Blockedness in Propositional Logic: Are You Satisfied With Your Neighborhood?*

Benjamin Kiesl
TU Wien, Austria
kiesl@kr.tuwien.ac.at

Martina Seidl
JKU Linz, Austria
martina.seidl@jku.at

Hans Tompits
TU Wien, Austria
tompits@kr.tuwien.ac.at

Armin Biere
JKU Linz, Austria
biere@jku.at

Abstract

Clause-elimination techniques that simplify formulas by removing redundant clauses play an important role in modern SAT solving. Among the types of redundant clauses, *blocked clauses* are particularly popular. For checking whether a clause C is blocked in a formula F, one only needs to consider the so-called *resolution neighborhood* of C, i.e., the set of clauses that can be resolved with C. Because of this, blocked clauses are referred to as being *locally redundant*. In this paper, we discuss powerful generalizations of blocked clauses that are still locally redundant, namely *set-blocked clauses* and *super-blocked clauses*. We furthermore present complexity results for deciding whether a clause is set-blocked or super-blocked.

1 Introduction

Every once in a while, you hear about neighbors who are just not the kind of neighbors you would wish for. For instance, the grumpy dude next door who starts mowing his lawn on an early Sunday morning while you are still lying in bed. Or, the student in the apartment below yours, who regularly decides around midnight that now might be the perfect time for listening to some not so calm rock music. Whatever it is that makes them such pleasant neighbors, in case they decide to leave your neighborhood, you rarely find anyone shedding tears over their departure.

In this paper, we show that bad neighbors also play a significant role in modern SAT solving. They might not rob you of your well-earned sleep, but—rather subtly—they present themselves in the form of *locally redundant clauses*. Here, *redundant* means that we can remove these clauses from a formula in conjunctive normal form without causing any sorrow, i.e., without affecting the formula's satisfiability. *Local* means that we do not need to consider the whole formula to find out that these clauses are redundant—it suffices to consider only their *resolution neighborhood*, i.e., the set of clauses they can be resolved with.

*This work has been supported by the Austrian Science Fund (FWF) under projects W1255-N23 and S11408-N23.

Among the types of locally redundant clauses, *blocked clauses* [Kullmann, 1999] are of particular interest. Informally, a clause C is *blocked* in a CNF-formula F if it contains a literal l such that all possible resolvents of C upon l are tautologies. The elimination of blocked clauses can significantly boost the performance of SAT solvers [Manthey *et al.*, 2013; Järvisalo *et al.*, 2010]. Moreover, blocked clauses yield the basis for blocked-clause decomposition [Heule and Biere, 2013], a technique that splits a formula into two parts that become solvable by blocked-clause elimination. Blocked-clause decomposition is successfully used for gate extraction, for efficiently finding backbone variables, and for the detection of implied binary equivalences [Iser *et al.*, 2015; Balyo *et al.*, 2014]. The winner of the SATRace 2015 competition, abcdSAT [Chen, 2015], uses blocked-clause decomposition as core technology.

These success stories provide the motivation for having a closer look at locally redundant clauses in general, and at blocked clauses in particular. In this paper, we discuss powerful generalizations of blocked clauses that are still locally redundant: *set-blocked clauses* and *super-blocked clauses*. The latter kind of clauses actually constitutes the most general concept of a locally redundant clause. Rounding off the picture, we furthermore discuss the complexity of deciding whether a clause is set-blocked or super-blocked.

The rest of this paper is structured as follows. After introducing the necessary preliminaries in Section 2, we recapitulate popular kinds of locally redundant clauses in Section 3. Afterwards, Section 4 contains our discussion of set-blocked clauses and super-blocked clauses, and Section 5 contains complexity results related to checking these redundancy notions. Section 6 concludes our paper with an outlook on future work.

This paper is an abridged version of a paper that appeared in the proceedings of IJCAR 2016, the Eight International Joint Conference on Automated Reasoning [Kiesl *et al.*, 2016].

