number of variables and clauses of the preprocessed formulas is given. Table 3 additionally contains information about the number of formulas directly solved by BLOQQER (the number of solved formulas in brackets). We ran the following configurations: (i) no preprocessing, i.e., only the solver DEPQBF, (ii) variable elimination and expansion turned off but the clause elimination techniques turned on, (iii) blocked clause elimination turned off, (iv) covered clause elimination turned off, (v) asymmetric clause elimination techniques turned off, and (vi) all preprocessing techniques turned on.

For both benchmark sets, we observe that the application of BLOQQER is very beneficial for DEPQBF. The best performance is achieved when applying all preprocessing techniques. By turning off variable elimination and universal expansion, we see that using only the clause elimination techniques is already beneficial for the solver. The results for different BLOQQER configurations show that the clause elimination techniques considerably improve the runtimes and the number of solved formulas, especially in the case of the QBFLib track benchmarks. In average, the application of BLOQQER increases the number of variables. This is mainly due to universal expansion. However, also for the configuration where expansion is disabled, we observe a modest increase of the number of variables. This is because BLOQQER splits large clauses into smaller ones what turned out to be beneficial for DEPQBF. Especially for the QBFLib track benchmarks, the application of BLOQQER drastically decreases the number of clauses (more than 50 percent for most configurations).

9. Conclusions

Preprocessing and inprocessing (generally, formula simplification) techniques have proven important in speeding up state-of-the-art SAT and QSAT solving. Understanding the effects of and relationships between various simplification procedures is important for gaining a better understanding of the procedures. In this article, we focused on a specific type of preprocessing techniques, clause elimination procedures that remove clauses from CNF and PCNF formulas based on different polynomial-time checkable redundancy properties. We introduced novel clause elimination procedures for both CNF and PCNF formulas as asymmetric variants of the known techniques of tautology, subsumption, and blocked clause elimination procedures, and additionally developed a novel family (including the plain and asymmetric variants) of so-called covered clause elimination procedures. We analyzed all of the variants from various perspectives—relative effectiveness, BCP-preserving, confluence, logical equivalence—highlighting intricate differences between the procedures. This also resulted in a relative power hierarchy, reflecting the relative strengths of the procedures in removing clauses. In terms of relative power, the asymmetric variant of covered clause elimination dominates the plain procedures, and the novel covered clause elimination procedures are the most powerful ones among the considered procedures. Complementing the more theoretical analysis, we presented results of an empirical evaluation on the practical effectiveness of the procedures in speeding-up the overall solving runtime of state-of-the-art SAT and QSAT solvers on real-world benchmark instances. The results show that, while the effects on the SAT-level are modest, applying the clause elimination procedures is clearly beneficial in the context of QSAT solving.

Many of the SAT-level clause elimination procedures have already been integrated as inprocessing techniques in a state-of-the-art SAT solver. An important aspect of future work would be to integrate these procedures as inprocessing techniques into a QSAT solver. The motivation for doing so is similar to that of the current state-of-the-art inprocessing SAT solvers: to speed up the satisfiability search further via interleaving applications of preprocessing techniques with the core search routine. For example, clauses may become blocked during the solving process and then removed. An additional question to investigate is whether it is possible to loosen the blocking criterion by taking variable dependencies into account. It would also be interesting to perform a thorough in-depth study of clause elimination in the context of other generalizations of and formalisms related to SAT, such as maximum satisfiability (MaxSAT) and the extraction of minimally unsatisfiable subsets (MUSes) of CNF formulas. While there is some recent work looking into possibilities of applying SAT-based preprocessing in the contexts of MUS and MaxSAT (Belov, Järvisalo, & Marques-Silva, 2013a; Belov, Morgado, & Marques-Silva, 2013b; Berg, Saikko, & Järvisalo, 2015)–including the use of BCE—we believe such directions have not yet been fully explored.

