# Non-Clausal Redundancy Properties*
## (Extended Version)

Lee A. Barnett[0000−0003−1282−8596] and Armin Biere[0000−0001−7170−9242]

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

**Abstract.** State-of-the-art refutation systems for SAT are largely based on the derivation of clauses meeting some redundancy criteria, ensuring their addition to a formula does not alter its satisfiability. However, there are strong propositional reasoning techniques whose inferences are not easily expressed in such systems. This paper extends the redundancy framework beyond clauses to characterize redundancy for Boolean constraints in general. We show this characterization can be instantiated to develop efficiently checkable refutation systems using redundancy properties for Binary Decision Diagrams (BDDs). Using a form of reverse unit propagation over conjunctions of BDDs, these systems capture, for instance, Gaussian elimination reasoning over XOR constraints encoded in a formula, without the need for clausal translations or extension variables. Notably, these systems generalize those based on the strong Propagation Redundancy (PR) property, without an increase in complexity.

## 1 Introduction

The correctness and reliability of Boolean satisfiability (SAT) solvers is critical for many applications. For instance SAT solvers are used for verifying hardware and software systems (e.g. [18,27,43]), to search for solutions to open problems in mathematics (e.g. [37,45]), and as subroutines of other logical reasoning tools (e.g. [6,66]). Solvers should be able to provide solution certificates that are easily and externally checkable. For a satisfiable formula, any satisfying assignment is a suitable certificate and typically can be easily produced by a solver. For an unsatisfiable formula, a solver should be able to produce a refutation proof.

Modern SAT solvers primarily refute unsatisfiable formulas using clausal proof systems, such as the popular DRAT system [68] used by the annual SAT competition in recent years [4], or newer systems based on the surprisingly strong Propagation Redundancy (PR) property [32]. Clausal proof systems iteratively extend a formula, typically given in conjunctive normal form (CNF), by adding clauses that are redundant; that is, their addition to the formula does not affect whether it is satisfiable. Systems are distinguished by their underlying redundancy properties, restricted but efficiently-decidable forms of redundancy.

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

https://doi.org/10.35011/fmvtr.2021-2
Technical Report 21/2, April 2021, FMV Reports Series
Institute for Formal Models and Verification, Johannes Kepler University
Altenbergerstr. 69, 4040 Linz, Austria

Redundancy is a useful notion in SAT as it captures most inferences made by state-of-the-art solvers. This includes clauses implied by the current formula, such as the resolvent of two clauses or clauses learned during conflict-driven clause learning (CDCL) [7,50], as well as clauses which are not implied but derived nonetheless by certain preprocessing and inprocessing techniques [42], such as those based on blocked clauses [41,44,47]. Further, clausal proof systems based on properties like PR include short refutations for several hard families of formulas, such as those encoding the pigeonhole principle, that have no polynomial-length refutations in resolution [2] (see [15] for an overview). These redundancy properties, seen as inference systems, thus potentially offer significant improvements in efficiency, as the CDCL algorithm at the core of most solvers searches only for refutations in resolution [8]. While the recent satisfaction-driven clause learning (SDCL) paradigm has shown some initial success [34,36], it is still unclear how to design solving techniques which take full advantage of this potential.

Conversely, there are existing strong reasoning techniques which similarly exceed the abilities of CDCL alone, but are difficult to express using clausal proof systems. Important examples include procedures for reasoning over CNF formulas encoding pseudo-Boolean and cardinality constraints (see [57]), as well as Gaussian elimination (see [11,60,61,67]), which has been highlighted as a challenge for clausal proof systems [30]. Gaussian elimination, applied to sets of "exclusive-or" (XOR) constraints, is a crucial technique for many problems from cryptographic applications [61], and can efficiently solve, for example, Tseitin formulas hard for resolution [63,65]. This procedure, implemented by CryptoMiniSAT [61], Lingeling [9], and Coprocessor [49] for example, can be polynomially simulated by extended resolution, allowing inferences over new variables, and similar systems (see [55,59]). However due to the difficulty of such simulations they are not typically implemented. Instead solvers supporting these techniques simply prevent them from running when proof output is required, preferring less efficient techniques whose inferences can be more easily represented.

This paper extends the redundancy framework for clausal proof systems to include non-clausal constraints, such as XOR or cardinality constraints, presenting a characterization of redundancy for Boolean functions in general. We demonstrate a particular use of this characterization by instantiating it for functions represented by Binary Decision Diagrams [12], a powerful representation with a long history in SAT solving (e.g. [13,22,23,51,53]) and other areas of automated reasoning (e.g. [14,28,46,56]). We show the resulting refutation systems succinctly express Gaussian elimination while also generalizing existing clausal systems. Results using a prototype implementation confirm these systems allow compact and efficiently checkable refutations of CNF formulas that include embedded XOR constraints solvable by Gaussian elimination.

In the rest of the paper, Section 2 includes preliminaries and Section 3 presents the characterization of redundancy for Boolean functions. Section 4 introduces redundancy properties for BDDs, and Section 5 demonstrates their use for Gaussian elimination. Section 6 presents the results of our preliminary implementation, and Section 7 concludes.

## 2 Preliminaries

We assume a set of Boolean variables $V$ under a fixed order $\prec$ and use standard SAT terminology. The set of truth values is $B = \{0, 1\}$. An *assignment* is a function $\tau : V \to B$ and the set of assignments is $B^V$. A function $f : B^V \to B$ is *Boolean*. If $f(\tau) = 1$ for some $\tau \in B^V$ then $f$ is *satisfiable*, otherwise $f$ is *unsatisfiable*. Formulas express Boolean functions as usual, are assumed to be in conjunctive normal form, and are written using capital letters $F$ and $G$. A clause can be represented by its set of literals and a formula by its set of clauses.

A *partial assignment* is a non-contradictory set of literals $\sigma$; that is, if $l \in \sigma$ then $\neg l \notin \sigma$. The *application* of a partial assignment $\sigma$ to a clause $C$ is written $C|_\sigma$ and defined by: $C|_\sigma = \top$ if every $\tau \in B^V$ that satisfies $\bigwedge_{l \in \sigma} l$ also satisfies $C$, otherwise $C|_\tau = \{l \mid l \in C \text{ and } l, \neg l \notin \sigma\}$. For example, $(x_1 \vee x_2)|_{\{\neg x_1, x_2\}} = \top$, and $(x_1 \vee x_2)|_{\{\neg x_2, \neg x_3\}} = (x_1)$. Similarly the application of $\sigma$ to a formula $F$ is written $F|_\sigma$ and defined by: $F|_\sigma = \top$ if $C|_\sigma = \top$ for all $C \in F$, otherwise $F|_\sigma = \{C|_\sigma \mid C \in F \text{ and } C|_\sigma \neq \top\}$. *Unit propagation* is the iterated replacement of $F$ with $F|_{\{l\}}$ for each unit clause $(l) \in F$, until $F$ includes the empty clause $\bot$, or $F$ contains no unit clauses. A formula $F$ implies a clause $C$ by *reverse unit propagation* (RUP) if unit propagation on $F \wedge \neg C$ ends by producing $\bot$ [26].

For a formula $F$ and clause $C$, if $F$ and $F \wedge C$ are equisatisfiable (both satisfiable or both unsatisfiable) then $C$ is *redundant* with respect to $F$. Efficiently identifiable redundant clauses are at the foundation of many formula simplification techniques and refutation systems (for instance, see [31,32,36,42]). In general, deciding whether a clause is redundant is complete for the complement of the class DP [5], containing both NP and co-NP [54], so solvers and proof systems rely on polynomially-decidable *redundancy properties* for checking specific instances of redundancy. The following characterization of redundant clauses provides a common framework for formulating such properties.

**Theorem 1 (Heule, Kiesl, and Biere [35]).** *A clause $C \neq \bot$ is redundant with respect to a formula $F$ if and only if there is a partial assignment $\omega$ such that $C|_\omega = \top$ and $F|_\alpha \vDash F|_\omega$, for the partial assignment $\alpha = \{\neg l \mid l \in C\}$.*

The partial assignment $\omega$, usually called a *witness* for $C$, includes at least one of the literals occurring in $C$, while $\alpha$ is said to *block* the clause $C$. Redundancy properties can be defined by replacing $\vDash$ in the theorem above with efficiently-decidable relations $R$ such that $R \subseteq \vDash$. *Propagation redundancy* (PR) [32] replaces $\vDash$ with $\vdash_1$, where $F \vdash_1 G$ if and only if $F$ implies each $D \in G$ by RUP. The property PR gives rise to a refutation system, in which a refutation is a list of clauses $C_1, \ldots, C_n$ and witnesses $\omega_1, \ldots, \omega_n$ such that $C_k|_{\omega_k} = \top$ and $(F \bigwedge_{i=1}^{k-1} C_i)|_{\alpha_k} \vdash_1 (F \bigwedge_{i=1}^{k-1} C_i)|_{\omega_k}$ for all $1 \leq k \leq n$, and $F \bigwedge_{i=1}^{n} C_i \vdash_1 \bot$.

Most redundancy properties used in SAT solving can be understood as restricted forms of propagation redundancy. The RAT property [42] is equivalent to *literal propagation redundancy*, where the witness $\omega$ for any clause $C$ may differ from the associated $\alpha$ on only one literal; that is, $\omega = (\alpha \setminus \{\neg l\}) \cup \{l\}$ for some $l \in C$ [35]. The DRAT system [68] is based on RAT, with the added ability to remove clauses from the accumulated formula $F \bigwedge C_i$.

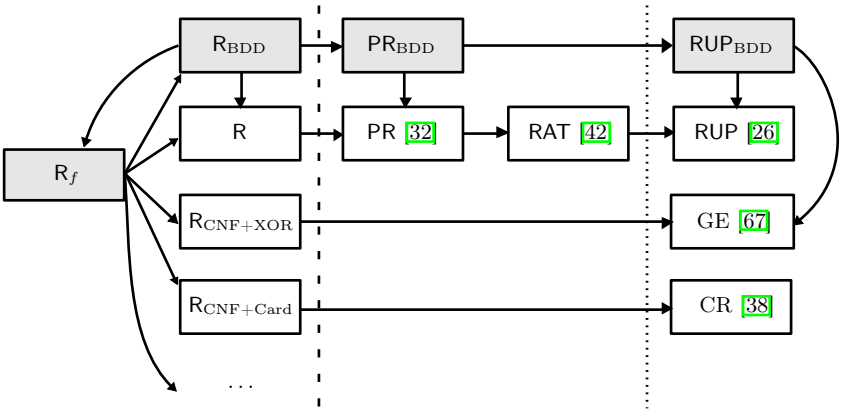



Fig. 1: Different notions of redundancy and their relationships. An arrow from $A$ to $B$ indicates $A$ generalizes $B$. Properties to the right of the thick dashed line are polynomially checkable; those to the right of the thin dotted line only derive logical consequences. Novel properties defined in this paper are grey.

