Short Proofs Without New Variables*

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

[1] Department of Computer Science, The University of Texas at Austin
[2] Institute of Information Systems, Vienna University of Technology
[3] Institute for Formal Models and Verification, JKU Linz

Abstract. Adding and removing redundant clauses is at the core of state-of-the-art SAT solving. Crucial is the ability to add short clauses whose redundancy can be determined in polynomial time. We present a characterization of the strongest notion of clause redundancy (i.e., addition of the clause preserves satisfiability) in terms of an implication relationship. By using a polynomial-time decidable implication relation based on unit propagation, we thus obtain an efficiently checkable redundancy notion. A proof system based on this notion is surprisingly strong, even without the introduction of new variables—the key component of short proofs presented in the proof complexity literature. We demonstrate this strength on the famous pigeon hole formulas by providing short clausal proofs without new variables.

1 Introduction

Satisfiability (SAT) solvers are used for determining the correctness of hardware and software systems [1,2]. It is therefore crucial that these solvers justify their claims by providing proofs that can be independently verified. This holds also for various other applications that use SAT solvers. Just recently, long-standing mathematical problems were solved using SAT, including the Erdős Discrepancy Problem [3] and the Pythagorean Triples Problem [4]. Especially in such cases, proofs are at the center of attention, and without them, the result of a solver is almost worthless. What the mathematical problems and the industrial applications have in common, is that proofs are often of considerable size—in the case of the Pythagorean Triples Problem about 200 terabytes. As the size of proofs is influenced by the strength of the underlying proof system, the search for shorter proofs goes hand in hand with the search for stronger proof systems.

In this paper, we introduce highly expressive clausal proof systems that are closely related to state-of-the-art SAT solving. Informally, a clausal proof system allows the addition of redundant clauses to a formula in conjunctive normal form (CNF). Here, a clause is considered *redundant* if its addition preserves satisfiability. If the repeated addition of clauses allows us finally to add the empty clause—which is, by definition, unsatisfiable—the unsatisfiability of the original formula has been established.

* This work has been supported by the National Science Foundation under grant CCF-1526760 and the Austrian Science Fund (FWF) under project W1255-N23.

Since satisfiability equivalence is not efficiently decidable, practical proof systems only allow the addition of a clause if it fulfills some efficiently decidable criterion that ensures redundancy. For instance, the popular DRAT proof system [5], which is the de-facto standard in practical SAT solving, only allows the addition of so-called *resolution asymmetric tautologies* [6]. Given a formula and a clause, one can decide in polynomial time whether the clause is a resolution asymmetric tautology with respect to the formula and therefore the soundness of DRAT proofs can be efficiently checked.

We present new redundancy criteria by introducing a characterization of clause redundancy based on a simple implication relationship between formulas. By replacing the logical implication relation in this characterization with stronger notions of implication that are computable in polynomial time, we then obtain powerful redundancy criteria that are still efficiently decidable. We show that these redundancy criteria not only generalize earlier ones like the above-mentioned resolution asymmetric tautologies or *set-blocked clauses* [7], but that they are also related to other concepts from the literature, namely *autarkies* [8], *safe assignments* [9], *variable instantiation* [10], and *symmetry breaking* [11].

Proof systems based on our new redundancy criteria turn out to be highly expressive, even without the introduction of new variables. This is in contrast to resolution, which is considered relatively weak as long as one does not allow the introduction of new variables via definitions as in the stronger proof system of *extended resolution* [12,13]. The introduction of new variables, however, has a major drawback: the search space of variables and clauses one could possibly add to a proof is infinite, even when bounding the size of clauses. Finding useful clauses with new variables is therefore hard in practice, although there have been a few successes in the past [14,15].

We illustrate the strength of our strongest proof system by providing short clausal proofs for the famous pigeon hole formulas without introducing new variables. The size of the proofs is linear in the size of the formulas and the longest clauses in the proofs have length two. In these proofs, we add redundant clauses that are similar in nature to symmetry-breaking predicates [11,16]. To verify the correctness of proofs in our new system, we implemented a proof checker. The checker is built on top of `DRAT-trim` [5], the checker used to validate the unsatisfiability results of the recent SAT competitions [17]. We compare our proofs with existing proofs of the pigeon hole formulas in other proof systems and show that our new proofs are much smaller and cheaper to validate.

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 $\overline{x}$ of a variable x (a *negative literal*). The *complementary literal* $\bar{l}$ of a literal l is defined as $\bar{l} = \overline{x}$ if $l = x$ and $\bar{l} = x$ if $l = \overline{x}$. Accordingly, for a set L of literals, we define $\overline{L} = \{\bar{l} \mid l \in L\}$. A *clause* is a disjunction of literals. If not stated otherwise, we assume that clauses do not contain complementary literals.

