Blocked Clause Elimination

Matti Järvisalo[1], Armin Biere[2], and Marijn Heule[3]

[1] Department of Computer Science, University of Helsinki, Finland
[2] Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria
[3] Algorithmics Group, Delft University of Technology, The Netherlands

Abstract. Boolean satisfiability (SAT) and its extensions are becoming a core technology for the analysis of systems. The SAT-based approach divides into three steps: encoding, preprocessing, and search. It is often argued that by encoding arbitrary Boolean formulas in conjunctive normal form (CNF), structural properties of the original problem are not reflected in the CNF. This should result in the fact that CNF-level preprocessing and SAT solver techniques have an inherent disadvantagecompared to related techniques applicable on the level of more structural SAT instance representations such as Boolean circuits. In this work we study the effect of a CNF-level simplification technique called blocked clause elimination (BCE). We show that BCE is surprisingly effective both in theory and in practice on CNFs resulting from a standard CNF encoding for circuits: without explicit knowledge of the underlying circuit structure, it achieves the same level of simplification as a combination of circuit-level simplifications and previously suggested polarity-based CNF encodings. Experimentally, we show that by applying BCE in preprocessing, further formula reduction and faster solving can be achieved, giving promise for applying BCE to speed up solvers.

1 Introduction

Boolean satisfiability (SAT) solvers and their extensions, especially satisfiability modulo theories (SMT) solvers, are becoming a core technology for the analysis of systems, ranging from hardware to software. SAT solvers are in the heart of SMT solvers, and in some cases such as the theory of bit-vectors, state-of-the-art SMT solvers are based on bit-blasting and use pure SAT solvers for actual solving. This gives motivation for developing even more efficient SAT techniques.

SAT-based approaches typically consist of three steps: encoding, preprocessing, and search. These steps, however, are tightly intertwined. For example, efficient propagation techniques applied in search (unit propagation as a simple example) are also applicable in preprocessing for simplifying the input formula. Furthermore, preprocessing and simplifications can be applied both on the conjunctive normal form (CNF) level—which still is the most typical input form for state-of-the-art SAT solvers–and on higher-level, more structural formula representations, such as Boolean circuits. Indeed, SAT encodings often go though a circuit-level formula representation, which is then translated into CNF. This highlights the importance of good CNF representations of Boolean circuits.

It is often argued that by encoding arbitrary Boolean formulas in CNF, structural properties of the original problem are not reflected in the resulting CNF. This should

J. Esparza and R. Majumdar (Eds.): TACAS 2010, LNCS 6015, pp. 129–144, 2010.

result in the fact that CNF-level preprocessing and SAT solver techniques have an inherent disadvantage compared to related techniques that can be applied on the level of more structural SAT instance representations such as Boolean circuits. Motivated by this, various simplification techniques and intricate CNF encoders for circuit-level SAT instance descriptions have been proposed [1,2,3,4,5]. On the other hand, based on the highly efficient CNF-level clause learning SAT solvers and CNF simplification techniques such as [6,7,8,9,10,11], there is also strong support for the claim that CNF is sufficient as an input format for SAT solvers.

In this work we study the effect of a CNF-level simplification technique called blocked clause elimination (BCE), based on the concept of blocked clauses [12]. We show that BCE is surprisingly effective both in theory and in practice on CNFs resulting from the standard "Tseitin" CNF encoding [13] for circuits: without explicit knowledge of the underlying circuit structure, BCE achieves the same level of simplification as a combination of circuit-level simplifications, such as *cone of influence*, *non-shared input elimination*, and *monotone input reduction*, and previously suggested polarity-based CNF encodings, especially the Plaisted-Greenbaum encoding [14]. This implies that, without losing simplification achieved by such specialized circuit-level techniques, one can resort to applying BCE after the straightforward Tseitin CNF encoding, and hence implementing these circuit-level techniques is somewhat redundant. Moreover, since other related circuit level optimizations for *sequential* problems—in particular, the *bounded cone of influence reduction* [15] and using functional instead of relational representations of circuits [16]—can be mapped to cone of influence, these can also be achieved by BCE purely on the CNF-level. Additionally, as regards CNF-level simplification techniques, BCE achieves the simplification resulting from, e.g., *pure literal elimination*. In addition to the more theoretical analysis in this paper, we present an experimental evaluation of the effectiveness of BCE combined with SatElite-style variable eliminating CNF preprocessing [10], comparing our implementation with the standard Tseitin and Plaisted-Greenbaum encodings and the more recent NiceDAG [4,5] and Minicirc [3] CNF encoders.

The rest of this paper is organized as follows. After background on Boolean circuits and CNF encodings of circuits (Sect. 2) and on resolution-based CNF preprocessing (Sect. 3), we introduce blocked clause elimination (Sect. 4). Then the effectiveness of BCE is analyzed w.r.t. known circuit-level simplification techniques and CNF encodings (Sect. 5) and resolution-based preprocessing (Sect. 6). Finally, our implementation of BCE is briefly described (Sect. 7) and experimental results are reported on the practical effectiveness of BCE (Sect. 8).

2 Boolean Circuits and CNF SAT

This section reviews the needed background related to Boolean circuits and CNF-level satisfiability, and well-known CNF encodings of circuits.

Given a Boolean variable x, there are two *literals*, the positive literal, denoted by x, and the negative literal, denoted by $\bar{x}$, the *negation of* x. As usual, we identify $\bar{\bar{x}}$ with x. A *clause* is a disjunction ($\vee$, or) of distinct literals and a CNF formula is a conjunction ($\wedge$, and) of clauses. When convenient, we view a clause as a finite set of literals and a

CNF formula as a finite set of clauses; e.g. the formula $(a \vee \bar{b}) \wedge (\bar{c})$ can be written as $\big\{\{a, \bar{b}\}, \{\bar{c}\}\big\}$. A clause is a *tautology* if it contains both x and $\bar{x}$ for some variable x.

2.1 Boolean Circuits

A Boolean circuit over a finite set G of *gates* is a set $\mathcal{C}$ of equations of form $g := f(g_1, \ldots, g_n)$, where $g, g_1, \ldots, g_n \in G$ and $f : \{\mathbf{t}, \mathbf{f}\}^n \to \{\mathbf{t}, \mathbf{f}\}$ is a Boolean function, with the additional requirements that (i) each $g \in G$ appears at most once as the left hand side in the equations in $\mathcal{C}$, and (ii) the underlying directed graph

$$\langle G, E(\mathcal{C}) = \{\langle g', g\rangle \in G \times G \mid g := f(\ldots, g', \ldots) \in \mathcal{C}\}\rangle$$

is acyclic. If $\langle g', g\rangle \in E(\mathcal{C})$, then g' is a *child* of g and g is a *parent* of g'. If $g := f(g_1, \ldots, g_n)$ is in $\mathcal{C}$, then g is an f-gate (or of type f), otherwise it is an *input gate*. A gate with no parents is an *output gate*. The fanout (fanin, resp.) of a gate is the number of parents (children, resp.) the gate has.