## 3 Redundancy for Boolean Functions

Theorem 1 provides a foundation for clausal proof systems by characterizing redundant clauses in a convenient way. However, the restriction to clauses places limitations on these systems, making some forms of non-clausal reasoning difficult to express. For solvers aiming to construct refutations in these systems, this translates directly to restrictions on which solving techniques can be used.

We show this characterization can be broadened to include redundancy for non-clausal constraints, and can be used to define useful redundancy properties and refutation systems. The contributions of this paper are divided into three corresponding levels of generality. The top level, covered in the current section, is the direct extension of Theorem 1 from redundancy for clauses, written $\mathsf{R}$, to redundancy for Boolean functions, written $\mathsf{R}_f$. The middle level, the focus of Section 4, instantiates the resulting Theorem 2 to define the refutation systems $\mathsf{RUP}_{\mathrm{BDD}}$ and $\mathsf{PR}_{\mathrm{BDD}}$ based on redundancy for Binary Decision Diagrams. At the bottom level, these systems are shown to easily handle Gaussian elimination (GE) in Section 5, as well as some aspects of cardinality reasoning (CR). The relationships between these notions of redundancy are shown in Figure 1.

Each level of generality is individually important to this work. At the bottom level, the straightforward expression of Gaussian elimination by $\mathsf{RUP}_{\mathrm{BDD}}$ and $\mathsf{PR}_{\mathrm{BDD}}$ makes it more feasible for solvers to use this efficient technique with proof production, especially as these systems generalize their clausal analogs already in use. The results in Section 6 confirm the usefulness of $\mathsf{RUP}_{\mathrm{BDD}}$ for this purpose. At the middle level, we show the notion of redundancy instantiated

for BDDs in this way may be capable of other strong forms of reasoning as well. Finally, the top level provides a very general form of redundancy, independent of function representation. This may make possible the design of redundancy properties and refutation systems in contexts where the BDD representation of constraints is too large; for example, it is known that some pseudo-Boolean constraints can in general have exponential size BDD representations [1,40].

This section presents in Theorem 2 a characterization of redundancy for Boolean functions in general. One way of instantiating this characterization is demonstrated in Section 4 where the functions are represented by Binary Decision Diagrams; the resulting refutation systems are shown in Section 5 to easily express Gaussian elimination. However, the applicability of Theorem 2 is much broader, providing a foundation for redundancy-based refutation systems independent of the representation used.

Proofs of theoretical results not included in the text can be found in the appendix. We begin with the definition corresponding to the property $\mathsf{R}_f$.

**Definition 1.** *A Boolean function $g$ is* redundant *with respect to a Boolean function $f$ if the functions $f$ and $f \wedge g$ are both satisfiable, or both unsatisfiable.*

As we will see, extending Theorem 1 to the non-clausal case relies on the notion of a *Boolean transformation*, or just transformation: a function $\varphi : B^V \to B^V$, mapping assignments to assignments. Importantly, for a function $f$ and transformation $\varphi$, in fact $f \circ \varphi : B^V \to B$ is a function as well, where as usual $f \circ \varphi(\tau) = f(\varphi(\tau))$. For instance let $F = x_1 \wedge x_2$ and for all $\tau \in B^V$, the transformation $\varphi$ *flips* $x_1$, so that $\varphi(\tau)(x_1) = \neg\tau(x_1)$, and *ignores* $x_2$, that is, $\varphi(\tau)(x_2) = \tau(x_2)$. Then in fact $F \circ \varphi$ is expressed by the formula $\neg x_1 \wedge x_2$.

Composing a function with a transformation can be seen as a generalization of the application of a partial assignment to a formula or clause as defined in the previous section. Specifically, for a partial assignment $\sigma$ let $\hat{\sigma}$ refer to the following transformation: for any assignment $\tau$, the assignment $\hat{\sigma}(\tau)$ satisfies $\bigwedge_{l \in \sigma} l$, and $\hat{\sigma}$ ignores any $x \in V$ such that $x, \neg x \notin \sigma$. Then for any formula $F$ the formula $F|_\sigma$ expresses exactly the function $F \circ \hat{\sigma}$. In particular, if $\alpha$ is the partial assignment blocking a clause $C$ then notice $C \circ \hat{\alpha}(\tau) = 0$ for all $\tau$, but $\hat{\alpha}$ ignores variables not appearing in $C$; consequently $\hat{\alpha}(\tau) = \tau$ if $\tau$ already falsifies $C$. Generalizing this idea to transformations that block non-clausal constraints is more complicated. In particular, there may be multiple blocking transformations.

*Example 1.* Let $g$ be the function $g(\tau) = 1$ if and only if $\tau(a) \neq \tau(b)$ (i.e. $g$ is an XOR constraint). Transformations $\alpha_1$, $\alpha_2$ are shown in the table below.

| $\tau(a)$ | $\tau(b)$ | $g$ | $\alpha_1(\tau)(a)$ | $\alpha_1(\tau)(b)$ | $g \circ \alpha_1$ | $\alpha_2(\tau)(a)$ | $\alpha_2(\tau)(b)$ | $g \circ \alpha_2$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |

Both transformations ignore all $x \neq a, b$. Notice if $g(\tau) = 0$ then $\tau$ is unaffected by either transformation, and $g \circ \alpha_1(\tau) = g \circ \alpha_2(\tau) = 0$ for any assignment $\tau$.

However $\alpha_1$ and $\alpha_2$ are different, so that, for example, if $F = \neg a \wedge (b \vee c)$ and $\tau$ satisfies the literals $\neg a$, $b$, and $c$ then $F \circ \alpha_1(\tau) = 1$ but $F \circ \alpha_2(\tau) = 0$.

Motivated by this we define transformations blocking a function as follows.

**Definition 2.** *A transformation* $\alpha$ blocks *a function* $g$ *if* $g \circ \alpha$ *is unsatisfiable, and for any assignment* $\tau$ *if* $g(\tau) = 0$ *then* $\alpha(\tau) = \tau$.

Notice any $g$ not equal to the constant function 1 has blocking transformations; for example, by mapping every $\tau$ satisfying $g$ to a particular assignment falsifying it. Using this definition, the following theorem shows how the redundancy of a Boolean function $g$ with respect to another function $f$ can be demonstrated. This is a direct generalization of Theorem 1, using a transformation blocking $g$ in the place of the partial assignment blocking a clause, and a transformation $\omega$ such that $g \circ \omega$ is the constant function 1 in place of the witnessing assignment.

**Theorem 2.** *Let* $f$ *be a function and* $g$ *a non-constant function. Then* $g$ *is redundant with respect to* $f$ *if and only if there exist transformations* $\alpha$ *and* $\omega$ *such that* $\alpha$ *blocks* $g$ *and* $g \circ \omega$ *is the constant function 1, and further* $f \circ \alpha \vDash f \circ \omega$.

*Proof.* ($\Rightarrow$) Suppose $g$ is redundant with respect to $f$ and let $\alpha$ be any transformation blocking $g$. If $f$ is unsatisfiable then $f \circ \alpha$ is as well, so that $f \circ \alpha \vDash f \circ \omega$ holds for any $\omega$. Thus we can take as $\omega$ the transformation $\omega(\tau) = \tau^*$ for all $\tau \in B^V$, where $\tau^*$ is some assignment satisfying $g$. If instead $f$ is satisfiable, by redundancy so is $f \wedge g$. Here we can take as $\omega$ the transformation $\omega(\tau) = \tau^*$ for all $\tau \in B^V$, where $\tau^*$ is some assignment satisfying $f \wedge g$. Then both $f \circ \omega$ and $g \circ \omega$ are the constant function 1, so that $f \circ \alpha \vDash f \circ \omega$ holds in this case as well.

($\Leftarrow$) Suppose $\alpha, \omega$ meet the criteria stated in the theorem. We show that $g$ is redundant by demonstrating that if $f$ is satisfiable, then so is $f \wedge g$. Suppose $\tau$ is an assignment satisfying $f$. If also $g(\tau) = 1$, then of course $\tau$ satisfies $f \wedge g$. If instead $g(\tau) = 0$, then $\alpha(\tau) = \tau$ as $\alpha$ blocks the function $g$. Thus $f \circ \alpha\,(\tau) = f(\alpha(\tau)) = f(\tau) = 1$. As $f \circ \alpha \vDash f \circ \omega$, this means $f(\omega(\tau)) = 1$. As $g \circ \omega$ is the constant function 1 then $g(\omega(\tau)) = 1$, so $\omega(\tau)$ satisfies $f \wedge g$. □

The clausal characterization in Theorem 1 shows that the redundancy of a clause can be evidenced by providing a witnessing assignment and demonstrating that an implication holds, providing a foundation for refutations based on the iterative conjunction of clauses. Theorem 2 above shows that the redundancy of a function in general can be seen in the same way by providing transformations $\alpha$ and $\omega$. Consequently this suggests how to construct refutations based on the iterative conjunction of Boolean functions.

**Definition 3.** *A sequence* $\sigma = (g_1, \alpha_1, \omega_1), \ldots, (g_n, \alpha_n, \omega_n)$ *is a* redundancy sequence *for a Boolean function* $f$ *if:*

1. $\alpha_k$ *blocks* $g_k$ *and* $g_k \circ \omega_k$ *is the constant function 1, for all* $1 \leq k \leq n$,
2. $(f \wedge \bigwedge_{i=1}^{k-1} g_i) \circ \alpha_k \vDash (f \wedge \bigwedge_{i=1}^{k-1} g_i) \circ \omega_k$, *for all* $1 \leq k \leq n$.

As for clausal redundancy, refutations are intuitively based on the following: if $g_1$ is redundant with respect to $f$, and $g_2$ is redundant with respect to $f \wedge g_1$, then $f$ and $f \wedge g_1 \wedge g_2$ are equisatisfiable; that is, $g_1 \wedge g_2$ is redundant with respect to $f$. The following holds as a direct consequence.