A *formula* is a conjunction of clauses. We view clauses as sets of literals and formulas as sets of clauses. For a set L of literals and a formula F, we define $F_L = \{C \in F \mid C \cap L \neq \emptyset\}$. We sometimes write F_l to denote $F_{\{l\}}$.

An *assignment* is a partial function from a set of variables to the truth values 1 (*true*) and 0 (*false*). An assignment is *total* w.r.t. a formula if it assigns a truth value to every variable occurring in the formula. A literal l is *satisfied* (*falsified*) by an assignment α if l is positive and $\alpha(var(l)) = 1$ ($\alpha(var(l)) = 0$, respectively) or if it is negative and $\alpha(var(l)) = 0$ ($\alpha(var(l)) = 1$, respectively). We often denote assignments by the sequences of literals they satisfy. For instance, $x\,\overline{y}$ denotes the assignment that assigns x to 1 and y to 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 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.

We denote the empty clause by $\bot$ and the satisfied clause by $\top$. Given an assignment α and a clause C, we define $C|\alpha = \top$ if α satisfies C, otherwise $C|\alpha$ denotes the result of removing from C all the literals falsified by α. Moreover, for a formula F, we define $F|\alpha = \{C|\alpha \mid C \in F \text{ and } C|\alpha \neq \top\}$. We say that a clause C *blocks* an assignment α if $C = \{x \mid \alpha(x) = 0\} \cup \{\overline{x} \mid \alpha(x) = 1\}$. A *unit clause* is a clause that contains only one literal. The result of applying the *unit clause rule* to a formula F is the formula $F|\alpha$ with α being the assignment that satisfies exactly the unit clauses in F. The iterated application of the unit clause rule to a formula, until no unit clauses are left, is called *unit propagation*. If unit propagation yields the empty clause $\bot$, we say that it derived a *conflict*.

By $F \vDash F'$, we denote that F implies F', i.e., all assignments satisfying F also satisfy F'. Furthermore, by $F \vdash_1 F'$ we denote that for every clause $C \in F'$, unit propagation of the negated literals of C on F derives a conflict (thereby, the negated literals of C are viewed as unit clauses). For example, $x \wedge y \vdash_1 (x \vee z) \wedge y$, since unit propagation of the unit clauses $\overline{x}$ and $\overline{z}$ derives a conflict with x, and propagation of $\overline{y}$ derives a conflict with y. Similarly, $F \vdash_0 F'$ denotes that every clause in F' is subsumed by (i.e., is a superset of) a clause in F. Observe that $F \supseteq F'$ implies $F \vdash_0 F'$, $F \vdash_0 F'$ implies $F \vdash_1 F'$, and $F \vdash_1 F'$ implies $F \vDash F'$.

3 Clause Redundancy and Clausal Proofs

In this section, we introduce a formal notion of clause redundancy and demonstrate how it provides the basis for clausal proof systems. We start by introducing clause redundancy [7]:

Definition 1. *A clause C is* redundant *w.r.t. a formula F if F and $F \cup \{C\}$ are satisfiability equivalent.*

For instance, the clause $C = x \vee y$ is redundant w.r.t. $F = \{\overline{x} \vee \overline{y}\}$ since F and $F \cup \{C\}$ are satisfiability equivalent (although they are not logically equivalent). Since this notion of redundancy allows us to add redundant clauses to a formula without affecting its satisfiability, it gives rise to clausal proof systems.

Definition 2. *A* proof *of a clause C_m from a formula F is a sequence of pairs $(C_1, \omega_1), \ldots, (C_m, \omega_m)$, where each C_i $(1 \leq i \leq m)$ is a clause that is redundant w.r.t. $F \cup \{C_j \mid 1 \leq j < i\}$, and this redundancy can be efficiently checked using the (arbitrary)* witness *ω_i. If $C_m = \bot$, the proof is a* refutation *of F.*

Clearly, since every clause-addition step preserves satisfiability, and since the empty clause is unsatisfiable, a refutation certifies the unsatisfiability of F due to transitivity. Note that the ω_i can be arbitrary witnesses (they can be assignments, or even left out if no explicit witness is needed) that certify the redundancy of C_i w.r.t. $F \cup \{C_j \mid 1 \leq j < i\}$, and by requiring that the redundancy can be *efficiently checked*, we mean that it can be checked in polynomial time w.r.t. the size of $F \cup \{C_j \mid 1 \leq j < i\}$.