A (partial) assignment for $\mathcal{C}$ is a (partial) function $\tau : G \to \{\mathbf{t}, \mathbf{f}\}$. An assignment τ is *consistent* with $\mathcal{C}$ if $\tau(g) = f(\tau(g_1), \ldots, \tau(g_n))$ for each $g := f(g_1, \ldots, g_n)$ in $\mathcal{C}$.

A *constrained Boolean circuit* $\mathcal{C}^\tau$ is a pair $\langle \mathcal{C}, \tau\rangle$, where $\mathcal{C}$ is a Boolean circuit and τ is a partial assignment for $\mathcal{C}$. With respect to a $\mathcal{C}^\tau$, each $\langle g, v\rangle \in \tau$ is a *constraint*, and g is *constrained* to v if $\langle g, v\rangle \in \tau$.

An assignment τ' *satisfies* $\mathcal{C}^\tau$ if (i) it is consistent with $\mathcal{C}$, and (ii) it respects the constraints in τ, meaning that for each gate $g \in G$, if $\tau(g)$ is defined, then $\tau'(g) = \tau(g)$. If some assignment satisfies $\mathcal{C}^\tau$, then $\mathcal{C}^\tau$ is *satisfiable* and otherwise *unsatisfiable*.

The following Boolean functions are some which often occur as gate types.

- $\textsc{not}(v)$ is $\mathbf{t}$ if and only if v is $\mathbf{f}$.
- $\textsc{or}(v_1, \ldots, v_n)$ is $\mathbf{t}$ if and only if at least one of $v_1, \ldots, v_n$ is $\mathbf{t}$.
- $\textsc{and}(v_1, \ldots, v_n)$ is $\mathbf{t}$ if and only if all $v_1, \ldots, v_n$ are $\mathbf{t}$.
- $\textsc{xor}(v_1, \ldots, v_n)$ is $\mathbf{t}$ if and only if an odd number of v_i's are $\mathbf{t}$.
- $\textsc{ite}(v_1, v_2, v_3)$ is $\mathbf{t}$ if and only if (i) v_1 and v_2 are $\mathbf{t}$, or (ii) v_1 is $\mathbf{f}$ and v_3 is $\mathbf{t}$.

As typical, we inline gate definitions of type $g := \textsc{not}(g')$. In other words, each occurrence of g as $\hat{g} := f(\ldots, g, \ldots)$ is expected to be rewritten as $\hat{g} := f(\ldots, \textsc{not}(g'), \ldots)$.

2.2 Well-Known CNF Encodings

The standard satisfiability-preserving "Tseitin" encoding [13] of a constrained Boolean circuit $\mathcal{C}^\tau$ into a CNF formula $\mathrm{TST}(\mathcal{C}^\tau)$ works by introducing a Boolean variable for each gate in $\mathcal{C}^\tau$, and representing for each gate $g := f(g_1, \ldots g_n)$ in $\mathcal{C}^\tau$ the equivalence $g \Leftrightarrow f(g_1, \ldots g_n)$ with clauses. Additionally, the constraints in τ are represented as unit clauses: if $\tau(g) = \mathbf{t}$ ($\tau(g) = \mathbf{f}$, resp.), introduce the clause (g) ($(\bar{g})$, resp.). A well-known fact is that unit propagation[1] on $\mathrm{TST}(\mathcal{C}^\tau)$ behaves equivalently to standard Boolean constraint propagation on the original circuit $\mathcal{C}^\tau$ (see, e.g., [17] for details).

[1] Given a CNF formula F, while there is a unit clause $\{l\}$ in F, unit propagation removes from F (i) all clauses in F in which l occurs, and (ii) the literal $\bar{l}$ from each clause in F.

A well-known variant of the Tseitin encoding is the Plaisted-Greenbaum encoding [14] which is based on *gate polarities*. Given a constrained Boolean circuit $\mathcal{C}^\tau$, a *polarity function* $\mathsf{pol}^\tau_{\mathcal{C}} : G \to 2^{\{\mathbf{t},\mathbf{f}\}}$ assigns polarities to each gate in the circuit. Here **t** and **f** stand for the *positive* and *negative* polarities, respectively. Any polarity function must satisfy the following requirements.

- If $\langle g, v\rangle \in \tau$, then $v \in \mathsf{pol}^\tau_{\mathcal{C}}(g)$.
- If $g := f(g_1, \ldots, g_n)$, then:
  - If $f = \text{NOT}$, then $v \in \mathsf{pol}^\tau_{\mathcal{C}}(g)$ implies $\bar{v} \in \mathsf{pol}^\tau_{\mathcal{C}}(g_1)$.
  - If $f \in \{\text{AND}, \text{OR}\}$, then $v \in \mathsf{pol}^\tau_{\mathcal{C}}(g)$ implies $v \in \mathsf{pol}^\tau_{\mathcal{C}}(g_i)$ for each i.
  - If $f = \text{XOR}$, then $\mathsf{pol}^\tau_{\mathcal{C}}(g) \neq \emptyset$ implies $\mathsf{pol}^\tau_{\mathcal{C}}(g_i) = \{\mathbf{t}, \mathbf{f}\}$.
  - If $f = \text{ITE}$, then $v \in \mathsf{pol}^\tau_{\mathcal{C}}(g)$ implies $\mathsf{pol}^\tau_{\mathcal{C}}(g_1) = \{\mathbf{t}, \mathbf{f}\}$ and $v \in \mathsf{pol}^\tau_{\mathcal{C}}(g_i)$ for $i = 2, 3$.

The Plaisted-Greenbaum encoding [14] uses the polarity function $\mathsf{minpol}^\tau_{\mathcal{C}}$ that assigns for each gate the subset-minimal polarities from $2^{\{\mathbf{t},\mathbf{f}\}}$ respecting the requirements above. In other words, for each gate g,

$$\mathsf{minpol}^\tau_{\mathcal{C}}(g) := \{v \mid \tau(g) = v \ \text{ or } \ v \in \mathsf{minpol}^\tau_{\mathcal{C}}(g') \text{ for some parent } g' \text{ of } g\}.$$

The Tseitin encoding, on the other hand, can be seen as using the subset-maximal polarity assigning polarity function $\mathsf{maxpol}^\tau_{\mathcal{C}}(g) := \{\mathbf{t}, \mathbf{f}\}$ for each gate g. For the gate types considered in this paper, the clauses introduced based on gates polarities are listed in Table 1.