**Proposition 1.** *Let $f$ be a Boolean function. If $(g_1, \alpha_1, \omega_1), \ldots, (g_n, \alpha_n, \omega_n)$ is a redundancy sequence for $f$, and $f \wedge \bigwedge_{i=1}^{n} g_i$ is unsatisfiable, then so is $f$.*

This shows, abstractly, how redundant Boolean functions can be used as a basis for refutations in the same way as redundant clauses. To define practical, and polynomially-checkable, refutation systems based on non-clausal redundancy in this way, we focus on a representation of Boolean functions that can be used within the framework described above. Specifically, we consider sets of BDDs in conjunction, just as formulas are sets of clauses in conjunction. Clauses are easily expressed by BDDs, and thus this representation easily expresses (CNF) formulas; this is necessary as we are typically interested in proving the unsatisfiability not of functions in general, but of (CNF) formulas. It is important to notice this is only a particular instantiation of Theorem 2, and that other representations of Boolean functions may give rise to useful and efficient systems as well.

BDDs [3,12,48] are compact expressions of Boolean functions in the form of rooted, directed, acyclic graphs consisting of *decision nodes*, each labeled by a variable $x \in V$ and having two children, and two *terminal nodes*, labeled by 0 and 1. The BDD for a function $f : B^V \to B$ is based on its *Shannon expansion*,

$$f \;=\; (\neg x \wedge f \circ \hat{\sigma}_0) \vee (x \wedge f \circ \hat{\sigma}_1)$$

where $\sigma_0 = \{\neg x\}$ and $\sigma_1 = \{x\}$, for $x \in V$. As is common we assume BDDs are *ordered* and *reduced*: if a node with variable label $x$ precedes a node with label $y$ in the graph then $x \prec y$, and the graph has no distinct, isomorphic subgraphs. Representation this way is canonical up to variable order, so that no two distinct BDDs with the same variable order represent the same Boolean function [12].

Our use of BDDs for representing non-clausal redundancy relies on the concept of *cofactors* as developed in BDD literature. The functions $f \circ \hat{\sigma}_0$ and $f \circ \hat{\sigma}_1$ are called *literal cofactors* of $f$ by $\neg x$ and $x$, respectively, and are usually written $f|_{\neg x}$ and $f|_x$. The cofactor of $f$ by a conjunction of literals $c = l_1 \wedge \cdots \wedge l_n$ can be defined similarly, so that $f|_c = f \circ \hat{\sigma}_c$, for the partial assignment $\sigma_c = \{l_1, \ldots, l_n\}$. This notation is the same as for the application of a partial assignment to a clause or formula from Section 2, as the notions coincide. More precisely, if a formula $F$ and BDD $f$ express the same function, so do the formula $F|_{\sigma_c}$ and BDD $f|_c$.

More broadly, for BDDs $f$ and $g$, a *generalized cofactor* of $f$ by $g$ is a BDD $h$ such that $f \wedge g = h \wedge g$; that is, $f$ and $h$ agree on all assignments satisfying $g$. This leaves unspecified what value $h(\tau)$ should take when $g(\tau) = 0$, and various different BDD operations have been developed for constructing generalized cofactors [19,20,21]. The *constrain* operation [20] produces for $f$ and $g$, with $g$ not equal to the always false 0 BDD, a generalized cofactor which can be seen

as the composition $f \circ \pi_g$, where $\pi_g$ is the transformation [62]:

$$\pi_g(\tau) = \begin{cases} \tau & \text{if } g(\tau) = 1 \\ \underset{\{\tau' \mid g(\tau')=1\}}{\arg\min} \; d(\tau, \tau') & \text{otherwise.} \end{cases}$$

The function $d$ is defined as follows: $d(\tau, \tau') = \sum_{i=1}^{n} |\tau(x_i) - \tau'(x_i)| \cdot 2^{n-i}$, where $V = \{x_1, \ldots, x_n\}$ with $x_1 \prec \cdots \prec x_n$. Intuitively, $d$ is a measure of distance between two assignments based on the variables on which they disagree, weighted by their position in the variable order. It is important to notice then that the transformation $\pi_g$ and the resulting $f \circ \pi_g$ depend on the variable order, and may differ for distinct orders. For a conjunction of literals $c$, though, $f \circ \pi_c = f|_c$ regardless of the order, so that $f|_g$ refers to $f \circ \pi_g$ in general.

As the transformation $\pi_g$ maps an assignment falsifying the function $g$ to the nearest assignment (with respect to $d$) satisfying it, a transformation that blocks the function $g$ can surely be obtained as follows.

**Lemma 1.** *If $g$ is not equal to the constant function 1 then $\pi_{\neg g}$ blocks $g$.*

This form of generalized cofactor, as computed by the constrain operation, is well suited for use in redundancy-based reasoning as described above, as the transformation $\pi_{\neg g}$ depends only on $g$. As a consequence, for BDDs $f_1$ and $f_2$ in fact $(f_1 \wedge f_2)|_{\neg g} \equiv f_1|_{\neg g} \wedge f_2|_{\neg g}$; that is, the BDD $(f_1 \wedge f_2)|_{\neg g}$ expresses the same function as the BDD for the conjunction $f_1|_{\neg g} \wedge f_2|_{\neg g}$. Thus given a set of BDDs $f_1, \ldots, f_n$ we can represent $(f_1 \wedge \cdots \wedge f_n)|_{\neg g}$ simply by the set of cofactors $f_i|_{\neg g}$ and without constructing the BDD for the conjunction $f_1 \wedge \cdots \wedge f_n$, which is NP-hard in general. In particular, given a formula $F = C_1 \wedge \cdots \wedge C_n$ and a Boolean constraint $g$, the function $F|_{\neg g}$ can be represented simply by applying the constrain operation to each of the BDDs representing $C_i$. Therefore, from Theorem 2 we can characterize redundancy for conjunctions of BDDs, written $\mathsf{R}_{\mathrm{BDD}}$, as follows.

**Proposition 2.** *Suppose $f_1, \ldots, f_n$ are BDDs and $g$ is a non-constant BDD. If there is a partial assignment $\{l_1, \ldots, l_k\}$ such that for $\omega = \bigwedge_{i=1}^{k} l_i$,*

$$f_1|_{\neg g} \wedge \cdots \wedge f_n|_{\neg g} \vDash f_1|_\omega \wedge \cdots \wedge f_n|_\omega$$

*and $g|_\omega = 1$ then $g$ is redundant with respect to $f_1 \wedge \ldots \wedge f_n$.*

## 4 BDD Redundancy Properties

The previous section provided a characterization of redundancy for Boolean functions, and showed how this could be instantiated for BDDs. In this section we develop polynomially-checkable properties for showing that a BDD is redundant with respect to a conjunction of BDDs, and describe their use in refutation systems for proving the unsatisfiability of formulas.

```
   UnitProp(f_1, ..., f_n)
1    repeat
2      if f_i = 0 or f_i = ¬f_j for some 1 ≤ i, j ≤ n then
3        return "conflict"
4      if U(f_i) ≠ ∅ for some 1 ≤ i ≤ n then
5        f_j := f_j|_{⋀ U(f_i)} for all 1 ≤ j ≤ n
6    until no update to f_1, ..., f_n
```

Fig. 2: A procedure for unit propagation over a set of BDDs

As Theorem 1 is used for defining clausal redundancy properties, Proposition 2 gives rise to BDD redundancy properties by replacing $\vDash$ with polynomially-decidable relations. Similar to the use of the unit propagation procedure by the clausal properties RUP and PR, we describe a unit propagation procedure for use with a set of BDDs and derive analogous properties $\mathsf{RUP}_{\mathrm{BDD}}$ and $\mathsf{PR}_{\mathrm{BDD}}$.

For a BDD $f$, the Shannon expansion shows that if $f|_{\neg l} = 0$ (i.e. $f|_{\neg l}$ is the always false 0 BDD) for some literal $l$, then $f = l \wedge f_l$, and therefore $f \vDash l$. Then the *units implied by* $f$, written $U(f)$, can be defined as follows.

**Definition 4.** *$U(f) = \{l \mid \mathrm{var}(l) \in V$ and $f|_{\neg l} = 0\}$, for $f : B^V \to B$.*

As $f|_{\neg l}$ can be computed in $O(|f|)$, where $|f|$ is the number of nodes in the BDD for $f$ [58], then $U(f)$ can certainly be computed in $O(|V| \cdot |f|) \subseteq O(|f|^2)$, though this can be reduced to $O(|f|)$. We write $\bigwedge U(f)$ to mean $\bigwedge_{l \in U(f)} l$.

Figure 2 provides a sketch of the unit propagation procedure. Whenever $U(f)$ is non-empty for some $f$ in a set of BDDs, each BDD in the set can be replaced with its cofactor by $\bigwedge U(f)$. This approach to unit propagation is largely similar to that of Olivo and Emerson [52], except we consider two conflict situations: if some BDD becomes 0, or if two BDDs are the negations of each other.

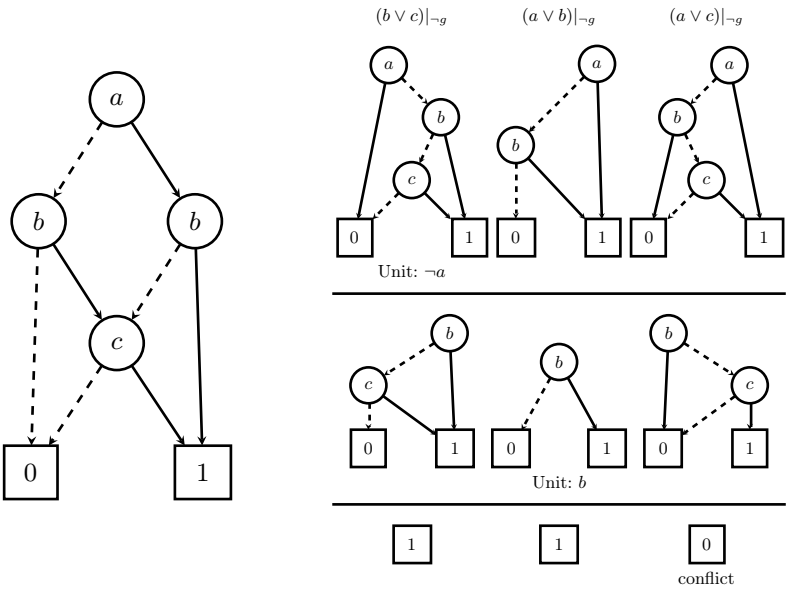
For $N = |f_1| + \cdots + |f_n|$ the procedure $\mathsf{UnitProp}(f_1, \ldots, f_n)$ can be performed in time $O(N^2)$. In line 5, if $f_j$ and $\bigwedge U(f_i)$ share no variables, then $f_j = f_j|_{\bigwedge U(f_i)}$, otherwise the BDD for $f_j|_{\bigwedge U(f_i)}$ can be constructed in time $O(|f_j|)$ and further $|f_j|_{\bigwedge U(f_i)}| < |f_j|$. This procedure is correct: "conflict" is only returned when $\bigwedge_{i=1}^{n} f_i$ is unsatisfiable (see the appendix for the proof).