Acknowledgments

The authors would like to thank Donald Knuth for his comments that helped to improve the article. The authors gratefully acknowledge financial support from DARPA contract number N66001-10-2-4087 (MH); Academy of Finland under grants 251170 COIN Centre of Excellence in Computational Inference Research, 276412, and 284591 (MJ); Austrian Science Foundation (FWF) NFN Grants S11408-N23 RiSE (AB) and S11409-N23 RiSE (FL); and Vienna Science and Technology Fund (WWTF) under grant ICT10-018 (MS).

References

Bacchus, F. (2002). Enhancing Davis Putnam with extended binary clause reasoning. In Dechter, R., & Sutton, R. S. (Eds.), *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI 2002)*, pp. 613–619. AAAI Press.

Balabanov, V., & Jiang, J.-H. R. (2011). Resolution proofs and Skolem functions in QBF evaluation and applications. In Gopalakrishnan, G., & Qadeer, S. (Eds.), *Proceedings of the 23rd*

International Conference on Computer Aided Verification (CAV 2011), Vol. 6806 of *Lecture Notes in Computer Science*, pp. 149–164. Springer.

Balint, A., Belov, A., Järvisalo, M., & Sinz, C. (2015). Overview and analysis of the SAT Challenge 2012 solver competition. *Artificial Intelligence*, *223*, 120–155.

Belov, A., Järvisalo, M., & Marques-Silva, J. (2013a). Formula preprocessing in MUS extraction. In Piterman, N., & Smolka, S. A. (Eds.), *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013)*, Vol. 7795 of *Lecture Notes in Computer Science*, pp. 108–123. Springer.

Belov, A., Morgado, A., & Marques-Silva, J. (2013b). SAT-based preprocessing for MaxSAT. In McMillan, K. L., Middeldorp, A., & Voronkov, A. (Eds.), *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-19)*, Vol. 8312 of *Lecture Notes in Computer Science*, pp. 96–111. Springer.

Benedetti, M. (2005a). Extracting certificates from quantified boolean formulas. In Kaelbling, L. P., & Saffiotti, A. (Eds.), *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pp. 47–53. Professional Book Center.

Benedetti, M. (2005b). sKizzo: A Suite to Evaluate and Certify QBFs. In Nieuwenhuis, R. (Ed.), *Proceedings of the 20th International Conference on Automated Deduction (CADE-20)*, Vol. 3632 of *Lecture Notes in Computer Science*, pp. 369–376. Springer.

Benedetti, M., & Mangassarian, H. (2008). QBF-based formal verification: Experience and perspectives. *Journal of Satisfiability, Boolean Modeling and Computation*, *5*(1-4), 133–191.

Berg, J., Saikko, P., & Järvisalo, M. (2015). Improving the effectiveness of SAT-based preprocessing for MaxSAT. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*. AAAI Press.

Biere, A. (2005). Resolve and expand. In Hoos, H. H., & Mitchell, D. G. (Eds.), *Revised Selected Papers of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, Vol. 3542 of *Lecture Notes in Computer Science*, pp. 59–70. Springer.

Biere, A. (2013). Lingeling, Plingeling and Treengeling entering the SAT Competition 2013. In Balint, A., Belov, A., Heule, M., & Järvisalo, M. (Eds.), *Proceedings of SAT Competition 2013*, Vol. B-2013-1 of *Department of Computer Science Series of Publications B*, pp. 51–52. University of Helsinki.

Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.). (2009). *Handbook of Satisfiability*, Vol. 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Biere, A., Lonsing, F., & Seidl, M. (2011). Blocked clause elimination for QBF. In Bjørner, N., & Sofronie-Stokkermans, V. (Eds.), *Proceedings of the 23rd International Conference on Automated Deduction (CADE 2011)*, Vol. 6803 of *Lecture Notes in Computer Science*, pp. 101–115. Springer.

Brafman, R. I. (2004). A simplifier for propositional formulas with many binary clauses. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, *34*(1), 52–59.