Table 1. CNF encoding for constrained Boolean circuits based on gate polarities. In the table, $\mathbf{g}_i$ is $\bar{g}'_i$ if $g_i := \text{NOT}(g'_i)$, and g_i otherwise.

gate g	$\mathbf{t} \in \mathsf{pol}^\tau_{\mathcal{C}}(g)$	$\mathbf{f} \in \mathsf{pol}^\tau_{\mathcal{C}}(g)$
$g := \text{OR}(g_1, \ldots, g_n)$	$(\bar{g} \vee \mathbf{g}_1 \vee \cdots \vee \mathbf{g}_n)$	$(g \vee \bar{\mathbf{g}}_1), \ldots, (g \vee \bar{\mathbf{g}}_n)$
$g := \text{AND}(g_1, \ldots, g_n)$	$(\bar{g} \vee \mathbf{g}_1), \ldots, (\bar{g} \vee \mathbf{g}_n)$	$(g \vee \bar{\mathbf{g}}_1 \vee \cdots \vee \bar{\mathbf{g}}_n)$
$g := \text{XOR}(g_1, g_2)$	$(\bar{g} \vee \bar{\mathbf{g}}_1 \vee \bar{\mathbf{g}}_2), (\bar{g} \vee \mathbf{g}_1 \vee \mathbf{g}_2)$	$(g \vee \bar{\mathbf{g}}_1 \vee \mathbf{g}_2), (g \vee \mathbf{g}_1 \vee \bar{\mathbf{g}}_2)$
$g := \text{ITE}(g_1, g_2, g_3)$	$(\bar{g} \vee \bar{\mathbf{g}}_1 \vee \mathbf{g}_2), (\bar{g} \vee \mathbf{g}_1 \vee \mathbf{g}_3)$	$(g \vee \bar{\mathbf{g}}_1 \vee \bar{\mathbf{g}}_2), (g \vee \mathbf{g}_1 \vee \bar{\mathbf{g}}_3)$
$\langle g, \mathbf{t}\rangle \in \tau$	(g)	
$\langle g, \mathbf{f}\rangle \in \tau$	$(\bar{g})$	

Given a constrained Boolean circuit $\mathcal{C}^\tau$, we denote the CNF resulting from the Plaisted-Greenbaum encoding of $\mathcal{C}^\tau$ by $\text{PG}(\mathcal{C}^\tau)$.

Relevant concepts additional concepts related to polarities are

- *monotone gates*: gate g is monotone if $|\mathsf{minpol}^\tau_{\mathcal{C}}(g)| = 1$; and
- *redundant gates*: gate g is redundant if $\mathsf{minpol}^\tau_{\mathcal{C}}(g) = \emptyset$.

3 Resolution and CNF-Level Simplification

The resolution rule states that, given two clauses $C_1 = \{x, a_1, \ldots, a_n\}$ and $C_2 = \{\bar{x}, b_2, \ldots, b_m\}$, the implied clause $C = \{a_1, \ldots, a_n, b_1, \ldots, b_m\}$, called the *resolvent* of C_1 and C_2, can be inferred by *resolving* on the variable x. We write $C = C_1 \otimes C_2$.

This notion can be lifted to sets of clauses: For two sets S_x and $S_{\bar{x}}$ of clauses which all contain x and $\bar{x}$, respectively, we define

$$S_x \otimes S_{\bar{x}} = \{C_1 \otimes C_2 \mid C_1 \in S_x, C_2 \in S_{\bar{x}}, \text{ and } C_1 \otimes C_2 \text{ is not a tautology}\}.$$

Following the Davis-Putnam procedure [18] (DP), a basic simplification technique, referred to as *variable elimination by clause distribution* in [10], can be defined. The elimination of a variable x in the whole CNF can be computed by pair-wise resolving each clause in S_x with every clause in $S_{\bar{x}}$. Replacing the original clauses in $S_x \cup S_{\bar{x}}$ with the set of *non-tautological* resolvents $S = S_x \otimes S_{\bar{x}}$ gives the CNF $(F \setminus (S_x \cup S_{\bar{x}})) \cup S$ which is satisfiability-equivalent to F.

Notice that DP is a complete proof procedure for CNFs, with exponential worst-case space complexity. Hence for practical applications of variable elimination by clause distribution as a simplification technique for CNFs, variable elimination needs to be bounded. Closely following the heuristics applied in the SatElite preprocessor [10] for applying variable elimination, in this paper we study as a simplification technique the bounded variant of variable elimination by clause distribution, VE, under which a variable x can be eliminated only if $|S| \leq |S_x \cup S_{\bar{x}}|$, i.e., when the resulting CNF formula $(F \setminus (S_x \cup S_{\bar{x}})) \cup S$ will not contain more clauses as the original formula F.[2]

It should be noted that the result of VE can vary significantly depending on the order in which variables are eliminated. In more detail, VE doesn't have a unique fixpoint for all CNF formulas, and the fixpoint reached in practice is dependent on variable elimination ordering heuristics. Hence VE is not *confluent*.

Proposition 1. VE *is not confluent.*

4 Blocked Clause Elimination

The main simplification technique studied in this paper is what we call *blocked clause elimination* (BCE), which removes so called *blocked clauses* [12] from CNF formulas.

Definition 1 (Blocking literal). *A literal l in a clause C of a CNF F blocks C (w.r.t. F) if for every clause $C' \in F$ with $\bar{l} \in C'$, the resolvent $(C \setminus \{l\}) \cup (C' \setminus \{\bar{l}\})$ obtained from resolving C and C' on l is a tautology.*

With respect to a fixed CNF and its clauses we have:

Definition 2 (Blocked clause). *A clause is blocked if it has a literal that blocks it.*

Example 1. Consider the formula $F_{\text{blocked}} = (a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$. Only the first clause of F_{blocked} is not blocked. Both of the literals a and $\bar{c}$ block the second clause. The literal c blocks the last clause. Notice that after removing either $(a \vee \bar{b} \vee \bar{c})$ or $(\bar{a} \vee c)$, the clause $(a \vee b)$ becomes blocked. This is actually an extreme case in which BCE can remove all clauses of a formula, resulting in a trivially satisfiable formula. □

[2] More precisely, the SatElite preprocessor [10] applies a variant of VE called *variable elimination by substitution*. The analysis on VE in this paper applies to this variant as well.

As a side-remark, notice that a literal l cannot block any clause in a CNF formula F if F contains the unit clause $\{\bar{l}\}$, and hence in this case no clause containing l can be blocked w.r.t. F.

An important fact is that BCE preserves satisfiability.

Proposition 2 ([12]). *Removal of an arbitrary blocked clause preserves satisfiability.*

Additionally, we have the following.

Proposition 3. *Given a CNF formula F, let clause $C \in F$ be blocked w.r.t. F. Any clause $C' \in F$, where $C' \neq C$, that is blocked w.r.t. F is also blocked w.r.t. $F \setminus \{C\}$.*

Therefore the result of blocked clause elimination is independent of the order in which blocked clauses are removed, and hence blocked clause elimination has a unique fixpoint for any CNF formula, i.e., BCE is confluent.