**Proposition 3.** *If $\mathsf{UnitProp}(f_1, \ldots, f_n)$ returns "conflict" then $f_1 \wedge \cdots \wedge f_n \equiv 0$.*

UnitProp generalizes the usual unit propagation procedure on a formula: if $C$ is a clause, then $U(C) \neq \emptyset$ implies $C$ is a unit clause and $\bigwedge_{l \in U(C)} l = C$. We extend the relation $\vdash_1$ and the definition of RUP accordingly.

**Definition 5.** *Let $f_1, \ldots, f_n$ and $g \neq 0$ be BDDs. Then $f_1 \wedge \cdots \wedge f_n$ implies $g$ by $\mathsf{RUP}_{\mathrm{BDD}}$ if $\mathsf{UnitProp}(f_1|_{\neg g}, \ldots, f_n|_{\neg g})$ returns "conflict."*

*Example 2.* Let $F = \{C_1 = b \vee c, C_2 = a \vee b, C_3 = a \vee c\}$, and assume $a \prec b \prec c$. Consider $g$ as shown in Figure 3, expressing the cardinality constraint $g(\tau) = 1$ if and only if $\tau$ satisfies at least two $a$, $b$, $c$; also written $\{a, b, c\} \geq 2$. Figure 3




(a) The constraint $g$ (b) $\mathsf{UnitProp}((b \vee c)|_{\neg g}, (a \vee b)|_{\neg g}, (a \vee c)|_{\neg g})$

Fig. 3: Example derivation of a constraint $g$, shown in (a), using $\mathsf{RUP}_{\mathrm{BDD}}$. In (b), the top line shows the BDDs for each of the clauses $(b \vee c), (a \vee c), (a \vee b)$ after cofactoring by $g$. The second line shows each of these BDDs after cofactoring by the unit $\neg a \in U((b \vee c)|_{\neg g})$. Here, the middle BDD becomes simply the unit $b$, and the third line shows each BDD cofactored by the unit $b$. In this line, the third BDD has become 0, so a conflict is returned.

shows the updates made throughout $\mathsf{UnitProp}(C_1|_{\neg g}, C_2|_{\neg g}, C_3|_{\neg g})$. Notice that $U(C_1|_{\neg g}) = \{\neg a\}$, and $U((C_2|_{\neg g})|_{\neg a}) = \{b\}$. Then $C_3|_{\neg g}$ after cofactoring by $\neg a$ and $b$ becomes the constant BDD 0, so the procedure returns "conflict." As a result, $F$ implies the BDD $g$ by $\mathsf{RUP}_{\mathrm{BDD}}$.

We show that $\mathsf{RUP}_{\mathrm{BDD}}$ is a redundancy property. Given BDDs $f_1, \ldots, f_n, g$, checking whether $g$ is implied by $\mathsf{RUP}_{\mathrm{BDD}}$ primarily consists of the $\mathsf{UnitProp}$ procedure, though each $f_i|_{\neg g}$ must first be constructed, which can be done in time $O(|f_i| \cdot |g|)$ [20]. The size of this BDD may in some cases be larger than the size of $f_i$, though it is typically smaller [20,62] and at worst $|f_i|_{\neg g}| \leq |f_i| \cdot |g|$. Consequently it can be decided in time $O(|g|^2 \cdot N^2)$ whether $g$ is implied by $\mathsf{RUP}_{\mathrm{BDD}}$. Finally if $g$ is implied by $\mathsf{RUP}_{\mathrm{BDD}}$ then it is redundant with respect to $f_1 \wedge \cdots \wedge f_n$; in fact, it is a logical consequence (proof available in the appendix).

**Proposition 4.** *If* $f_1 \wedge \cdots \wedge f_n \vdash_1 g$, *then* $f_1 \wedge \cdots \wedge f_n \vDash g$.

From $\mathsf{RUP}_{\mathrm{BDD}}$ the property $\mathsf{PR}$ can be directly generalized to this setting as well. Specifically, we define the redundancy property $\mathsf{PR}_{\mathrm{BDD}}$ as follows.

**Definition 6.** *Suppose $f_1, \ldots, f_n$ are BDDs and $g$ is a non-constant BDD. Then $g$ is $\mathsf{PR}_{\mathrm{BDD}}$ with respect to $\bigwedge_{i=1}^{n} f_i$ if there is partial assignment $\{l_1, \ldots, l_k\}$ such that $g|_\omega = 1$ and $\bigwedge_{i=1}^{n} f_i|_{\neg g} \vdash_1 f_j|_\omega$ for all $1 \leq j \leq n$, where $\omega = \bigwedge_{i=1}^{k} l_i$.*

Proposition 2 shows if $g$ is $\mathsf{PR}_{\mathrm{BDD}}$ with respect to $f = f_1 \wedge \cdots \wedge f_n$ then $g$ is redundant with respect to $f$, thus $\mathsf{PR}_{\mathrm{BDD}}$ is a redundancy property.

Notice these properties and derivations directly generalize their clausal equivalents; for example, if $C$ is $\mathsf{PR}$ with respect to a formula $F$, then (the BDD expressing) $C$ is $\mathsf{PR}_{\mathrm{BDD}}$ with respect to (the set of BDDs expressing) $F$. Deciding whether a clause $C$ is $\mathsf{PR}$ with respect to a formula $F$ is NP-complete [36]. As $\mathsf{PR}_{\mathrm{BDD}}$ generalizes $\mathsf{PR}$, then $\mathsf{PR}_{\mathrm{BDD}}$ is NP-hard as well. Further, checking whether $g$ is $\mathsf{PR}_{\mathrm{BDD}}$ with respect to $f_1 \wedge \cdots \wedge f_n$ by some candidate $\omega$ can be done polynomially as argued above, thus the following holds.

**Proposition 5.** *Deciding whether $g$ is $\mathsf{PR}_{\mathrm{BDD}}$ with respect to $f_1 \wedge \cdots \wedge f_n$, given the BDDs $g, f_1, \ldots, f_n$, is NP-complete.*

In other words, the decision problems for $\mathsf{PR}$ and $\mathsf{PR}_{\mathrm{BDD}}$ are of equal complexity.

The properties $\mathsf{RUP}_{\mathrm{BDD}}$ and $\mathsf{PR}_{\mathrm{BDD}}$ as defined in this section can be used to show that a BDD can be added to a set of BDDs in a satisfiability-preserving way. Of course, any clause has a straightforward and simple representation as a BDD, so that a formula can be easily represented this way as a set of BDDs. As a result $\mathsf{RUP}_{\mathrm{BDD}}$ and $\mathsf{PR}_{\mathrm{BDD}}$ can be used as systems for refuting unsatisfiable formulas. In the following, we identify a clause with its representation as a BDD, and a formula with its representation as a set of such BDDs.

To simplify the presentation of derivations based on $\mathsf{RUP}_{\mathrm{BDD}}$ and $\mathsf{PR}_{\mathrm{BDD}}$ we introduce an additional redundancy property, allowing derivations to include steps to directly derive certain BDDs *path-wise* in the following way.

**Definition 7.** *$f_1 \wedge \cdots \wedge f_n$ implies $g$ by $\mathsf{RUP}_{\mathrm{path}}$ if (1) $f_1 \wedge \cdots \wedge f_n \vdash_1 \neg c$ for every $c = l_1 \wedge \cdots \wedge l_m$ such that $l_1, \ldots, l_m$ is a path from the root of $g$ to the 0 terminal, and (2) $|g| \leq \log_2(|f_1| + \cdots + |f_n|)$.*

If $f_1 \wedge \cdots \wedge f_n$ implies $g$ by $\mathsf{RUP}_{\mathrm{path}}$ then it is a logical consequence of $f_1 \wedge \cdots \wedge f_n$, as this checks that no assignment satisfies both $\neg g$ and $f_1 \wedge \cdots \wedge f_n$. The number of paths in a BDD $g$ can however be exponential in $|g|$, as in the BDD for an XOR constraint, so the second condition ensures $\mathsf{RUP}_{\mathrm{path}}$ is polynomially-checkable.

The property $\mathsf{RUP}_{\mathrm{path}}$ is primarily useful as it allows the derivation of a BDD $g$ whose representation as a set of clauses is included in $\{f_1, \ldots, f_n\}$: if $c$ corresponds to a path to 0 in $g$, the clause $\neg c$ is included in the direct clausal translation of $g$. In this context, the restrictive condition (2) in Definition 7 can in fact be removed, since the number of paths in $g$ is then at most $n$.

**Definition 8.** *A sequence of BDDs $g_1, \ldots, g_n$ is a $\mathsf{RUP}_{\mathrm{BDD}}$ derivation from a formula $F$ if $F \wedge \bigwedge_{i=1}^{k-1} g_i$ implies $g_k$ by $\mathsf{RUP}_{\mathrm{BDD}}$, or by $\mathsf{RUP}_{\mathrm{path}}$, for all $1 \leq k \leq n$. A sequence of BDD and assignment pairs $(g_1, \omega_1), \ldots, (g_n, \omega_n)$ is*

*a* $\mathsf{PR}_{\mathrm{BDD}}$ derivation *from a formula* $F$ *if* $F \wedge \bigwedge_{i=1}^{k-1} g_i$ *implies* $g_k$ *by* $\mathsf{RUP}_{\mathrm{path}}$*, or* $\omega_k$ *is a* $\mathsf{PR}_{\mathrm{BDD}}$*-witness for* $g_k$ *with respect to* $F \wedge \bigwedge_{i=1}^{k-1} g_i$*, for all* $1 \leq k \leq n$*.*

As $\mathsf{RUP}_{\mathrm{BDD}}$, $\mathsf{RUP}_{\mathrm{path}}$, and $\mathsf{PR}_{\mathrm{BDD}}$ are redundancy properties, any $\mathsf{RUP}_{\mathrm{BDD}}$ or $\mathsf{PR}_{\mathrm{BDD}}$ derivation corresponds to a redundancy sequence of the same length.