Bubeck, U., & Kleine Büning, H. (2007). Bounded universal expansion for preprocessing QBF. In Marques-Silva, J., & Sakallah, K. A. (Eds.), *Proceedings of the 10th International Conference* *on Theory and Applications of Satisfiability Testing (SAT 2007)*, Vol. 4501 of *Lecture Notes in Computer Science*, pp. 244–257. Springer.

Cadoli, M., Giovanardi, A., & Schaerf, M. (1998). An algorithm to evaluate quantified boolean formulae. In Mostow, J., & Rich, C. (Eds.), *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 1998)*, pp. 262–267. AAAI Press / The MIT Press.

Calabro, C., Impagliazzo, R., & Paturi, R. (2009). The complexity of satisfiability of small depth circuits. In Chen, J., & Fomin, F. V. (Eds.), *Revised Selected Paper of the 4th International Workshop on Parameterized and Exact Computation (IWPEC 2009)*, Vol. 5917 of *Lecture Notes in Computer Science*, pp. 75–85. Springer.

Claessen, K., Eén, N., Sheeran, M., & Sörensson, N. (2008). SAT-solving in practice. In *Proceedings of the 9th International Workshop on Discrete Event Systems (WODES 2008)*, pp. 61–67. IEEE.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In Harrison, M. A., Banerji, R. B., & Ullman, J. D. (Eds.), *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971)*, pp. 151–158. ACM.

Eén, N., & Biere, A. (2005). Effective preprocessing in SAT through variable and clause elimination. In Bacchus, F., & Walsh, T. (Eds.), *Proceedings of 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, Vol. 3569 of *Lecture Notes in Computer Science*, pp. 61–75. Springer.

Egly, U., Lonsing, F., & Widl, M. (2013). Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In McMillan, K., Middeldorp, A., & Voronkov, A. (Eds.), *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2013)*, Vol. 8312 of *Lecture Notes in Computer Science*, pp. 291–308. Springer.

Fourdrinoy, O., Grégoire, É., Mazure, B., & Saïs, L. (2007a). Eliminating redundant clauses in SAT instances. In Hentenryck, P. V., & Wolsey, L. A. (Eds.), *Proceedings of the 4th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2007)*, Vol. 4510 of *Lecture Notes in Computer Science*, pp. 71–83. Springer.

Fourdrinoy, O., Grégoire, É., Mazure, B., & Saïs, L. (2007b). Reducing hard SAT instances to polynomial ones. In *Proceedings of the 8th IEEE International Conference on Information Reuse and Integration (IRI 2007)*, pp. 18–23. IEEE.

Freeman, J. (1995). *Improvements to propositional satisfiability search algorithms*. Ph.D. thesis, University of Pennsylvania.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Gershman, R., & Strichman, O. (2005). Cost-effective hyper-resolution for preprocessing CNF formulas. In Bacchus, F., & Walsh, T. (Eds.), *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, Vol. 3569 of *Lecture Notes in Computer Science*, pp. 423–429. Springer.

Giunchiglia, E., Marin, P., & Narizzano, M. (2010). sQueezeBF: An effective preprocessor for QBFs based on equivalence reasoning. In Strichman, O., & Szeider, S. (Eds.), *Proceedings of* *the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT 2010)*, Vol. 6175 of *Lecture Notes in Computer Science*, pp. 85–98. Springer.

Goultiaeva, A., & Bacchus, F. (2013). Recovering and utilizing partial duality in QBF. In Järvisalo, M., & Gelder, A. V. (Eds.), *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT 2013)*, Vol. 7962 of *Lecture Notes in Computer Science*, pp. 83–99. Springer.

Goultiaeva, A., Seidl, M., & Biere, A. (2013). Bridging the gap between dual propagation and CNF-based QBF solving. In Macii, E. (Ed.), *Proceedings of Design, Automation and Test in Europe Conference & Exhibition (DATE 2013)*, pp. 811–814. EDA Consortium / ACM DL.