Proposition 4. BCE *is confluent.*

It should be noted that, from a proof complexity theoretic point of view, there are CNF formulas which can be made easier to prove unsatisfiable with resolution (and hence also with clause learning SAT solvers) by *adding* blocked clauses [12]. In more detail, there are CNF formulas for which minimal resolution proofs are guaranteed to be of exponential length originally, but by adding instance-specific blocked clauses to the formulas, the resulting formulas yield short resolution proofs. The effect of adding (instance-specific) blocked clauses has also been studied in different contexts [19,20,21]. However, in a more general practical sense, we will show that removal of blocked clauses by BCE yields simplified CNF formulas which are both smaller in size and easier to solve.

As a final remark before proceeding to the main contributions of this paper, we note that this is not the first time removing blocked clauses is proposed for simplifying CNFs [6]. However, in contrast to this paper, the work of [6] does not make the connection between blocked clauses and circuit-level simplifications and CNF encodings and, most importantly, [6] concentrates on extracting underlying circuit gate definitions for applying this knowledge in CNF simplification; blocked clause removal in [6] is actually *not* applied in the case any underlying gate definitions can be extracted, but rather as an auxiliary simplification over those clauses which cannot be associated with gate definitions.

5 Effectiveness of Blocked Clause Elimination

The main results of this section show the surprising effectiveness of blocked clause elimination when applied until fixpoint. We will apply the following definition of the relative effectiveness of CNF encodings and both circuit and CNF-level simplification techniques.

Definition 3. *Assume two methods T_1 and T_2 that take as input an arbitrary constrained Boolean circuit $\mathcal{C}^\tau$ and output CNF formulas $T_1(\mathcal{C}^\tau)$ and $T_2(\mathcal{C}^\tau)$, respectively, that are satisfiability-equivalent to $\mathcal{C}^\tau$. We say that T_1 is* at least as effective as *T_2 if, for any $\mathcal{C}^\tau$, $T_1(\mathcal{C}^\tau)$ contains at most as many clauses and variables as $T_2(\mathcal{C}^\tau)$ does. If T_1 is at least as effective as T_2 and vice versa, then T_1 and T_2 are* equally effective.

Notice that, considering BCE, a stricter variant of this definition, based on clause elimination, could be applied: T_1 is at least as effective as T_2 , if for every circuit $\mathcal{C}^\tau$ we have $T_1(\mathcal{C}^\tau) \subseteq T_2(\mathcal{C}^\tau)$. However, for VE this stricter definition cannot be naturally applied, since in general VE produces non-tautological resolvents which are not subsumed by the original clauses. Because of this inherent property of VE, we will for simplicity in the following use the "weaker" version, as in Definition 3. All the results presented not concerning VE also hold under the stricter version of the definition. Also notice that the "at least as effective" relation is analogously defined for two CNF-level simplification methods which, instead of Boolean circuits, take CNF formulas as input.

When considering the effectiveness of VE in this paper, we apply a non-deterministic interpretation which allows for *any* variable elimination order, i.e., we say that VE can achieve the effectiveness of another simplification technique, if there is some elimination order for which VE achieves the same effectiveness. Finally, note that in the following we always assume that Boolean circuits (CNF formulas, resp.) are closed under standard circuit-level Boolean constraint propagation (unit propagation, resp.).

An overview of the main results of this section is presented in Fig. 1. An edge from X to Y implies that X is as least as effective as Y; for further details, see the caption.

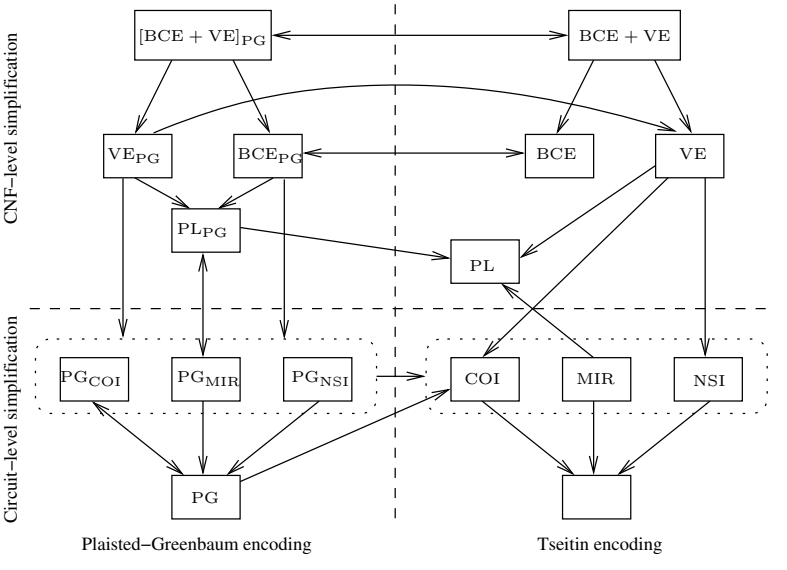

Fig. 1. Relative effectiveness of combinations of CNF encodings with both circuit and CNF-level simplification techniques. An edge from X to Y implies that X is as least as effective as Y. Notice that transitive edges are omitted. On the left side, X_{PG} means the combination of first applying the Plaisted-Greenbaum and then the CNF-level simplification technique X on the resulting CNF. Analogously, PG_X means the combination of first applying the circuit-level simplification X and then the Plaisted-Greenbaum encoding. On the right side the standard Tseitin encoding is always applied. The pointed circles around COI, MIR, and NSI on the left and right represent applying the combination of these three simplifications and then the Plaisted-Greenbaum (left) or Tseitin encoding (right). Additionally, $\mathrm{BCE} + \mathrm{VE}$ refers to all possible ways of alternating BCE and VE until fixpoint.

Notice that transitive edges are omitted: for example, BCE is at least as effective as the combination of PG, COI, NSI, and MIR.

5.1 Pure Literal Elimination by BCE

Before turning to the main results, relating BCE with circuit-level simplification techniques, we begin by first arguing that both BCE and VE actually achieve the same simplifications as the well-known *pure literal elimination*. Given a CNF formula F, a literal l occurring in F is *pure* if $\bar{l}$ does not occur in F.

Pure Literal Elimination (PL)**:** While there is a pure literal l in F, remove all clauses containing l from F.

Notice that the following two lemmas apply for all CNF formulas, and is not restricted to CNFs produced by the TST or PG encodings.

Lemma 1. BCE *is at least as effective as* PL.

Proof sketch. A pure literal blocks all clauses which contain it by definition, and hence clauses containing a pure literal are blocked. □

Lemma 2. VE *is at least as effective as* PL.