2 Preliminaries

We consider propositional formulas in *conjunctive normal form* (CNF), which are defined as follows. A *literal* is either a variable x (a *positive literal*) or the negation $\neg x$ of a variable x (a *negative literal*). For a literal l, we define its *complement* $\overline{l}$ as $\neg x$ if $l = x$ and as x if $l = \neg x$. Accordingly,

for a set L of literals, we define $\bar{L} = \{\bar{l} \mid l \in L\}$. A *clause* is a disjunction of literals. A *formula* in CNF is a conjunction of clauses. We identify a clause with a set of literals and a formula with a set of clauses. A *tautology* is a clause that contains both l and $\bar{l}$ for some literal l. For a literal, clause, or formula F, $var(F)$ denotes the set of variables occurring in F. For convenience, we often treat $var(F)$ as a variable if F is a literal. For a set L of literals and a formula F, we denote by F_L the set of all clauses in F that contain a literal of L, i.e., $F_L = \{C \mid C \in F, C \cap L \neq \emptyset\}$. In case L is a singleton set of the form $\{l\}$, we sometimes write F_l instead of $F_{\{l\}}$.

An *assignment* is a partial function from a set of variables to the truth values 1 (*true*) and 0 (*false*). Given an assignment α and a literal l, α_l is the assignment obtained from α by flipping the truth value of l, i.e., $\alpha_l(v) = 1 - \alpha(v)$ if $v = var(l)$ and $\alpha_l(v) = \alpha(v)$ otherwise. A literal l is *satisfied* by an assignment α if l is positive and $\alpha(var(l)) = 1$ or if it is negative and $\alpha(var(l)) = 0$. A clause is satisfied by an assignment α if it contains a literal that is satisfied by α. Finally, a formula is satisfied by an assignment α if all of its clauses are satisfied by α. A formula is *satisfiable* if there exists an assignment that satisfies it. Two formulas are *logically equivalent* if they are satisfied by the same assignments. Two formulas F and F' are *satisfiability equivalent* if F is satisfiable if and only if F' is satisfiable.

Given two clauses C and D together with a literal $l \in C$ such that $\bar{l} \in D$, the clause $(C \setminus \{l\}) \cup (D \setminus \{\bar{l}\})$ is called the *resolvent* of C and D upon l. Given a formula F and a clause C, the *resolution neighborhood*, $\mathsf{RN}_F(C)$, of C in F is the set of all clauses in F that can be resolved with C:

$$\mathsf{RN}_F(C) = \{D \in F \mid \exists l \in D \text{ such that } \bar{l} \in C\}.$$

The variables in $var(C)$ are called *local variables* and the variables in $var(\mathsf{RN}_F(C)) \setminus var(C)$ are the *external variables*, denoted by $ext_F(C)$.

Next, we recapitulate the formal notion of clause redundancy [Kiesl *et al.*, 2016]. Intuitively, a clause C is redundant w.r.t. a formula F if neither its addition to F nor its removal from F changes the satisfiability or unsatisfiability of F.

Definition 1 *A clause C is* redundant *w.r.t. a formula F if the formulas $F \setminus \{C\}$ and $F \cup \{C\}$ are satisfiability equivalent. A* redundancy property *is a set of pairs (F, C) where C is redundant w.r.t. F. Finally, for two redundancy properties $\mathcal{P}_1$ and $\mathcal{P}_2$, $\mathcal{P}_1$ is* more general *than $\mathcal{P}_2$ if $\mathcal{P}_2 \subseteq \mathcal{P}_1$. Accordingly, $\mathcal{P}_1$ is* strictly more general *than $\mathcal{P}_2$ if $\mathcal{P}_2 \subset \mathcal{P}_1$.*

Consider, for example, the formula $F = \{(a \vee b), (\neg a \vee \neg b)\}$. The clause $C = (\neg a \vee \neg b)$ is redundant w.r.t. F since the formulas $F \setminus \{C\}$ and $F \cup \{C\}$ are satisfiability equivalent (although they are not logically equivalent). Moreover, the set $\{(F, C) \mid F \text{ is a formula and } C \text{ is a tautology}\}$ is a redundancy property since for every formula F and every tautology C, $F \setminus \{C\}$ is satisfiability equivalent to $F \cup \{C\}$.