*Example 3.* Consider the formula $F = \{a \vee b, a \vee c, b \vee c, a \vee d, b \vee d, c \vee d\}$ and let $g$ be the BDD such that $g(\tau) = 1$ if and only if $\tau$ satisfies at least 3 of $a, b, c, d$; that is, $g$ is the cardinality constraint $\{a, b, c, d\} \geq 3$. As seen in Example 2, the constraint $g_1 = \{a, b, c\} \geq 2$ is $\mathsf{RUP}_{\mathrm{BDD}}$ with respect to $F$; similarly so are the constraints, $g_2 = \{a, c, d\} \geq 2$, and $g_3 = \{b, c, d\} \geq 2$. Now, $\neg a \in U(g_3|_{\neg g})$: for any $\tau$ the assignment $\pi_{\neg g}(\tau)$ satisfies at most 2 of $a, b, c, d$, and if $a$ is one of them then $\pi_{\neg g}(\tau)$ surely falsifies $g_3$. As a result, $(g_3|_{\neg g})|_a = 0$. In a similar way $\neg b \in U(g_2|_{\neg g})$. Since $g_1|_{\neg g}$ cofactored by the units $\neg a$ and $\neg b$ is falsified, then $\mathsf{UnitProp}(g_1|_{\neg g}, g_2|_{\neg g}, g_3|_{\neg g})$ returns "conflict." Consequently $g$ is $\mathsf{RUP}_{\mathrm{BDD}}$ with respect to $F \wedge g_1 \wedge g_2 \wedge g_3$, and $g_1, g_2, g_3, g$ is a $\mathsf{RUP}_{\mathrm{BDD}}$ derivation from $F$.

This example can be generalized to show that $\mathsf{RUP}_{\mathrm{BDD}}$ is capable of expressing an inference rule for cardinality constraints called the *diagonal sum* [39]. For $L = \{l_1, \ldots, l_n\}$ let $L_i = L \setminus \{l_i\}$; the diagonal sum derives $L \geq k + 1$ from the set of all $n$ constraints $L_i \geq k$.

While the properties and refutation systems $\mathsf{RUP}_{\mathrm{BDD}}$ and $\mathsf{PR}_{\mathrm{BDD}}$ easily extend their clausal counterparts, it is important to notice that redundancy-based systems using BDDs can be defined in other ways. For instance, say $\bigwedge_{i=1}^{n} f_i$ implies $g$ by $\mathsf{IMP}_{\mathrm{pair}}$ if $f_i|_{\neg g} \wedge f_j|_{\neg g} = 0$ for some $i, j$. Then $\mathsf{IMP}_{\mathrm{pair}}$ is polynomially checkable, computing the conjunction for each pair $i, j$. Moreover, it is clear that $f_1 \wedge f_2 \vDash g$ if and only if $f_1 \wedge f_2$ implies $g$ by $\mathsf{IMP}_{\mathrm{pair}}$. As many logical inference rules have this form, it is possible that systems based on $\mathsf{IMP}_{\mathrm{pair}}$ are very strong.

## 5 Gaussian Elimination

Next, we show how the Gaussian elimination technique for simplifying XOR constraints embedded in a formula is captured by the redundancy properties defined in the previous section. Specifically, if an XOR constraint $X$ is derivable from a formula $F$ by Gaussian elimination, we show there is a $\mathsf{RUP}_{\mathrm{BDD}}$ derivation from $F$ including the BDD expressing $X$ with only a linear size increase.

An *XOR clause* $[x_1, \ldots, x_n]^p$ expresses the function $f : B^V \to B$, where $V = \{x_1, \ldots, x_n\}$ and $p$ is 0 or 1, such that $f(\tau) = 1$ if and only if the number of $x_i \in V$ satisfied by $\tau$ is equal modulo 2 to $p$. In other words, $p$ expresses the parity of the positive literals $x_i$ an assignment must satisfy in order to satisfy the XOR clause. As $[x, y, y]^p$ and $[x]^p$ express the same function, we assume no variable occurs more than once in an XOR clause. Notice that $[]^0$ expresses the constant function 1, while $[]^1$ expresses 0.

The Gaussian elimination procedure begins by detecting XOR clauses encoded in a formula $F$. The *direct encoding* $\mathcal{D}(X)$ of $X = [x_1, \ldots, x_n]^p$ is the collection of clauses of the form $C = \{l_1, \ldots, l_n\}$, where each $l_i$ is either $x_i$ or

$\neg x_i$ and the number of negated literals in each $C$ is not equal modulo 2 to $p$ The formula $\mathcal{D}(X)$ expresses the same function as $X$, containing the clauses preventing each assignment over the variables in $X$ not satisfying $X$. As a result, $\mathcal{D}(X)$ implies the BDD expressing $X$ by $\mathsf{RUP}_{\mathrm{path}}$ (see appendix for proof).

**Lemma 2.** *$\mathcal{D}(X)$ implies $X$ by $\mathsf{RUP}_{\mathrm{path}}$, for $X = [x_1, \ldots, x_n]^p$.*

Similar to the approach of Philipp and Rebola-Pardo [55], we represent Gaussian elimination steps by deriving the addition $X \oplus Y$ of XOR clauses $X = [x_1, \ldots, x_m, z_1, \ldots, z_r]^p$ and $Y = [y_1, \ldots, y_n, z_1, \ldots, z_r]^q$, given by:

$$X \oplus Y = [x_1, \ldots, x_m, y_1, \ldots, y_n]^{p \oplus q}.$$

The following lemma shows that $X \oplus Y$ is $\mathsf{RUP}_{\mathrm{BDD}}$ with respect to $X \wedge Y$; that is, if a $\mathsf{RUP}_{\mathrm{BDD}}$ derivation includes $X$ and $Y$ then $X \oplus Y$ can be derived as well. This is a result of the following observation: while the precise cofactors of $X$ and $Y$ by $\neg(X \oplus Y)$ depend on the variable order $\prec$, they are the negations of one another (proof is included in the appendix).

**Lemma 3.** *Let $v$ be the $\prec$-greatest variable in occurring in exactly one of $X$ and $Y$, and assume $v$ occurs in $Y$. Then $X|_{\neg(X \oplus Y)} = X$, and $Y|_{\neg(X \oplus Y)} = \neg X$.*

The above lemma shows that the procedure $\mathsf{UnitProp}(X|_{\neg X \oplus Y}, Y|_{\neg X \oplus Y})$ returns "conflict" immediately, and as a result $X \oplus Y$ is $\mathsf{RUP}_{\mathrm{BDD}}$ with respect to $f_1 \wedge \cdots \wedge f_n \wedge X \wedge Y$ for any set of BDDs $f_1, \ldots, f_n$.

Define a Gaussian elimination derivation $\Pi$ from a formula $F$ as a sequence of XOR clauses $\Pi = X_1, \ldots, X_N$, such that for all $1 \leq i \leq N$, either $X_i = X_j \oplus X_k$ for $j, k < i$, or $\mathcal{D}(X_i) \subseteq F$. The size of the derivation is $|\Pi| = \sum_{i=1}^{N} s_i$, where $s_i$ is the number of variables occurring in $X_i$. We show that $\Pi$ corresponds to a $\mathsf{RUP}_{\mathrm{BDD}}$ derivation with only a linear size increase. This size increase is a result of the fact that the BDD expressing an XOR clause $X = [x_1, \ldots, x_n]^p$ has size $2n + 1$ (proof of the following theorem is available in the appendix).

**Theorem 3.** *Suppose $\Pi = X_1, \ldots, X_N$ is a Gaussian elimination derivation from a formula $F$. Then there is a $\mathsf{RUP}_{\mathrm{BDD}}$ derivation from $F$ with size $O(|\Pi|)$.*

A consequence of this theorem is that $\mathsf{RUP}_{\mathrm{BDD}}$ includes short refutations for formulas whose unsatisfiability can be shown by Gaussian elimination. More precisely, suppose a formula $F$ includes the direct representations of an unsatisfiable collection of XOR clauses. Then there is a polynomial-length Gaussian elimination derivation of the unsatisfiable XOR clause $[]^1$ from $F$ [61], and by Theorem 3, a polynomial-length $\mathsf{RUP}_{\mathrm{BDD}}$ derivation of the unsatisfiable BDD 0.

Notably, $\mathsf{RUP}_{\mathrm{BDD}}$ then includes short refutations of, for example, the Tseitin formulas, for which no polynomial-length refutations exist in the resolution system [63,65]. This limitation of resolution holds as well for the clausal RUP system, without the ability to introduce new variables, as it can be polynomially simulated by resolution [8,24]. As the translation into $\mathsf{RUP}_{\mathrm{BDD}}$ used to prove Theorem 3 introduces no new variables, this demonstrates the strength of $\mathsf{RUP}_{\mathrm{BDD}}$ compared to resolution and its clausal analog RUP.

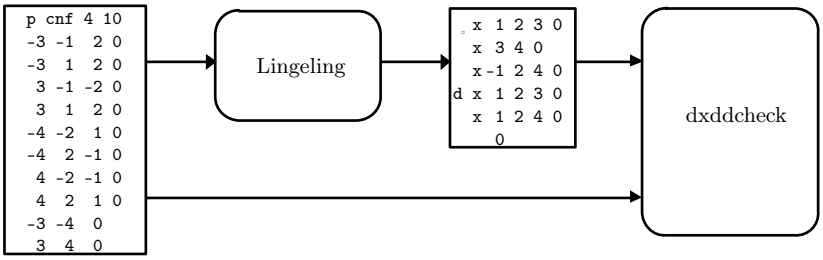



Fig. 4: Usage of the tool dxddcheck, showing an example formula and refutation.