Proof sketch. Let l be a pure literal. By definition, $S_{\bar{l}}$ (the set of clauses containing $\bar{l}$) is empty. Hence $S_l \otimes S_{\bar{l}} = \emptyset$, and therefore VE removes the clauses in S_l. □

5.2 Effectiveness of BCE on Circuit-Based CNFs

In this section we will consider several circuit-level simplification techniques—*non-shared input elimination*, *monotone input elimination*, and *cone of influence reduction* [17]—and additionally the Plaisted-Greenbaum CNF encoding.

For the following, we consider an arbitrary constrained Boolean circuit $\mathcal{C}^\tau$.

Non-shared input elimination (NSI)**:** While there is a (non-constant) gate g with the definition $g := f(g_1, \ldots, g_n)$ such that each g_i is an input gate with fanout one (non-shared) in $\mathcal{C}^\tau$, remove the gate definition $g := f(g_1, \ldots, g_n)$ from $\mathcal{C}^\tau$.

Monotone input reduction (MIR)**:** While there is a monotone input gate g in $\mathcal{C}^\tau$, assign g to $\mathsf{minpol}^\tau_{\mathcal{C}}(g)$.

Cone of influence reduction (COI)**:** While there is a redundant gate g in $\mathcal{C}^\tau$, remove the gate definition $g := f(g_1, \ldots, g_n)$ from $\mathcal{C}^\tau$.

First, we observe that the Plaisted-Greenbaum encoding actually achieves the effectiveness of COI.

Lemma 3. $\mathrm{PG}(\mathcal{C}^\tau)$ *is at least as effective as* $\mathrm{PG}(\mathrm{COI}(\mathcal{C}^\tau))$.

Proof sketch. For any redundant gate g, $\mathsf{minpol}^\tau_{\mathcal{C}}(g) = \emptyset$ by definition. Hence the Plaisted-Greenbaum encoding does not introduce any clauses for such a gate. □

On the other hand, blocked clause elimination can achieve the Plaisted-Greenbaum encoding starting with the result of the Tseitin encoding.

Lemma 4. $\mathrm{BCE}(\mathrm{TST}(\mathcal{C}^\tau))$ *is at least as effective as* $\mathrm{PG}(\mathcal{C}^\tau)$.

Proof sketch. We claim that BCE removes all clauses in $\mathrm{TST}(\mathcal{C}^\tau) \setminus \mathrm{PG}(\mathcal{C}^\tau)$ from $\mathrm{TST}(\mathcal{C}^\tau)$. There are two cases to consider: redundant and monotone gates. For both cases, BCE works implicitly in a top-down manner, starting from the output gates (although BCE has no explicit knowledge of the circuit $\mathcal{C}^\tau$ underlying $\mathrm{TST}(\mathcal{C}^\tau)$.

Consider an arbitrary redundant output gate definition $g := f(g_1, \ldots, g_n)$. Since g is not constrained under τ, all clauses in $\mathrm{TST}(\mathcal{C}^\tau)$ in which g occurs are related to this definition. Now it is easy to see that the literals associated with g (recall Table 1) block each of these clauses, and hence the clauses are blocked. On the circuit level, this is equivalent to removing the definition $g := f(g_1, \ldots, g_n)$.

Now consider an arbitrary monotone output gate definition $g := f(g_1, \ldots, g_n)$ with $\mathsf{minpol}^\tau_{\mathcal{C}}(g) = \{v\}$, where $v \in \{\mathbf{t}, \mathbf{f}\}$. Then g must be constrained: $\tau(g) = v$. Hence unit propagation on g removes all clauses produced by TST for the case "if $\bar{v} \in \mathsf{pol}^\tau_{\mathcal{C}}(g)$" in Table 1 and removes the occurrences of g from the clauses produced for the case "if $v \in \mathsf{pol}^\tau_{\mathcal{C}}(g)$". To see how BCE removes in a top-down manner those clauses related to monotone gate definitions which are not produced by PG, consider the gate definition $g_i := f'(g'_1, \ldots, g'_{n'})$. Assume that unit propagation on g has no effect on the clauses produced by TST for this definition, that $\mathsf{minpol}^\tau_{\mathcal{C}}(g_i) = \{v\}$, and that BCE has removed all clauses related to the parents of g_i in $\mathrm{TST}(\mathcal{C}^\tau) \setminus \mathrm{PG}(\mathcal{C}^\tau)$. Now one can check that the literals associated with g_i block each of the clauses produced by TST for the case "if $\bar{v} \in \mathsf{pol}^\tau_{\mathcal{C}}(g_i)$". This is because all the clauses produced by TST for the definitions of g_i's parents and in which g_i occurs have been already removed by BCE (or by unit propagation). Hence all the clauses produced by TST for the case "if $\bar{v} \in \mathsf{pol}^\tau_{\mathcal{C}}(g_i)$" in Table 1 are blocked. □

Combining Lemmas 3 and 4, we have

Lemma 5. $\mathrm{BCE}(\mathrm{TST}(\mathcal{C}^\tau))$ *is at least as effective as* $\mathrm{PG}(\mathrm{COI}(\mathcal{C}^\tau))$.

Next, we consider non-shared input elimination.

Lemma 6. $\mathrm{BCE}(\mathrm{TST}(\mathcal{C}^\tau))$ *is at least as effective as* $\mathrm{PG}(\mathrm{NSI}(\mathcal{C}^\tau))$.

Proof sketch. Assume a gate definition $g := f(g_1, \ldots, g_n)$ such that each g_i is a non-shared input gate. It is easy to check from Table 1 that for each g_i, each clause produced by TST for $g := f(g_1, \ldots, g_n)$ is blocked by $\mathbf{g}_i$. The result now follows from Lemma 4 and Proposition 3 (notice that $\mathrm{PG}(\mathcal{C}^\tau)$ is always a subset of $\mathrm{TST}(\mathcal{C}^\tau)$). □

On the other hand, PL cannot achieve the effectiveness of NSI when applying PG: since PG produces the same set of clauses as TST for any gate g with $\mathsf{minpol}^\tau_{\mathcal{C}}(g) = \{\mathbf{t}, \mathbf{f}\}$, no literal occurring in these clauses can be pure.

We now turn to the monotone input reduction. Notice that MIR is a proper generalization of PL: given a CNF formula F, any pure literal in F is monotone in the straight-forward circuit representation of F where each clause $C \in F$ is represented as an output OR-gate the children of which are the literals in C. On the other hand, a monotone input gate in a circuit $\mathcal{C}^\tau$ is not necessarily a pure literal in $\mathrm{TST}(\mathcal{C}^\tau)$: TST introduces clauses which together contain both positive and negative occurrences of all gates, including monotone ones. However, it actually turns out that, when applying the Plaisted-Greenbaum encoding, PL and MIR are equally effective.