Note that C is *not* redundant w.r.t. F if and only if $F \setminus \{C\}$ is satisfiable and $F \cup \{C\}$ is unsatisfiable. To prove that C is redundant w.r.t. F, it therefore suffices to show that the satisfiability of $F \setminus \{C\}$ implies that of $F \cup \{C\}$. Redundancy properties as defined above yield not only the basis for clause-elimination but also for clause-addition procedures [Järvisalo *et al.*, 2012].

3 Local Redundancy Properties

In this section, we discuss some well-known redundancy properties from the literature to motivate the definition of so-called *local redundancy properties*. The simplest example of a redundant clause is a *tautology*. As every assignment satisfies exactly one of the two complementary literals contained in a tautology, tautologies are always true and therefore redundant w.r.t. every formula. Because of this, the set $\{(F, C) \mid F \text{ is a formula and } C \text{ is a tautology}\}$ is a redundancy property. To decide whether a clause is a tautology w.r.t. a formula F, we only need to look at C itself.

Another example for redundant clauses are so-called *clauses with pure literals* [Davis and Putnam, 1960]. A literal is *pure* in a formula if its complementary literal does not occur in the formula. In contrast to tautologies, clauses with pure literals are not guaranteed to be satisfied by every assignment. Still, we can easily see that they are redundant: Let C be a clause containing a literal l that is pure in a formula F and assume we have some assignment α that satisfies $F \setminus \{C\}$ but falsifies C. We can then turn α into a satisfying assignment α_l of $F \cup \{C\}$ by flipping the truth value of l. None of the clauses in $F \setminus \{C\}$ can be falsified by this since they do not contain the literal $\bar{l}$, which is the only literal whose truth value is negatively affected by setting l to true. The corresponding redundancy property is the set $\{(F, C) \mid C \text{ contains a literal that is pure in } F\}$.

The redundancy of a clause with pure literals is based on the fact that we can make one of its literals true without falsifying other clauses. In the case of pure literals, this is guaranteed since there are no clauses that contain the complementary literal. The more general *blocked clauses* [Kullmann, 1999] guarantee redundancy even in cases where other clauses might contain the complementary literal:

Definition 2 *A clause C is blocked by a literal $l \in C$ in a formula F if, for each clause $D \in F_{\bar{l}}$, $C \cup (D \setminus \{\bar{l}\})$ is a tautology.*

We say that a clause is blocked in F if one of its literals blocks it in F. The following is an example of a blocked clause:

Example 1 *Consider the clause $C = (a \vee b)$ and the formula $F = \{(\neg a \vee c), (\neg b \vee \neg a)\}$. The literal b blocks C in F since the only clause in F that contains $\neg b$ is the clause $D = (\neg b \vee \neg a)$, and $C \cup (D \setminus \{\bar{l}\}) = (a \vee b \vee \neg a)$ is a tautology.*

To see that blocked clauses are redundant, let C be a clause that is blocked by a literal l in a formula F and assume that some assignment α satisfies $F \setminus \{C\}$ but falsifies C. Like with pure literals, we can then easily turn α into a satisfying assignment α_l of C by flipping the truth value of l. This flipping could possibly falsify some of the clauses in $F \setminus \{C\}$ that contain $\bar{l}$, but the condition that l blocks C guarantees that these clauses stay satisfied: Let D be such a clause that contains $\bar{l}$. Then, since $C \cup (D \setminus \{\bar{l}\})$ is a tautology, either D is itself a tautology (and therefore trivially satisfied) or it contains a literal $l' \neq l$ such that $\bar{l}' \in C$. In the latter case, since α falsifies C, it must satisfy l' and since α_l agrees with α on all literals but l, α_l is a satisfying assignment of $F \cup \{C\}$. Hence, C is redundant w.r.t. F. We illustrate this argument on a concrete example:

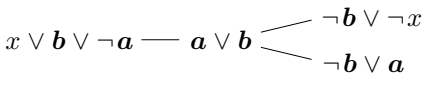

Figure 1: The clause $(a \vee b)$ from Example 3 and its resolution neighborhood.