## 6 Results

To begin to assess the practical usefulness of the systems introduced in Section 4 we have implemented in Python a prototype of a tool called dxddcheck[1] for checking refutations in a subset of $\mathsf{RUP}_{\mathrm{BDD}}$. In particular we focus on the result of Section 5, that Gaussian elimination is succinctly captured by $\mathsf{RUP}_{\mathrm{BDD}}$.

We ran the SAT solver Lingeling (version `bcp`) on a collection of crafted unsatisfiable formulas, all of which can be solved using Gaussian elimination. From Lingeling output we extract a list of XOR clause additions and deletions, ending with the addition of the empty clause, as shown in Figure 4. This list is passed directly to dxddcheck, which carries it out as a $\mathsf{DRUP}_{\mathrm{BDD}}$ refutation; that is, a $\mathsf{RUP}_{\mathrm{BDD}}$ refutation also allowing steps which remove or "delete" BDDs from the set. These deletion steps can be removed without affecting the correctness of the refutation, though their inclusion can decrease the time required for checking it, as is the case with DRUP and RUP.

| Formula | number of variables | number of clauses | solving time (s) | proof lines | proof size (KB) | checking time (s) |
|---|---|---|---|---|---|---|
| `rpar_50` | 148 | 394 | 0.1 | 297 | 7 | 0.34 |
| `rpar_100` | 298 | 794 | 0.1 | 597 | 15 | 1.35 |
| `rpar_200` | 598 | 1594 | 0.2 | 1197 | 35 | 6.67 |
| `mchess_19` | 680 | 2291 | 0.0 | 1077 | 41 | 4.07 |
| `mchess_21` | 836 | 2827 | 0.1 | 1317 | 50 | 5.09 |
| `mchess_23` | 1008 | 3419 | 0.1 | 1581 | 63 | 6.42 |
| `urquhart-s5-b2` | 107 | 742 | 0.0 | 150 | 7 | 0.95 |
| `urquhart-s5-b3` | 121 | 1116 | 0.1 | 150 | 9 | 1.64 |
| `urquhart-s5-b4` | 114 | 888 | 0.0 | 150 | 8 | 1.20 |

For these experiments we used a 1.8 GHz Intel Core i5 CPU with 8 GB of memory. The table shows the time Lingeling took to solve each formula, the number of lines in the constructed proof and its size, and the time dxddcheck took to construct and check the associated $\mathsf{DRUP}_{\mathrm{BDD}}$ proof. These benchmarks

[1] Source code is available under the MIT license at `http://fmv.jku.at/dxddcheck` along with the benchmarks used and our experimental data.

are well-known challenging examples in the contexts of XOR reasoning and proof production. The `rpar_`$n$ formulas are compact, permuted encodings of two contradictory parity constraints on $n$ variables, described by Chew and Heule [17]. The `mchess_`$n$ formulas are encodings of the mutilated $n \times n$-chessboard problem, as studied by Heule, Kiesl, and Biere [33] as well as Bryant and Heule [13]. The `urquhart` formulas [16, 64] are examples of hard Tseitin formulas.

Lingeling solved each formula by Gaussian elimination almost instantly. We ran Lingeling and Kissat [10], winner of the main track of the SAT competition in 2020, on the benchmarks without Gaussian elimination, as is required for producing clausal refutations, using an Intel Xeon E5-2620 v4 CPU at 2.10 GHz. Only `rpar_50` was solved in under about 10 hours, producing significantly larger proofs; for instance, Kissat produced a refutation of size 6911 MB.

While methods to construct clausal proofs from Gaussian elimination have been proposed, most are either lacking a public implementation or are limited in scope [17, 55]. An exception is the approach very recently proposed by Gocht and Nordström using pseudo-Boolean reasoning [25], with which we are interested in carrying out a thorough comparison of results in the future.

## 7 Conclusion

We presented a characterization of redundancy for Boolean functions, generalizing the framework of clausal redundancy and efficient clausal proof systems. We showed this can be instantiated to design redundancy properties for functions given by BDDs, and polynomially-checkable refutation systems based on the conjunction of redundant BDDs, including the system $\mathsf{PR}_{\mathrm{BDD}}$ generalizing the clausal system PR. The system $\mathsf{PR}_{\mathrm{BDD}}$ also generalizes $\mathsf{RUP}_{\mathrm{BDD}}$, which can express Gaussian elimination reasoning without extension variables or clausal translations. The results of a preliminary implementation of a subset of $\mathsf{RUP}_{\mathrm{BDD}}$ confirms such refutations are compact and can be efficiently checked.

Examples 2 and 3 show $\mathsf{RUP}_{\mathrm{BDD}}$ reasoning over cardinality constraints, and we are interested in exploring rules such as *generalized resolution* [38, 39]. Other forms of non-clausal reasoning may be possible using BDD-based redundancy systems as well. We are particularly interested in exploring the property $\mathsf{IMP}_{\mathrm{pair}}$.

While the system $\mathsf{RUP}_{\mathrm{BDD}}$ derives only constraints implied by the conjunction of the formula and previously derived constraints, $\mathsf{PR}_{\mathrm{BDD}}$ is capable of *interference-based* reasoning [29], like its clausal analog PR; there are possibly novel, non-clausal reasoning techniques taking advantage of this ability. Further, $\mathsf{RUP}_{\mathrm{BDD}}$ and $\mathsf{PR}_{\mathrm{BDD}}$ are based on the conjunction of BDDs, though Theorem 2 is more general and could be used for other ways of expressing Boolean functions. Finally we are interested in developing an optimized tool for checking proofs in the system $\mathsf{PR}_{\mathrm{BDD}}$, as well as a certified proof checker.

**Acknowledgements.** We extend our thanks to Marijn Heule for his helpful comments on an earlier draft of this paper.

## Appendix

**Proposition 2.** *Suppose $f_1, \ldots, f_n$ are BDDs and $g$ is a non-constant BDD. If there is a partial assignment $\{l_1, \ldots, l_k\}$ such that for $\omega = \bigwedge_{i=1}^{k} l_i$,*

$$f_1|_{\neg g} \wedge \cdots \wedge f_n|_{\neg g} \vDash f_1|_{\omega} \wedge \cdots \wedge f_n|_{\omega}$$

*and $g|_\omega = 1$ then $g$ is redundant with respect to $f_1 \wedge \ldots \wedge f_n$.*

*Proof.* Let $f = f_1 \wedge \cdots \wedge f_n$ and $\alpha = \pi_{\neg g}$. We know $\alpha$ blocks $g$ by Lemma 1 and $f \circ \alpha \equiv f_1|_{\neg g} \wedge \cdots \wedge f_n|_{\neg g}$. Further, let $\sigma = \{l_1, \ldots, l_k\}$ so that $f \circ \hat{\sigma}$ is equal to $f|_\omega \equiv f_1|_\omega \wedge \cdots \wedge f_n|_\omega$. By Theorem 2 then $g$ is redundant with respect to $f$. □

**Proposition 3.** *If* $\mathsf{UnitProp}(f_1, \ldots, f_n)$ *returns "conflict" then $f_1 \wedge \cdots \wedge f_n \equiv 0$.*

*Proof.* Let $f_j^k$ refer to the BDD $f_j$ after $k$ iterations of the outer loop, and let $U_k$ refer to the conjunction of all units produced in iteration $k$. Then $f_j^k = f_j^{k-1}|_{U_k}$ for any $k > 0$. As $\bigwedge_{j=1}^{n} f_j^{k-1} \vDash U_k$, then $\bigwedge_{j=1}^{n} f_j^{k-1} \equiv \bigwedge_{j=1}^{n} f_j^{k-1} \wedge U_k$, also equivalent to $(\bigwedge_{j=1}^{n} f_j^{k-1})|_{U_k} \wedge U_k$. As this cofactor distributes over conjunction, this is equivalent to $\bigwedge_{j=1}^{n} f_j^{k-1}|_{U_k} \wedge U_k$. Thus we have $\bigwedge_{j=1}^{n} f_j^k \wedge U_k \equiv \bigwedge_{j=1}^{n} f_j^{k-1}$.

If "conflict" is returned in the first iteration, then clearly $\bigwedge_{j=1}^{n} f_j$ is unsatisfiable. By the reasoning above, after iteration $k$, inductively $\bigwedge_{j=1}^{n} f_j^k$ and $\bigwedge_{j=1}^{n} f_j^{k-1}$ are equisatisfiable. As a result, if "conflict" is returned after any number of iterations $k \geq 0$, then $\bigwedge_{j=1}^{n} f_j$ is unsatisfiable. □

**Proposition 4.** *If $f_1 \wedge \cdots \wedge f_n \vdash_1 g$, then $f_1 \wedge \cdots \wedge f_n \vDash g$.*

*Proof.* If $f_1 \wedge \cdots \wedge f_n \vdash_1 g$ then $f_1|_{\neg g} \wedge \cdots \wedge f_n|_{\neg g} \equiv 0$ by Proposition 3. Then for $f = \bigwedge_{i=1}^{n} f_i$, the BDD $f|_{\neg g}$ is the constant 0. As $f|_{\neg g}$ is a generalized cofactor of $f$ by $\neg g$, then in fact $f \wedge \neg g = f|_{\neg g} \wedge \neg g$ is unsatisfiable, and thus $f \vDash g$. □

**Lemma 2.** *$\mathcal{D}(X)$ implies $X$ by $\mathsf{RUP}_{\text{path}}$, for $X = [x_1, \ldots, x_n]^p$.*

*Proof.* Let $c = l_1 \wedge \cdots \wedge l_n$, with each $\text{var}(l_i) \in \{x_1, \ldots, x_n\}$, and suppose $X|_c = 0$, so that $l_1, \ldots, l_n$ is a path to 0 in $X$. The number of positive literals $l_i = x_i$ in $c$ is then not equal modulo 2 to $p$, so the number of negative literals in the clause $\neg c$ is not equal modulo 2 to $p$. Then $\neg c \in \mathcal{D}(X)$ and thus $\mathcal{D}(X) \vdash_1 \neg c$.