Lemma 7. $\mathrm{PL}(\mathrm{PG}(\mathcal{C}^\tau))$ *and* $\mathrm{PG}(\mathrm{MIR}(\mathcal{C}^\tau))$ *are equally effective.*

Proof sketch. Assume a gate definition $g := f(g_1, \ldots, g_n)$, where some g_i is a monotone input gate. To see that $\mathrm{PL}(\mathrm{PG}(\mathcal{C}^\tau))$ is at least as effective as $\mathrm{PG}(\mathrm{MIR}(\mathcal{C}^\tau))$, first notice that since g_i is monotone, g is monotone. Now, it is easy to check (recall Table 1) that g_i occurs only either negatively or positively in the clauses introduced by PG for $g := f(g_1, \ldots, g_n)$, and hence $\mathbf{g}_i$ is pure.

To see that $\mathrm{PG}(\mathrm{MIR}(\mathcal{C}^\tau))$ is at least as effective as $\mathrm{PL}(\mathrm{PG}(\mathcal{C}^\tau))$, notice that in order to be a pure literal in $\mathrm{PG}(\mathcal{C}^\tau)$, a gate has to be both monotone and an input. □

Using this lemma, we arrive at the fact that BCE on TST can achieve the combined effectiveness of MIR and PG.

Lemma 8. $\mathrm{BCE}(\mathrm{TST}(\mathcal{C}^\tau))$ *is at least as effective as* $\mathrm{PG}(\mathrm{MIR}(\mathcal{C}^\tau))$.

Proof sketch. Since BCE can remove all clauses in $\mathrm{TST}(\mathcal{C}^\tau) \setminus \mathrm{PG}(\mathcal{C}^\tau)$ by Lemma 4, after this BCE can remove all clauses containing some monotone input gate g_i since BCE is at least as effective as PL (Lemma 1). The result then follows by Lemma 7. □

Combining Lemmas 4, 5, 6, and 8, we finally arrive at

Theorem 1. $\mathrm{BCE}(\mathrm{TST}(\mathcal{C}^\tau))$ *is at least as effective as first applying the combination of* COI*,* MIR*, and* NSI *on* $\mathcal{C}^\tau$ *until fixpoint, and then applying* PG *on the resulting circuit.*

As an interesting side-remark, we have

Proposition 5. *The combination of* NSI*,* MIR*, and* COI *is confluent.*

Moreover, BCE is more effective than applying the combination of COI, MIR, and NSI on $\mathcal{C}^\tau$ until fixpoint, and then applying PG on the resulting circuit. To see this, consider for example a gate definition $g := \textsc{or}(g_1, \ldots, g_n)$, where g has $\mathsf{minpol}^\tau_{\mathcal{C}}(g) = \{\mathbf{t}, \mathbf{f}\}$ and only a single g_i is an input gate with fanout one (non-shared), i.e. it occurs only in the definition of g. In this case the clauses in $\mathrm{TST}(\mathcal{C}^\tau)$ in which g_i occurs are blocked.

6 Benefits of Combining BCE and VE

We will now consider aspects of applying BCE in combination with VE. As implemented in the SatElite CNF preprocessor, VE has proven to be an extremely effective preprocessing technique in practice [10].

First, we show that VE, using an optimal elimination ordering, can also achieve the effectiveness of many of the considered circuit-level simplifications.

Theorem 2. *The following claims hold.*

1. $\mathrm{VE}(\mathrm{TST}(\mathcal{C}^\tau))$ *is at least as effective as (i)* $\mathrm{TST}(\mathrm{COI}(\mathcal{C}^\tau))$*; (ii)* $\mathrm{TST}(\mathrm{NSI}(\mathcal{C}^\tau))$.
2. $\mathrm{VE}(\mathrm{PG}(\mathcal{C}^\tau))$ *is at least as effective as* $\mathrm{VE}(\mathrm{TST}(\mathcal{C}^\tau))$.
3. $\mathrm{VE}(\mathrm{PG}(\mathcal{C}^\tau))$ *is at least as effective as*
   (i) $\mathrm{PG}(\mathrm{COI}(\mathcal{C}^\tau))$*; (ii)* $\mathrm{PG}(\mathrm{NSI}(\mathcal{C}^\tau))$*; and (iii)* $\mathrm{PG}(\mathrm{MIR}(\mathcal{C}^\tau))$.

Proof sketch.

1. (i) Assume a redundant output gate definition $g := f(g_1, \ldots, g_n)$. Now $S_g \otimes S_{\bar{g}} = \emptyset$ since all resolvents are tautologies when resolving on g (recall Table 1).
   (ii) Assume a gate definition $g := f(g_1, \ldots, g_n)$ such that each g_i is an non-shared input gate. For OR (similarly for AND), $S_{g_1} \otimes S_{\bar{g}_1} = \emptyset$. After resolving on g_1 we are left with the clauses $\cup_{i=2}^{k} \{g \vee \bar{\mathbf{g}}_i\}$, where each $\bar{\mathbf{g}}_i$ is then a pure literal. For XOR, simply notice that $S_{g_1} \otimes S_{\bar{g}_1} = \emptyset$. For ITE, notice that $S_{g_1} \otimes S_{\bar{g}_1} = \{\bar{g} \vee \mathbf{g}_2 \vee \mathbf{g}_3\}$, and then $\mathbf{g}_2$ and $\mathbf{g}_3$ are both pure literals.
2. Follows from $\mathrm{PG}(\mathcal{C}^\tau) \subseteq \mathrm{TST}(\mathcal{C}^\tau)$
3. (i) Follows directly from Lemma 3.
   (ii) By a similar argument as in Item 1 (ii).
   (iii) Follows directly from Lemmas 2 and 7. □

However, there are cases in which VE is not as effective as BCE. Namely, VE cannot achieve the effectiveness of MIR when applying TST, in contrast to BCE. To see this, notice that an input gate can have arbitrarily large finite fanout and still be monotone. On the other hand, VE cannot be applied on gates which have arbitrarily large fanout and fanin, since the elimination bound of VE can then be exceeded (number of clauses produced would be greater than the number of clauses removed). In general, a main point to notice is that for VE, in order to achieve the effectiveness of BCE (on the standard Tseitin encoding), one has to apply the Plaisted-Greenbaum encoding before applying VE. In addition, since VE is not confluent in contrast to BCE, in practice the variable elimination ordering heuristics for VE has to be good enough so that it forces the "right" elimination order. In addition, there are cases in which BCE is more effective than $\mathrm{VE}_{\mathrm{PG}}$. For some intuition on this, consider a clause C with blocking literal l. Notice that the result of performing VE on l is not dependent on whether C is removed. However, for any non-blocking literal $l' \in C$ the number of non-tautological clauses after applying VE on l' would be smaller if BCE would first remove C.