Example 2 *Consider again the clause $C = (a \vee b)$ and the formula $F = \{(\neg a \vee c), (\neg b \vee \neg a)\}$ from Example 1. We already know that b blocks C in F. Now, let α denote the assignment that falsifies the variables a, b, and c. Clearly, α satisfies F but falsifies C. We would expect that the assignment α_b, obtained from α by flipping the truth value of b, satisfies not only C but also all clauses of F. This is indeed the case. The only clause that could have been falsified by flipping the truth value of b is the clause $D = (\neg b \vee \neg a)$. But D stays true since α_b satisfies $\neg a$, which is not a coincidence: $C \cup (D \setminus \{\neg b\})$ is a tautology and must therefore be satisfied by α_b. Moreover, we know that α falsifies C. Therefore, α_b must satisfy $D \setminus \{\neg b\} = (\neg a)$.*

A closer look at blocked clauses reveals two things. First, the redundancy property of blocked clauses (i.e., the set $\{(F, C) \mid C$ is blocked in $F\}$) is more general than the redundancy properties of tautologies and pure literals. If a clause C is a tautology, then $C \cup (D \setminus \{l\})$ is a tautology for every D, implying that C is blocked. If C contains a literal that is pure in a formula F, then $F_{\bar{l}} = \emptyset$ and thus it vacuously holds that $C \cup (D \setminus \{l\})$ is a tautology for all $D \in F_{\bar{l}}$. Therefore, tautologies and clauses with pure literals are blocked clauses.

Second, to detect whether a clause C is blocked in a formula F, it suffices to consider only those clauses of F that contain a literal $\bar{l}$ such that $l \in C$, i.e., the clauses in $\mathsf{RN}_F(C)$. The question arises whether there exist redundant clauses that are not blocked but whose redundancy can be identified by considering only their resolution neighborhood. As shown in the next example, this is indeed the case:

Example 3 *Consider $C = (a \vee b)$ and let F be a formula such that $\mathsf{RN}_F(C) = \{(x \vee b \vee \neg a), (\neg b \vee \neg x), (\neg b \vee a)\}$ (cf. Figure 1). Clause C is not blocked in F but redundant: Suppose there exists an assignment α that satisfies F but falsifies C. Then, α must satisfy either x or $\neg x$. If $\alpha(x) = 1$, then C can be satisfied by flipping the truth value of a, resulting in assignment $\alpha' = \alpha_a$. Since $\alpha'(x) = 1$, the clause $(x \vee b \vee \neg a)$ stays satisfied. In contrast, if $\alpha(x) = 0$, we can satisfy C by the assignment α'', obtained from α by flipping the truth values of both a and b: The fact that $\alpha''(b) = 1$ guarantees that $(x \vee b \vee \neg a)$ stays satisfied whereas $\alpha''(x) = 0$ and $\alpha''(a) = 1$ guarantee that both $(\neg b \vee \neg x)$ and $(\neg b \vee a)$ stay satisfied. Since flipping the truth values of literals in C does not affect the truth of clauses outside the resolution neighborhood $\mathsf{RN}_F(C)$ of C, we obtain in both cases a satisfying assignment of F.*

If a clause C is redundant w.r.t. a formula F and this redundancy can be identified by considering only its resolution neighborhood in F, then C is redundant w.r.t. every formula F' in which it has the same resolution neighborhood as in F. Such a redundancy is called *local* [Kiesl *et al.*, 2016]:

Definition 3 *A redundancy property $\mathcal{P}$ is* local *if, for every clause C and any two formulas F, F' with $\mathsf{RN}_F(C) = \mathsf{RN}_{F'}(C)$, it holds that either $\{(F, C), (F', C)\} \subseteq \mathcal{P}$ or $\{(F, C), (F', C)\} \cap \mathcal{P} = \emptyset$.*

In the next section, we present redundancy properties that are strictly more general than blocked clauses while still being local.