Goultiaeva, A., Van Gelder, A., & Bacchus, F. (2011). A uniform approach for generating proofs and strategies for both true and false QBF formulas. In Walsh, T. (Ed.), *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pp. 546–553. IJCAI/AAAI Press.

Han, H., & Somenzi, F. (2007). Alembic: An efficient algorithm for CNF preprocessing. In *Proceedings of the 44th Design Automation Conference (DAC 2007)*, pp. 582–587. IEEE.

Han, H., & Somenzi, F. (2009). On-the-fly clause improvement. In Kullmann, O. (Ed.), *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT 2009)*, Vol. 5584 of *Lecture Notes in Computer Science*, pp. 209–222. Springer.

Heule, M., Järvisalo, M., & Biere, A. (2010). Clause elimination procedures for CNF formulas. In Fermüller, C., & Voronkov, A. (Eds.), *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-17)*, Vol. 6397 of *Lecture Notes in Computer Science*, pp. 357–371. Springer.

Heule, M., Järvisalo, M., & Biere, A. (2013a). Covered clause elimination. In Fermüller, C., & Voronkov, A. (Eds.), *Short Paper Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-17)*, Vol. 13 of *EasyChair Proceedings in Computing*, pp. 41–46.

Heule, M., Järvisalo, M., & Biere, A. (2013b). Revisiting hyper binary resolution. In Gomes, C. P., & Sellmann, M. (Eds.), *Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2013)*, Vol. 7874 of *Lecture Notes in Computer Science*, pp. 77–93. Springer.

Heule, M., Seidl, M., & Biere, A. (2014a). A Unified Proof System for QBF Preprocessing. In Demri, S., Kapur, D., & Weidenbach, C. (Eds.), *Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR 2014)*, Vol. 8562 of *Lecture Notes in Computer Science*, pp. 91–106. Springer.

Heule, M., Seidl, M., & Biere, A. (2014b). Efficient extraction of Skolem functions from QRAT proofs. In Claessen, K., & Kuncak, V. (Eds.), *Proceedings of the 14th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2014)*, pp. 107–114. IEEE.

Heyman, T., Smith, D., Mahajan, Y., Leong, L., & Abu-Haimed, H. (2014). Dominant controllability check using QBF-solver and netlist optimizer. In Sinz, C., & Egly, U. (Eds.), *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT 2014)*, Vol. 8561 of *Lecture Notes in Computer Science*, pp. 227–242. Springer.

Impagliazzo, R., Paturi, R., & Zane, F. (2001). Which problems have strongly exponential complexity?. *Journal of Computer and System Sciences*, *63*(4), 512–530.

Janota, M., Grigore, R., & Marques-Silva, J. (2013). On QBF proofs and preprocessing. In McMillan, K. L., Middeldorp, A., & Voronkov, A. (Eds.), *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-19)*, Vol. 8312 of *Lecture Notes in Computer Science*, pp. 473–489. Springer.

Janota, M., Klieber, W., Marques-Silva, J., & Clarke, E. (2012). Solving QBF with counterexample guided refinement. In Cimatti, A., & Sebastiani, R. (Eds.), *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT 2012)*, Vol. 7317 of *Lecture Notes in Computer Science*, pp. 114–128. Springer.

Järvisalo, M., & Biere, A. (2010). Reconstructing solutions after blocked clause elimination. In Strichman, O., & Szeider, S. (Eds.), *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT 2010)*, Vol. 6175 of *Lecture Notes in Computer Science*, pp. 340–345. Springer.

Järvisalo, M., Biere, A., & Heule, M. (2010). Blocked clause elimination. In Esparza, J., & Majumdar, R. (Eds.), *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010)*, Vol. 6015 of *Lecture Notes in Computer Science*, pp. 129–144. Springer.

Järvisalo, M., Biere, A., & Heule, M. (2012a). Simulating circuit-level simplifications on CNF. *Journal of Automated Reasoning*, *49*(4), 583–619.