On the other hand, there are also cases in which the combination of BCE and VE can be more effective than applying BCE only. For instance, by applying VE on a CNF, new blocked clauses may arise. For more concreteness, consider a circuit with an XOR-gate $g := \mathrm{XOR}(g_1, g_2)$ where g_1 and g_2 are input gates with fanout one (non-shared). Assume that $g := \mathrm{XOR}(g_1, g_2)$ is rewritten as an AND-OR circuit structure $g := \mathrm{AND}(a, b)$, $a := \mathrm{OR}(g_1, g_2)$, $b := \mathrm{OR}(\mathrm{NOT}(g_1), \mathrm{NOT}(g_2))$, where a and b are newly introduced gates with fanout one. Notice that g_1 and g_2 now have fanout two. In the Tseitin encoding of this structure, BCE cannot see the non-sharedness of g_1 and g_2 in the underlying XOR. However, by first eliminating the OR-gates a and b with VE, BCE can then remove the clauses containing the variables g_1 and g_2 (the gates become implicitly "non-shared" again). In other words, there are cases in which variable elimination results in additional clauses to be blocked.

7 Implementation

In short, BCE can be implemented in a similar way as VE in the SatElite preprocessor [10]: first "touch" all literals. Then, as long as there is a touched literal l: find clauses that are blocked by l, mark l as not touched any more, remove these blocked clauses,

and touch the negation of all literals in these clauses. The priority list of touched literals can be ordered by the number of occurrences. Literals with few occurrences of their negations are to be tried first. This algorithm is implemented in PrecoSAT version 465 (http://fmv.jku.at/precosat) and can be used to run BCE until completion.

In principle, the result is unique. However, as in our implementation of VE [10] in PrecoSAT, we have a heuristic cut-off limit in terms of the number of occurrences of a literal. If the number of occurrences of a literal is too large, then we omit trying to find blocked clauses for its negation. This may prevent the actual implementation from removing some blocked clauses. In general, however, as also witnessed by the results of using BCE on the CNFs generated with the Tseitin and Plaisted-Greenbaum encodings, this cut-off heuristic does not have any measurable effect.

8 Experiments

We evaluated how much reduction can be achieved using BCE in combination with VE and various circuit encoding techniques. Reduction is measured in the size of the CNF before and after preprocessing, and on the other hand, as gain in the number of instances solved.

We used all formulas of SMT-Lib (http://smtlib.org) over the theory of bit-vectors (QF_BV) made available on July 2, 2009, as a practice benchmark set for the SMT competition 2009. From these we removed the large number of mostly trivial SAGE examples. The remaining 3672 SMT problems were bit-blasted to And-Inverter Graphs (AIGs) in the AIGER format (http://fmv.jku.at/aiger) using our SMT solver Boolector [22]. Furthermore, we used the AIG instances used in [5], consisting of two types of instances: (i) AIGs representing BMC problems (with step bound $k = 45$) obtained from all the 645 sequential HWMCC'08 (http://fmv.jku.at/hwmcc08) model checking problems, and (ii) 62 AIGs from the structural SAT track of the SAT competition. We have made the SMT-Lib instances publicly available at http://fmv.jku.at/aiger/smtqfbv-aigs.7z (260MB); the others cannot be distributed due to license restrictions. However, the HWMCC'08 instances can easily be regenerated using publicly available tools [3] and the model checking benchmarks available at http://fmv.jku.at/hwmcc08. We encoded these 4379 structural SAT instances with four algorithms: the standard Tseitin encoding [13], the Plaisted-Greenbaum polarity-based encoding [14], the Minicirc encoder based on technology mapping [3] and VE, and the most recent NiceDAG encoder [4,5]. The NiceDAG implementation was obtained from the authors. For Minicirc, we used an improved implementation of Niklas Eén.

In order to additionally experiment with application benchmarks already in CNF, we also included 292 CNFs of the application track of the SAT competition 2009 to our benchmark set. All resulting CNFs were preprocessed with VE alone (further abbreviated e), and separately first with BCE (b), followed by VE (e), and both repeated again, which altogether gives 6 versions of each CNF (no BCE or VE, e, b, be, beb, bebe). We call such an application of one preprocessing algorithm, either BCE or VE, which is run to completion, a *preprocessing phase*.

[3] Notice that COI is performed already in the generation process by these tools. However, we did not implement the non-trivial NSI or MIR for the experiments.

Table 2. Effectiveness of BCE in combination with VE using various encoders

	encoding			b			be			beb			bebe			e		
	t	V	C	t	V	C	t	V	C	t	V	C	t	V	C	t	V	C
S U	0	46	256	2303	29	178	1042	11	145	1188	11	145	569	11	144	2064	11	153
A T	12	9	27	116	7	18	1735	1	8	1835	1	6	34	1	6	244	1	9
A P	10	9	20	94	7	18	1900	1	6	36	1	6	34	1	6	1912	1	6
A M	190	1	8	42	1	7	178	1	7	675	1	7	68	1	7	48	1	8
A N	9	3	10	50	3	10	1855	1	6	36	1	6	34	1	6	1859	1	6
H T	147	121	347	1648	117	277	2641	18	118	567	18	118	594	18	116	3240	23	140
H P	130	121	286	1398	117	277	2630	18	118	567	18	118	595	18	116	2835	19	119
H M	6961	16	91	473	16	84	621	12	78	374	12	77	403	12	76	553	15	90
H N	134	34	124	573	34	122	1185	17	102	504	17	101	525	17	100	1246	17	103
B T	577	442	1253	5799	420	1119	7023	57	321	1410	56	310	1505	52	294	8076	64	363
B P	542	442	1153	5461	420	1119	7041	57	321	1413	56	310	1506	52	294	7642	57	322
B M	10024	59	311	1252	58	303	1351	53	287	1135	53	286	1211	52	280	1435	55	303
B N	13148	196	643	2902	193	635	4845	108	508	2444	107	504	2250	105	500	5076	114	518

The results are presented in Table 2. The first column lists the benchmark family: S = SAT'09 competition, A = structural SAT track, H = HWMCC'08, B = bit-blasted bit-vector problems from SMT-Lib. These are all AIGs except for the CNF instances in S. The next column gives the encoding algorithm used: T = Tseitin, P = Plaisted-Greenbaum, M = Minicirc, N = NiceDAG, and U = unknown for the S family already in CNF. The t columns give the sum of the time in seconds spent in one encoding/preprocessing phase. The columns V and C list in millions the sum of numbers of variables and clauses over all produced CNFs in each phase.

The results show that the combination "be" of BCE and VE always gives better results than VE (e) alone, with comparable speed. Using a second phase (beb) of BCE gives further improvements, even more if VE is also applied a second time (bebe). The CNF sizes after applying BCE (b) for the P encoder and the T encoder are equal, as expected. Further preprocessing, however, diverges: since clauses and literals are permuted, VE is not confluent, and thus VE phases can produce different results.