By the Shannon expansion $X = (x_1 \wedge X|_{x_1}) \vee (\neg x_1 \wedge X|_{\neg x_1})$, where $X|_{x_1}$ and $X|_{\neg x_1}$ are the functions expressed by the XOR clauses $[x_2, \ldots, x_n]^{\neg p}$ and $[x_2, \ldots, x_n]^p$, respectively. As $[x_n]^0$ and $[x_n]^1$ are equivalent to just $\neg x_n$ and $x_n$, respectively, then $|X| = 2n + 1$: with $x_1 \prec \cdots \prec x_n$, the BDD $X$ includes one node with variable $x_1$, and two nodes with $x_i$ for each $1 < i \leq n$. The formula $\mathcal{D}(X)$ includes $2^n - 1$ clauses each with $n$ literals, thus $|X| \leq \log |\mathcal{D}(X)|$. □

**Lemma 3.** *Let $v$ the $\prec$-greatest variable in occurring in exactly one of $X$ and $Y$, and assume $v$ occurs in $Y$. Then $X|_{\neg(X \oplus Y)} = X$, and $Y|_{\neg(X \oplus Y)} = \neg X$.*

*Proof.* Let $g = \neg(X \oplus Y)$ and suppose $X(\tau) = 1$. If $Y(\tau) = 0$, then $g(\tau) = 1$ and $\pi_g(\tau) = \tau$. If instead $Y(\tau) = 1$, then $g(\tau) = 0$. The nearest assignment (with respect to $d$) satisfying $g$ differs only on $\tau(v)$; that is, $\pi_g(\tau)(x) = \tau(x)$ for $x \neq v$, and $\pi_g(\tau)(v) = \neg\tau(v)$. This way $Y(\pi_g(\tau)) = 0$, and as $v$ does not occur in $X$ then still $X(\pi_g(\tau)) = 1$. In either case, $X \circ \pi_g(\tau) = X(\tau)$ and $Y \circ \pi_g(\tau) = \neg X(\tau)$.

Next, suppose $X(\tau) = 0$. If $Y(\tau) = 1$ then $g(\tau) = 1$ and $\pi_g(\tau) = \tau$. If instead $Y(\tau) = 0$, then also $g(\tau) = 0$. Again, $\pi_g(\tau)$ need only alter the assignment of $v$ to satisfy $Y$ and thus $g$; that is, $\pi_g(\tau)(x) = \tau(x)$ for $x \neq v$, and $\pi_g(\tau)(v) = \neg\tau(v)$. Now $Y(\pi_g(\tau)) = 1$, and again $X(\pi_g(\tau)) = 0$ is unaffected. Then $X \circ \pi_g(\tau) = X(\tau)$ and $Y \circ \pi_g(\tau) = \neg X(\tau)$ holds in either case. □

**Theorem 3.** *Suppose $\Pi = X_1, \ldots, X_N$ is a Gaussian elimination derivation from a formula $F$. Then there is a $\mathsf{RUP}_{\mathrm{BDD}}$ derivation from $F$ with size $O(|\Pi|)$.*

*Proof.* By assumption $\mathcal{D}(X_1) \subseteq F$, so that by Lemma 2, $F$ implies $X_1$ by $\mathsf{RUP}_{\mathrm{path}}$. Next, for $i > 1$, either $\mathcal{D}(X_i) \subseteq F$ as well, or $X_n = X_j \oplus X_k$ for $j, k < n$, and then $X_n$ is $\mathsf{RUP}_{\mathrm{BDD}}$ with respect to $F \wedge X_1 \wedge \cdots \wedge X_{n-1}$ by Lemma 3. Thus the sequence of BDDs representing $\sigma = X_1, \ldots, X_N$ is a $\mathsf{RUP}_{\mathrm{BDD}}$ derivation from $F$. As $|X_i| = 2 \cdot s_i + 1$ for $1 \leq i \leq N$, then $\sigma$ has size $O(|\Pi|)$. □

## References


1. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Mayer-Eichberger, V.: A new look at BDDs for pseudo-Boolean constraints. Journal of Artificial Intelligence Research **45**, 443–480 (2012). https://doi.org/10.1613/jair.3653
2. Ajtai, M.: The complexity of the pigeonhole principle. Combinatorica **14**(4), 417–433 (1994). https://doi.org/10.1007/BF01302964
3. Akers, S.B.: Binary decision diagrams. IEEE Trans. Computers **27**(6), 509–516 (1978). https://doi.org/10.1109/TC.1978.1675141
4. Balyo, T., Heule, M.J.H., Järvisalo, M.: SAT competition 2016: Recent developments. In: Singh, S.P., Markovitch, S. (eds.) 31st AAAI Conference on Artificial Intelligence. pp. 5061–5063. AAAI Press (2017)
5. Barnett, L.A., Cerna, D., Biere, A.: Covered clauses are not propagation redundant. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) 10th Intl. Joint Conference on Automated Reasoning – IJCAR. LNCS, vol. 12166, pp. 32–47. Springer (2020). https://doi.org/10.1007/978-3-030-51074-9_3
6. Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. pp. 1267–1329. IOS Press (2021). https://doi.org/10.3233/FAIA201017
7. Bayardo, R.J., Schrag, R.: Using CSP look-back techniques to solve real-world SAT instances. In: Kuipers, B., Webber, B.L. (eds.) 14th AAAI National Conference on Artificial Intelligence. pp. 203–208. AAAI Press (1997)
8. Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. Journal of Artificial Intelligence Research **22**(1), 319–351 (2004). https://doi.org/10.1613/jair.1410
9. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT competition 2018. In: Heule, M.J.H., Järvisalo, M., Suda, M. (eds.) Proc. of SAT Competition 2018. pp. 13–14. Department of Computer Science Series of Publications B, University of Helsinki (2018)

10. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
11. Biere, A., Järvisalo, M., Kiesl, B.: Preprocessing in SAT solving. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. pp. 391–435. IOS Press (2021). https://doi.org/10.3233/FAIA200992
12. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers **35**(8), 677–691 (1986). https://doi.org/10.1109/TC.1986.1676819
13. Bryant, R.E., Heule, M.J.H.: Generating extended resolution proofs with a BDD-based SAT solver. In: Groote, J.F., Larsen, K.G. (eds.) 27th Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems – TACAS. LNCS, vol. 12651, pp. 76–93. Springer (2021). https://doi.org/10.1007/978-3-030-72016-2_5
14. Burch, J.R., Clarke, E.M., Long, D.E., McMillan, K.L., Dill, D.L.: Symbolic model checking for sequential circuit verification. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **13**(4), 401–424 (1994). https://doi.org/10.1109/43.275352
15. Buss, S., Thapen, N.: DRAT proofs, propagation redundancy, and extended resolution. In: Janota, M., Lynce, I. (eds.) 22nd Intl. Conference on Theory and Applications of Satisfiability Testing – SAT. LNCS, vol. 11628, pp. 71–89. Springer (2019). https://doi.org/10.1007/978-3-030-24258-9_5
16. Chatalic, P., Simon, L.: Multi-resolution on compressed sets of clauses. In: 12th IEEE Intl. Conference on Tools with Artificial Intelligence – ICTAI. pp. 2–10. IEEE Computer Society (2000). https://doi.org/10.1109/TAI.2000.889839
17. Chew, L., Heule, M.J.H.: Sorting parity encodings by reusing variables. In: Pulina, L., Seidl, M. (eds.) 23rd Intl. Conference on Theory and Applications of Satisfiability Testing – SAT. LNCS, vol. 12178, pp. 1–10. Springer (2020). https://doi.org/10.1007/978-3-030-51825-7_1
18. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design **19**(1), 7–34 (2001). https://doi.org/10.1023/A:1011276507260
19. Coudert, O., Berthet, C., Madre, J.C.: Verification of synchronous sequential machines based on symbolic execution. In: Sifakis, J. (ed.) Intl. Workshop on Automatic Verification Methods for Finite State Systems. LNCS, vol. 407, pp. 365–373. Springer (1990). https://doi.org/10.1007/3-540-52148-8_30
20. Coudert, O., Madre, J.C.: A unified framework for the formal verification of sequential circuits. In: IEEE Intl. Conference on Computer-Aided Design – ICCAD. pp. 126–129. IEEE Computer Society (1990). https://doi.org/10.1109/ICCAD.1990.129859
21. Coudert, O., Madre, J.C., Berthet, C.: Verifying temporal properties of sequential machines without building their state diagrams. In: Clarke, E.M., Kurshan, R.P. (eds.) 2nd Intl. Workshop on Computer Aided Verification – CAV. LNCS, vol. 531, pp. 23–32. Springer (1990). https://doi.org/10.1007/BFb0023716
22. Damiano, R.F., Kukula, J.H.: Checking satisfiability of a conjunction of BDDs. In: 40th Design Automation Conference – DAC. pp. 818–823. ACM (2003). https://doi.org/10.1145/775832.776039
23. Franco, J., Kouril, M., Schlipf, J., Ward, J., Weaver, S., Dransfield, M., Vanfleet, W.M.: SBSAT: a state-based, BDD-based satisfiability solver. In: Giunchiglia,