4 Set-Blocking and Super-Blocking

We now turn our attention to the local redundancy properties of set-blocked clauses and super-blocked clauses [Kiesl *et al.*, 2016]. Both strictly generalize blocked clauses. Super-blocked clauses even constitute the most general local redundancy property. The idea behind set-blocked clauses is to allow for flipping the truth values of more than one literal:

Definition 4 *Let F be a formula and C a clause. A non-empty set $L \subseteq C$* blocks *C in F if, for each clause $D \in F_{\bar{L}}$, $(C \setminus L) \cup \bar{L} \cup D$ is a tautology.*

We say that a clause is set-blocked in F if there exists a set that blocks it in F. Moreover, we write SET for the set $\{(F, C) \mid C$ is set-blocked in $F\}$.

Example 4 *Let $C = (a \vee b)$ and $F = \{(\neg a \vee b), (\neg b \vee a)\}$. Then, C is set-blocked by $L = \{a, b\}$: Clearly, $F_{\bar{L}} = F$ and $C \setminus L = \emptyset$. Therefore, for $D_1 = (\neg a \vee b)$ we get that $(C \setminus L) \cup \bar{L} \cup D_1 = (a \vee b \vee \neg a)$ is a tautology and for $D_2 = (\neg b \vee a)$ we get that $(C \setminus L) \cup \bar{L} \cup D_2 = (a \vee b \vee \neg b)$ is a tautology as well. Note that C is not blocked in F.*

Given an assignment α that satisfies $F \setminus \{C\}$ but falsifies C, the existence of a blocking set L guarantees that we can turn α into a satisfying assignment α_L of $F \cup \{C\}$ by flipping the truth values of all the literals in L. Since $(C \setminus L) \cup \bar{L} \cup D$ is a tautology for every $D \in F_{\bar{L}}$, at least one of the following holds: (i) D is itself a tautology and thus satisfied by α_L, or (ii) D contains a literal of L which is satisfied by α_L since its truth value is flipped, or (iii) D contains a literal l which is satisfied since $\bar{l} \in C$ is falsified by α and the truth value of l is not flipped. Hence, α_L satisfies $F \cup \{C\}$ and thus set-blocked clauses are redundant.

In order for a clause to be set-blocked, it is required that *every* assignment that satisfies $F \setminus \{C\}$ but falsifies C can be turned into a satisfying assignment of $F \setminus \{C\}$ by flipping the truth values of the literals in the blocking set L. This involves some inflexibility as we are not allowed to choose different blocking sets for different assignments. Consider the following example:

Example 5 *Let $C = (a \vee b)$ and $F = \{(\neg a \vee x), (\neg b \vee \neg x)\}$. The clause C is not set-blocked in F but redundant w.r.t. F. Suppose an assignment α satisfies F but falsifies C. We can easily turn α into a satisfying assignment of $F \cup \{C\}$: If α satisfies x, flip the truth value of a. If α falsifies x, flip the truth value of b. In both cases, all clauses in F stay satisfied.*

We obtain a more general redundancy property if we allow blocking sets to depend on the assignment at hand. In particular, on the assignment over the external variables. This gives rise to the notion of *super-blocking*. In the following,

for a formula F and an assignment α, we denote by $F|\alpha$ the set of clauses obtained from F by removing all clauses that are satisfied by α. Recall that the external variables, $ext_F(C)$, are the variables that are contained in $\mathsf{RN}_F(C)$ but not in C.