We applied a time limit of 900 seconds and a memory limit of 4096 MB for each encoder and each preprocessing phase. Thus 139 out of $106848 = 6 \cdot (4 \cdot 4379 + 292)$ CNFs were not generated: HM encoding ran out of memory on 5 very large BMC instances, one large CNF in S could not be preprocessed at all, and there was a problem with the parser in NiceDAG, which could not parse 14 actually rather small AIGs in BN. Furthermore, there were 10 timeouts for various preprocessing phases in the A family: 2 in AT/beb, 2 in AN/be, 2 in AN/e, 2 in AP/be, and 2 in AP/e. However, except for the one large CNF, where also VE run out of memory, there is not a single case where BCE did not run until completion within the given time and memory limits.

Reducing the size of a CNF by preprocessing does not necessarily lead to faster running times. Since it was impossible to run all structural instances with an appropriate time limit, we only performed preliminary experiments with a very small time limit of 90 seconds. We used PrecoSAT v236, the winner of the application track of the SAT competition 2009, and PicoSAT v918, a fast clause learning solver which does not use

sophisticated preprocessing algorithms, in contrast to PrecoSAT. In both cases the results were inconclusive. Running preprocessing until completion takes a considerable portion of the 90 seconds time limit, even if restricted to VE. In addition, the success of PrecoSAT shows that not running preprocessing until completion is a much better strategy, particularly if the preprocessor is run repeatedly again, with enough time spent on search in-between. However, this strategy is hard to evaluate when many preprocessing techniques are combined.[4] Therefore we decided to stick with the run-to-completion approach, which also gives some clear indication of how much CNF size reduction can be achieved through BCE.

For the 292 SAT competition instances we were able to run PrecoSAT with a more reasonable timeout of 900 seconds. The cluster machines used for the experiments, with Intel Core 2 Duo Quad Q9550 2.8 GHz processor, 8 GB main memory, running Ubuntu Linux version 9.04, are around two times as fast as the ones used in the first phase of the 2009 SAT competition. In the first phase of the competition, with a similar time limit, PrecoSAT solved many more instances than competitors. Nevertheless, using BCE we can improve the number of solved instances considerable: PrecoSAT solves 176 original instances, 177 preprocessed by BCE and VE alone (b and e), 179 be instances, 180 beb instances, and 183 bebe instances. If we accumulate the time for all the preprocessing phases and add it to the actual running time, then 181 instances can be solved in the last case. For the other cases the number of solved instances does not change.

It would be interesting to compare our results to pure circuit-level solvers. To our understanding, however, such solvers have not proven to be more efficient than running CNF solvers in combination with specialized circuit to CNF encodings.

9 Conclusions

We study a CNF-level simplification technique we call BCE (blocked clause elimination). We show that BCE, although a simple concept, is surprisingly effective: without any explicit knowledge of the underlying circuit structure, BCE achieves the same simplifications as combinations of circuit-level simplifications and the well-known polarity-based Plaisted-Greenbaum CNF encoding. This implies that the effect of such specialized circuit-level techniques can actually be accomplished directly on the CNF-level. To our best knowledge, these connections have not been known before. Furthermore, in contrast to specialized circuit-level techniques, BCE can be naturally applied on any CNF formula, regardless of its origin. Experimental results with an implementation of a CNF-level preprocessor combining BCE and SatElite-style variable elimination are presented, showing the effectiveness and possible benefits of applying BCE.

Acknowledgements. The authors thank Niklas Eén and Pete Manolios for providing up-to-date versions of the Minicirc and NiceDAG encoders used in the experiments. The first author is financially supported by Academy of Finland under the project "Extending the Reach of Boolean Constraint Reasoning" (#132812). The third author is supported by the Dutch Organization for Scientific Research under grant 617.023.611.

[4] In PrecoSAT v465, we have failed literal preprocessing, various forms of equivalence reasoning, explicit pure literal pruning, BCE, VE, combined with on-the-fly subsumption.

References

1. Jackson, P., Sheridan, D.: Clause form conversions for Boolean circuits. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 183–198. Springer, Heidelberg (2005)
2. Mishchenko, A., Chatterjee, S., Brayton, R.K.: DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In: DAC 2006, pp. 532–535. ACM, New York (2006)
3. Eén, N., Mishchenko, A., Sörensson, N.: Applying logic synthesis for speeding up SAT. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 272–286. Springer, Heidelberg (2007)
4. Manolios, P., Vroon, D.: Efficient circuit to CNF conversion. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 4–9. Springer, Heidelberg (2007)
5. Chambers, B., Manolios, P., Vroon, D.: Faster SAT solving with better CNF generation. In: DATE 2009, pp. 1590–1595. IEEE, Los Alamitos (2009)
6. Ostrowski, R., Grégoire, É., Mazure, B., Sais, L.: Recovering and exploiting structural knowledge from CNF formulas. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 185–199. Springer, Heidelberg (2002)
7. Brafman, R.I.: A simplifier for propositional formulas with many binary clauses. IEEE Transactions on Systems, Man, and Cybernetics, Part B 34(1), 52–59 (2004)
8. Bacchus, F.: Enhancing Davis Putnam with extended binary clause reasoning. In: AAAI 2002, pp. 613–619. AAAI Press, Menlo Park (2002)
9. Subbarayan, S., Pradhan, D.K.: NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 276–291. Springer, Heidelberg (2005)
10. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005)
11. Gershman, R., Strichman, O.: Cost-effective hyper-resolution for preprocessing CNF formulas. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 423–429. Springer, Heidelberg (2005)
12. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics 96-97, 149–176 (1999)
13. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Siekmann, J., Wrightson, G. (eds.) Automation of Reasoning 2: Classical Papers on Computational Logic 1967–1970, pp. 466–483. Springer, Heidelberg (1983)
14. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. Journal of Symbolic Computation 2(3), 293–304 (1986)
15. Biere, A., Clarke, E.M., Raimi, R., Zhu, Y.: Verifiying safety properties of a power PC microprocessor using symbolic model checking without BDDs. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 60–71. Springer, Heidelberg (1999)
16. Jussila, T., Biere, A.: Compressing BMC encodings with QBF. Electronic Notes in Theoretical Computer Science 174(3), 45–56 (2007)
17. Drechsler, R., Junttila, T., Niemelä, I.: Non-clausal SAT and ATPG. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 655–694. IOS Press, Amsterdam (2009)
18. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM 7(3), 201–215 (1960)
19. Purdom, P.W.: Solving satisfiability with less searching. IEEE Transactions on Pattern Analysis and Machine Intelligence 6(4), 510–513 (1984)

20. Kautz, H.A., Ruan, Y., Achlioptas, D., Gomes, C.P., Selman, B., Stickel, M.E.: Balance and filtering in structured satisfiable problems. In: IJCAI 2001, pp. 351–358. Morgan Kaufmann, San Francisco (2001)
21. Heule, M.J.H., Verwer, S.: Using a satisfiability solver to identify deterministic finite state automata. In: BNAIC 2009, pp. 91–98 (2009)
22. Brummayer, R., Biere, A.: Boolector: An efficient SMT solver for bit-vectors and arrays. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 174–177. Springer, Heidelberg (2009)