E., Tacchella, A. (eds.) 6th Intl. Conference on Theory and Applications of Satisfiability Testing – SAT. LNCS, vol. 2919, pp. 398–410. Springer (2004). https://doi.org/10.1007/978-3-540-24605-3_30
24. Gelder, A.: Verifying RUP proofs of propositional unsatisfiability. In: 10th Intl. Symposium on Artificial Intelligence and Mathematics – ISAIM (2008)
25. Gocht, S., Nordström, J.: Certifying parity reasoning efficiently using pseudo-Boolean proofs. In: 35th AAAI Conference on Artificial Intelligence. AAAI Press (2021), to appear
26. Goldberg, E.I., Novikov, Y.: Verification of proofs of unsatisfiability for CNF formulas. In: Conference on Design, Automation and Test in Europe– DATE. pp. 886–891. IEEE Computer Society (2003). https://doi.org/10.1109/DATE.2003.10008
27. Goldberg, E.I., Prasad, M.R., Brayton, R.K.: Using SAT for combinational equivalence checking. In: Nebel, W., Jerraya, A. (eds.) Conference on Design, Automation and Test in Europe – DATE. pp. 114–121. IEEE Computer Society (2001). https://doi.org/10.1109/DATE.2001.915010
28. Groote, J.F., Tveretina, O.: Binary decision diagrams for first-order predicate logic. J. Log. Algebraic Methods Program. **57**(1-2), 1–22 (2003). https://doi.org/10.1016/S1567-8326(03)00039-0
29. Heule, M., Kiesl, B.: The potential of interference-based proof systems. In: Reger, G., Traytel, D. (eds.) 1st Intl. Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements – ARCADE. EPiC Series in Computing, vol. 51, pp. 51–54. EasyChair (2017)
30. Heule, M.J.H., Biere, A.: All about Proofs, Proofs for All, Mathematical Logic and Foundations, vol. 55, chap. Proofs for Satisfiability Problems, pp. 1–22. College Publications (2015)
31. Heule, M.J.H., Järvisalo, M., Lonsing, F., Seidl, M., Biere, A.: Clause elimination for SAT and QSAT. Journal of Artificial Intelligence Research **53**(1), 127–168 (2015). https://doi.org/10.1613/jair.4694
32. Heule, M.J.H., Kiesl, B., Biere, A.: Short proofs without new variables. In: de Moura, L. (ed.) 26th Intl. Conference on Automated Deduction – CADE. LNCS, vol. 10395, pp. 130–147. Springer (2017). https://doi.org/10.1007/978-3-319-63046-5_9
33. Heule, M.J.H., Kiesl, B., Biere, A.: Clausal proofs of mutilated chessboards. In: Badger, J.M., Rozier, K.Y. (eds.) 11th NASA Formal Methods Symposium – NFM. LNCS, vol. 11460, pp. 204–210. Springer (2019). https://doi.org/10.1007/978-3-030-20652-9_13
34. Heule, M.J.H., Kiesl, B., Biere, A.: Encoding redundancy for satisfaction-driven clause learning. In: Vojnar, T., Zhang, L. (eds.) 25th Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems – TACAS. LNCS, vol. 11427, pp. 41–58. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_3
35. Heule, M.J.H., Kiesl, B., Biere, A.: Strong extension-free proof systems. Journal of Automated Reasoning **64**(3), 533–554 (2020). https://doi.org/10.1007/s10817-019-09516-0
36. Heule, M.J.H., Kiesl, B., Seidl, M., Biere, A.: PRuning through satisfaction. In: Strichman, O., Tzoref-Brill, R. (eds.) 13th Intl. Haifa Verification Conference – HVC. LNCS, vol. 10629, pp. 179–194. Springer (2017). https://doi.org/10.1007/978-3-319-70389-3_12
37. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In: Creignou, N., Le Berre, D. (eds.) 19th Intl. Conference on Theory and Applications of Satisfiability Testing –

SAT. LNCS, vol. 9710, pp. 228–245. Springer (2016). https://doi.org/10.1007/978-3-319-40970-2_15
38. Hooker, J.N.: Generalized resolution and cutting planes. Annals of Operations Research **12**, 217–239 (1988). https://doi.org/10.1007/BF02186368
39. Hooker, J.N.: Generalized resolution for 0-1 linear inequalities. Annals of Mathematics and Artificial Intelligence **6**, 271–286 (1992). https://doi.org/10.1007/BF01531033
40. Hosaka, K., Takenaga, Y., Kaneda, T., Yajima, S.: Size of ordered binary decision diagrams representing threshold functions. Theor. Comput. Sci. **180**(1-2), 47–60 (1997). https://doi.org/10.1016/S0304-3975(97)83807-8
41. Järvisalo, M., Biere, A., Heule, M.J.H.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) 16th Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems – TACAS. LNCS, vol. 6015, pp. 129–144. Springer (2010). https://doi.org/10.1007/978-3-642-12002-2_10
42. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, Dalea nd Sattler, U. (eds.) 6th Intl. Joint Conference on Automated Reasoning – IJCAR. LNCS, vol. 7364, pp. 355–370. Springer (2012). https://doi.org/10.1007/978-3-642-31365-3_28
43. Kaiss, D., Skaba, M., Hanna, Z., Khasidashvili, Z.: Industrial strength SAT-based alignability algorithm for hardware equivalence verification. In: 7th Intl. Conference on Formal Methods in Computer Aided Design – FMCAD. pp. 20–26. IEEE Computer Society (2007). https://doi.org/10.1109/FAMCAD.2007.37
44. Kiesl, B., Seidl, M., Tompits, H., Biere, A.: Super-blocked clauses. In: Olivetti, N., Tiwari, A. (eds.) 8th Intl. Joint Conference on Automated Reasoning – IJCAR. LNCS, vol. 9706, pp. 45–61. Springer (2016). https://doi.org/10.1007/978-3-319-40229-1_5
45. Konev, B., Lisitsa, A.: Computer-aided proof of Erdős discrepancy properties. Artificial Intelligence **224**, 103–118 (2015). https://doi.org/10.1016/j.artint.2015.03.004
46. Kuehlmann, A., Krohm, F.: Equivalence checking using cuts and heaps. In: Yoffa, E.J., Micheli, G.D., Rabaey, J.M. (eds.) 34th Design Automation Conference – DAC. pp. 263–268. ACM (1997). https://doi.org/10.1145/266021.266090
47. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics **96-97**, 149–176 (1999). https://doi.org/10.1016/S0166-218X(99)00037-2
48. Lee, C.Y.: Representation of switching circuits by binary-decision programs. The Bell System Technical Journal **38**(4), 985–999 (1959)
49. Manthey, N.: Coprocessor 2.0 – a flexible CNF simplifier. In: Cimatti, A., Sebastiani, R. (eds.) 15th Intl. Conference on Theory and Applications of Satisfiability Testing – SAT 2012. LNCS, vol. 7317, pp. 436–441. Springer (2012). https://doi.org/10.1007/978-3-642-31612-8_34
50. Marques-Silva, J.P., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: IEEE Intl. Conference on Computer Aided Design – ICCAD. pp. 220–227. IEEE Computer Society / ACM (1996). https://doi.org/10.1109/ICCAD.1996.569607
51. Motter, D.B., Markov, I.L.: A compressed breadth-first search for satisfiability. In: Mount, D.M., Stein, C. (eds.) 4th Intl. Workshop on Algorithm Engineering and Experiments – ALENEX. LNCS, vol. 2409, pp. 29–42. Springer (2002). https://doi.org/10.1007/3-540-45643-0_3
52. Olivo, O., Emerson, E.A.: A more efficient BDD-based QBF solver. In: Lee, J. (ed.) 17th Intl. Conference on Principles and Practice of Constraint Programming

– CP. pp. 675–690. LNCS, Springer (2011). https://doi.org/10.1007/978-3-642-23786-7_51
53. Pan, G., Vardi, M.Y.: Search vs. symbolic techniques in satisfiability solving. In: 7th Intl. Conference on Theory and Applications of Satisfiability Testing – SAT. LNCS, vol. 3542, pp. 235–250. Springer (2004). https://doi.org/10.1007/11527695_19
54. Papadimitriou, C., Yannakakis, M.: The complexity of facets (and some facets of complexity). Journal of Computer and System Sciences **28**(2), 244–259 (1984). https://doi.org/10.1016/0022-0000(84)90068-0
55. Philipp, T., Rebola-Pardo, A.: DRAT proofs for XOR reasoning. In: Michael, L., Kakas, A.C. (eds.) 15th European Conference on Logics in Artificial Intelligence – JELIA. LNCS, vol. 10021, pp. 415–429 (2016). https://doi.org/10.1007/978-3-319-48758-8_27
56. Posegga, J., Ludäscher, B.: Towards first-order deduction based on Shannon graphs. In: Ohlbach, H.J. (ed.) 16th German Conference on Artificial Intelligence – GWAI. LNCS, vol. 671, pp. 67–75. Springer (1992). https://doi.org/10.1007/BFb0018993
57. Roussel, O., Manquinho, V.: Pseudo-Boolean and cardinality constraints. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. pp. 1087–1129. IOS Press (2021). https://doi.org/10.3233/978-1-58603-929-5-695
58. Sieling, D., Wegener, I.: Reduction of OBDDs in linear time. Information Processing Letters **48**(3), 139 – 144 (1993). https://doi.org/10.1016/0020-0190(93)90256-9
59. Sinz, C., Biere, A.: Extended resolution proofs for conjoining BDDs. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) Computer Science - Theory and Applications, 1st Intl. Computer Science Symposium in Russia – CSR. vol. 3967, pp. 600–611. Springer (2006). https://doi.org/10.1007/11753728_60
60. Soos, M., Gocht, S., Meel, K.S.: Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling. In: Lahiri, S.K., Wang, C. (eds.) 32nd Intl. Conference on Computer Aided Verification – CAV. LNCS, vol. 12224, pp. 463–484. Springer (2020). https://doi.org/10.1007/978-3-030-53288-8_22
61. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) 12th Intl. Conference on Theory and Applications of Satisfiability Testing – SAT. pp. 244–257. LNCS, Springer (2009). https://doi.org/10.1007/978-3-642-02777-2_24
62. Touati, H.J., Savoj, H., Lin, B., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: Implicit state enumeration of finite state machines using BDDs. In: IEEE Intl. Conference on Computer-Aided Design – ICCAD. pp. 130–133. IEEE Computer Society (1990). https://doi.org/10.1109/ICCAD.1990.129860
63. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Slissenko, A.O. (ed.) Studies in Constructive Mathematics and Mathematical Logic, vol. 2, pp. 115–125. Steklov Mathematical Institute (1970)
64. Urquhart, A.: Hard examples for resolution. Journal of the ACM **34**(1), 209–219 (1987). https://doi.org/10.1145/7531.8928
65. Urquhart, A.: The complexity of propositional proofs. Bulletin of Symbolic Logic **1**(4), 425–467 (12 1995). https://doi.org/10.2307/421131
66. Voronkov, A.: AVATAR: The architecture for first-order theorem provers. In: Biere, A., Bloem, R. (eds.) 26th Intl. Conference on Computer Aided Verification – CAV. LNCS, vol. 8559, pp. 696–710. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_46
67. Warners, J.P., Maaren, H.V., Warners, J.P., Maaren, H.V.: A two phase algorithm for solving a class of hard satisfiability problems. Operations Research Letters **23**, 81–88 (1998). https://doi.org/10.1016/S0167-6377(98)00052-2

68. Wetzler, N., Heule, M.J.H., Hunt, W.A.: DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In: Sinz, C., Egly, U. (eds.) 17th Intl. Conference on Theory and Applications of Satisfiability Testing – SAT. LNCS, vol. 8561, pp. 422–429. Springer (2014). https://doi.org/10.1007/978-3-319-09284-3_31