Definition 5 *A clause C is* super-blocked *in a formula F if, for every assignment τ over the variables in $ext_F(C)$, C is set-blocked in $F|\tau$.*

We write SUP for the set $\{(F, C) \mid C \text{ is super-blocked in } F\}$. For instance, the clause C in Example 5 is not set-blocked but super-blocked in F since it is set-blocked in $F|\alpha$ and $F|\alpha'$ for $\alpha(x) = 1$ and $\alpha'(x) = 0$. Likewise for the clause C and the corresponding formula F in Example 3. By removing the clauses that are satisfied by a certain assignment, we use the fact that the truth of clauses in the resolution neighborhood cannot only be established by literals whose variables occur in C, but also by the truth of literals over the external variables. Finally, note that if a clause is set-blocked in F, then it is also set-blocked in every $F' \subseteq F$ and thus in every $F|\alpha$. Hence we get:

Proposition 1 *Super-blocking is strictly more general than set-blocking, i.e., it holds that* $\mathsf{SET} \subset \mathsf{SUP}$.

One can show that super-blocked clauses constitute the most general local redundancy property [Kiesl *et al.*, 2016]:

Theorem 2 *The set* SUP *is the most general local redundancy property.*

5 Complexity Results

In this section, we discuss complexity results for deciding whether a clause is set-blocked or super-blocked. We furthermore consider the complexity of deciding restricted variants of set-blocking and super-blocking that are obtained by bounding the size of blocking sets. As the redundancy properties of set-blocked clauses and super-blocked clauses are local, we consider only a clause and its resolution neighborhood as input to the corresponding decision problems.

Formally, the *set-blocking problem* is the following decision problem: Given a clause C and a formula F such that every $D \in F$ contains a literal $\bar{l}$ with $l \in C$, decide whether C is set-blocked in F. The *super-blocking problem* is defined analogously.

Clearly, the set-blocking problem is in NP: For a non-empty literal set $L \subseteq C$, we can check in polynomial time whether it blocks C in F—we just have to iterate over all clauses in $F_{\bar{L}}$ and check whether $(C \setminus L) \cup \bar{L} \cup D$ is a tautology. The following is thus an NP-procedure for deciding set-blockedness: Guess a non-empty set $L \subseteq C$ and check if it blocks C in F. Hardness for the complexity class NP can be shown via a reduction from the satisfiability problem of propositional logic. The set-blocking problem is thus NP-complete.

Deciding whether a clause is super-blocked is harder: Membership and hardness for Π_2^P can respectively be shown by an encoding to and a reduction from 2-QBF ($\forall\exists$-SAT). The intuition behind this is that super-blocking is "for-all-exists" in nature—a clause C is super-blocked in a formula F if *for all* assignments α over the external variables, there *exists* a non-empty set $L \subseteq C$ of literals such that C is set-blocked in $F|\alpha$.

	$\lvert L\rvert$ unrestricted	$\lvert L\rvert \leq k$ for $k \in \mathbb{N}^+$
Super-Blocking	Π_2^P-complete	co-NP-complete
Set-Blocking	NP-complete	P

Table 1: Summary of Complexity Results

Although the set-blocking problem is NP-complete in the general case, we can obtain a restricted variant of set-blocking by only allowing blocking sets whose cardinalities are bounded by a constant. Then, the resulting problem of testing whether a clause C is blocked by some non-empty set $L \subseteq C$, whose size is at most k for $k \in \mathbb{N}^+$, turns out to be polynomial: For a finite set C and $k \in \mathbb{N}^+$, there are only polynomially many non-empty subsets $L \subseteq C$ with $|L| \leq k$. To see this, observe (by basic combinatorics) that the exact number of such subsets is given by the sum $\sum_{i=1}^{k} \binom{|C|}{i}$, which reduces to a polynomial with degree at most k.