Järvisalo, M., Heule, M., & Biere, A. (2012b). Inprocessing rules. In Gramlich, B., Miller, D., & Sattler, U. (Eds.), *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR 2012)*, Vol. 7364 of *Lecture Notes in Computer Science*, pp. 355–370. Springer.

Järvisalo, M., & Korhonen, J. H. (2014). Conditional lower bounds for failed literals and related techniques. In Sinz, C., & Egly, U. (Eds.), *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT 2014)*, Vol. 8561 of *Lecture Notes in Computer Science*, pp. 75–84. Springer.

Järvisalo, M., Le Berre, D., Roussel, O., & Simon, L. (2012). The international SAT solver competitions. *AI Magazine*, *33*(1), 89–92.

Jin, H., & Somenzi, F. (2005). An incremental algorithm to check satisfiability for bounded model checking. *Electronic Notes in Theoretical Computer Science*, *119*(2), 51–65.

Jordan, C., & Seidl, M. (2014). QBF Gallery 2014. `http://qbf.satisfiability.org/gallery/`.

Kleine Büning, H., & Bubeck, U. (2009). Theory of quantified Boolean formulas. In Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.), *Handbook of Satisfiability*, Vol. 185 of *Frontiers in Artificial Intelligence and Applications*, pp. 735–760. IOS Press.

Kleine Büning, H., Karpinski, M., & Flögel, A. (1995). Resolution for Quantified Boolean Formulas. *Information and Computation*, *117*(1), 12–18.

Klieber, W. (2014). *Formal Verification Using Quantified Boolean Formulas (QBF)*. Ph.D. thesis, Carnegie Mellon University, available at http://reports-archive.adm.cs.cmu.edu/anon/2014/CMU-CS-14-117.pdf.

Klieber, W., Sapra, S., Gao, S., & Clarke, E. M. (2010). A non-prenex, non-clausal QBF solver with game-state learning. In Strichman, O., & Szeider, S. (Eds.), *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT 2010)*, Vol. 6175 of *Lecture Notes in Computer Science*, pp. 128–142. Springer.

Kullmann, O. (1999). On a generalization of extended resolution. *Discrete Applied Mathematics*, *96–97*, 149–176.

Kupferschmid, S., Lewis, M. D. T., Schubert, T., & Becker, B. (2011). Incremental preprocessing methods for use in BMC. *Formal Methods in System Design*, *39*(2), 185–204.

Le Berre, D. (2001). Exploiting the real power of unit propagation lookahead. *Electronic Notes in Discrete Mathematics*, *9*, 59–80.

Liberatore, P. (2005). Redundancy in logic I: CNF propositional formulae. *Artificial Intelligence*, *163*(2), 203–232.

Lonsing, F., & Biere, A. (2010). DepQBF: A dependency-aware QBF solver. *Journal of Satisfiability, Boolean Modeling and Computation*, *7*(2-3), 71–76.

Lynce, I., & Marques-Silva, J. (2001). The interaction between simplification and search in propositional satisfiability. In *CP'01 Workshop on Modeling and Problem Formulation*.

Mangassarian, H., Le, B., Goultiaeva, A., Veneris, A. G., & Bacchus, F. (2010). Leveraging dominators for preprocessing QBF. In *Design, Automation and Test in Europe (DATE 2010)*, pp. 1695–1700. IEEE.

Manthey, N., Heule, M., & Biere, A. (2013). Automated reencoding of boolean formulas. In Biere, A., Nahir, A., & Vos, T. E. J. (Eds.), *Revised Selected Papers of the 8th International Haifa Verification Conference (HVC 2012)*, Vol. 7857 of *Lecture Notes in Computer Science*, pp. 102–117. Springer.

Marin, P., Miller, C., & Becker, B. (2012). Incremental QBF preprocessing for partial design verification - (poster presentation). In Cimatti, A., & Sebastiani, R. (Eds.), *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT 2012)*, Vol. 7317 of *Lecture Notes in Computer Science*, pp. 473–474. Springer.