Hence, the number of non-empty subsets $L \subseteq C$ with $|L| \leq k$ is polynomial in the size of C. To decide whether a clause is set-blocked by a set of size at most k we thus just have to iterate over polynomially many candidate sets and check, in polynomial time, whether they block C. Likewise, bounding the size of the blocking sets in the definition of super-blocked clauses by a constant makes the corresponding decision problem simpler: Deciding super-blocking for bounded blocking sets is co-NP-complete.

The complexity results are summarized in Table 1 (from [Kiesl *et al.*, 2016]). Note that the cardinality $|L|$ of blocking sets is of course bounded by the length of the clauses. This is particularly interesting for formula instances with (uniform) constant or maximal clause length.

6 Conclusion

We discussed local redundancy properties and considered set-blocked clauses and super-blocked clauses, which generalize blocked clauses while still being locally redundant, meaning that their redundancy can be identified by considering only their resolution neighborhood. In fact, super-blocked clauses constitute the most general local redundancy property. Deciding whether a clause is set-blocked is NP-complete and deciding whether a clause is super-blocked is Π_2^P-complete. Bounding the size of the blocking sets by a constant makes the corresponding decision problems go down one level in the polynomial hierarchy. The locality of redundancy properties is particularly interesting when dealing with formulas in which the resolution neighborhoods of clauses are small compared to the size of the whole formula. Concerning future work, we plan to implement clause-elimination techniques based on set-blocked clauses and super-blocked clauses. Moreover, we want to investigate whether these notions can be lifted to the level of quantified Boolean formulas.

References

[Balyo *et al.*, 2014] Tomas Balyo, Andreas Fröhlich, Marijn Heule, and Armin Biere. Everything you always wanted to know about blocked sets (but were afraid to ask). In Carsten Sinz and Uwe Egly, editors, *Proc. of the 17th Int. Conference on Theory and Applications of Satisfiability Testing* (*SAT 2014*), volume 8561 of *LNCS*, pages 317–332, Cham, 2014. Springer.

[Chen, 2015] Jingchao Chen. Fast blocked clause decomposition with high quality. *CoRR*, abs/1507.00459, 2015.

[Davis and Putnam, 1960] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[Heule and Biere, 2013] Marijn Heule and Armin Biere. Blocked clause decomposition. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning* (*LPAR-19*), volume 8312 of *LNCS*, pages 423–438, Heidelberg, 2013. Springer.

[Iser *et al.*, 2015] Markus Iser, Norbert Manthey, and Carsten Sinz. Recognition of nested gates in CNF formulas. In Marijn Heule and Sean Weaver, editors, *Proc. of the 18th Int. Conference on Theory and Applications of Satisfiability Testing* (*SAT 2015*), volume 9340 of *LNCS*, pages 255–271, Cham, 2015. Springer.

[Järvisalo *et al.*, 2010] Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In Javier Esparza and Rupak Majumdar, editors, *Proc. of the 16th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems* (*TACAS 2010*), volume 6015 of *LNCS*, pages 129–144, Heidelberg, 2010. Springer.

[Järvisalo *et al.*, 2012] Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Proc. of the 6th Int. Joint Conference on Automated Reasoning* (*IJCAR 2012*), volume 7364 of *LNCS*, pages 355–370, Heidelberg, 2012. Springer.

[Kiesl *et al.*, 2016] Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. Super-blocked clauses. In Nicola Olivetti and Ashish Tiwari, editors, *Proc. of the 8th Int. Joint Conference on Automated Reasoning* (*IJCAR 2016*), volume 9706 of *LNCS*, pages 45–61, Cham, 2016. Springer.

[Kullmann, 1999] Oliver Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 96-97:149–176, 1999.

[Manthey *et al.*, 2013] Norbert Manthey, Tobias Philipp, and Christoph Wernhard. Soundness of inprocessing in clause sharing SAT solvers. In Matti Järvisalo and Allen Van Gelder, editors, *Proc. of the 16th Int. Conference on Theory and Applications of Satisfiability Testing* (*SAT 2013*), volume 7962 of *LNCS*, pages 22–39, Heidelberg, 2013. Springer.