Marques-Silva, J. (2008). Practical applications of Boolean satisfiability. In *Proceedings of the 9th International Workshop on Discrete Event Systems (WODES 2008)*, pp. 74–80. IEEE.

Niemetz, A., Preiner, M., Lonsing, F., Seidl, M., & Biere, A. (2012). Resolution-based certificate extraction for QBF - (tool presentation). In Cimatti, A., & Sebastiani, R. (Eds.), *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT 2012)*, Vol. 7317 of *Lecture Notes in Computer Science*, pp. 430–435. Springer.

Ostrowski, R., Grégoire, É., Mazure, B., & Saïs, L. (2002). Recovering and exploiting structural knowledge from CNF formulas. In Hentenryck, P. V. (Ed.), *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002)*, Vol. 2470 of *Lecture Notes in Computer Science*, pp. 185–199. Springer.

Piette, C., Hamadi, Y., & Saïs, L. (2008). Vivifying propositional clausal formulae. In Ghallab, M., Spyropoulos, C. D., Fakotakis, N., & Avouris, N. M. (Eds.), *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, Vol. 178 of *Frontiers in Artificial Intelligence and Applications*, pp. 525–529. IOS Press.

Pigorsch, F., & Scholl, C. (2010). An AIG-based QBF-solver using SAT for preprocessing. In Sapatnekar, S. S. (Ed.), *Proceedings of the 47th Design Automation Conference (DAC 2010)*, pp. 170–175. ACM.

Pulina, L., & Tacchella, A. (2009). A structural approach to reasoning with quantified boolean formulas. In Boutilier, C. (Ed.), *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pp. 596–602.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM*, *12*(1), 23–41.

Sabharwal, A., Ansótegui, C., Gomes, C. P., Hart, J. W., & Selman, B. (2006). QBF Modeling: Exploiting Player Symmetry for Simplicity and Efficiency. In Biere, A., & Gomes, C. P. (Eds.), *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT 2006)*, Vol. 4121 of *Lecture Notes in Computer Science*, pp. 382–395. Springer.

Samer, M. (2008). Variable dependencies of quantified CSPs. In Cervesato, I., Veith, H., & Voronkov, A. (Eds.), *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2008)*, Vol. 5330 of *Lecture Notes in Computer Science*, pp. 512–527. Springer.

Samulowitz, H., Davies, J., & Bacchus, F. (2006). Preprocessing QBF. In Benhamou, F. (Ed.), *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP 2006)*, Vol. 4204 of *Lecture Notes in Computer Science*, pp. 514–529. Springer.

SAT Competitions Organizing Committee (2014). The international SAT Competitions web page. `http://satcompetition.org/`.

Schaefer, T. J. (1978). On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, *16*(2), 185–225.

Seidl, M., & Biere, A. (2015). Bloqqer. `http://fmv.jku.at/bloqqer`.

Seidl, M., & Könighofer, R. (2014). Partial witnesses from preprocessed quantified Boolean formulas. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE 2014)*, pp. 1–6. IEEE.

Subbarayan, S., & Pradhan, D. K. (2005). NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In Hoos, H. H., & Mitchell, D. G. (Eds.), *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, Vol. 3542 of *Lecture Notes in Computer Science*, pp. 276–291. Springer.

Van Gelder, A. (2005). Toward leaner binary-clause reasoning in a satisfiability solver. *Annals of Mathematics and Artificial Intelligence*, *43*(1), 239–253.

Van Gelder, A., Wood, S. B., & Lonsing, F. (2012). Extended failed-literal preprocessing for quantified boolean formulas. In Cimatti, A., & Sebastiani, R. (Eds.), *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT 2012)*, Vol. 7317 of *Lecture Notes in Computer Science*, pp. 86–99. Springer.

Zhang, L. (2006). Solving QBF by combining conjunctive and disjunctive normal forms. In Gil, Y., & Mooney, R. J. (Eds.), *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*, pp. 143–150. AAAI Press.