By specifying in detail what kind of redundant clauses—and corresponding witnesses—one can add to a proof, we obtain concrete proof systems. This is usually done by defining an efficiently checkable syntactic criterion that guarantees that clauses fulfilling this criterion are redundant. A popular example for a clausal proof system is DRAT [5], the de-facto standard for unsatisfiability proofs in practical SAT solving. DRAT allows the addition of a clause if it is a so-called *resolution asymmetric tautology* [6] (RAT, defined in the next section). As it can be efficiently checked whether a clause is a RAT, and since RATs cover a large portion of redundant clauses, the DRAT proof system is very powerful.

The strength of a clausal proof system depends on the generality of the underlying redundancy criterion. We say that a redundancy criterion $\mathcal{R}_1$ is *more general* than a redundancy criterion $\mathcal{R}_2$ if, whenever $\mathcal{R}_2$ identifies a clause C as redundant w.r.t. a formula F, then $\mathcal{R}_1$ also identifies C as redundant w.r.t. F. For instance, whenever a clause is subsumed in some formula, it is a RAT w.r.t. that formula. Therefore, the RAT redundancy criterion is more general than the subsumption criterion. In the next section, we develop redundancy criteria that are even more general than RAT. This gives rise to proof systems that are stronger than DRAT but still closely related to practical SAT solving.

4 Clause Redundancy via Implication

In the following, we introduce a characterization of clause redundancy that reduces the question whether a clause is redundant w.r.t. a certain formula to a simple question of implication. The advantage of this is that we can replace the logical implication relation by stronger, polynomially decidable implication relations to derive powerful redundancy criteria that are still efficiently checkable. These redundancy criteria can then be used to obtain highly expressive clausal proof systems.

Our characterization is based on the observation that a clause in a CNF formula can be seen as a constraint that blocks those assignments falsifying the clause. Therefore, a clause can be safely added to a formula if it does not constrain the formula too much. What we mean by this, is that after adding the clause, there should still exist other assignments (i.e., assignments not blocked

by the clause) under which the formula is at least as satisfiable as under the assignments blocked by the clause. Consider the following example:

Example 1. Let $F = \{x \vee y,\ x \vee z,\ \overline{x} \vee y \vee z\}$ and consider the (unit) clause $C = x$ which blocks all assignments that assign x to 0. The addition of C to F does not affect satisfiability: Let $\alpha = \overline{x}$ and $\omega = x$. Then, $F|_\alpha = \{y,\ z\}$ while $F|_\omega = \{y \vee z\}$. Clearly, every satisfying assignment of $F|_\alpha$ is also a satisfying assignment of $F|_\omega$ (i.e., $F|_\alpha \vDash F|_\omega$). Thus, F is at least as satisfiable under ω as it is under α. Moreover, ω satisfies C. The addition of C does therefore not affect the satisfiability of F. □

This motivates the characterization of clause redundancy we introduce next. Note that for a given clause C, "*the* assignment α blocked by C" can be a partial assignment, meaning that C actually rules out all assignments that extend α:

Theorem 1. *Let F be a formula, C a clause, and α the assignment blocked by C. Then, C is redundant w.r.t. F if and only if there exists an assignment ω such that ω satisfies C and $F|_\alpha \vDash F|_\omega$.*

Proof. For the "only if" direction, assume that F and $F \cup \{C\}$ are satisfiability equivalent. If $F|_\alpha$ is unsatisfiable, then $F|_\alpha \vDash F|_\omega$ for every ω, hence the statement trivially holds. Assume now that $F|_\alpha$ is satisfiable, implying that F is satisfiable. Then, since F and $F \cup \{C\}$ are satisfiability equivalent, there exists an assignment ω that satisfies both F and C. Since ω satisfies F, it holds that $F|_\omega = \emptyset$ and so $F|_\alpha \vDash F|_\omega$.

For the "if" direction, assume that there exists an assignment ω such that ω satisfies C and $F|_\alpha \vDash F|_\omega$. Now, let γ be a (total) assignment that satisfies F and assume it falsifies C. As γ falsifies C, it coincides with α on $var(\alpha)$. Therefore, since γ satisfies F, it must satisfy $F|_\alpha$ and since $F|_\alpha \vDash F|_\omega$ it must also satisfy $F|_\omega$. Now, consider the following assignment γ':

$$\gamma'(x) = \begin{cases} \omega(x) & \text{if } x \in var(\omega), \\ \gamma(x) & \text{otherwise.} \end{cases}$$

Clearly, since ω satisfies C, γ' also satisfies C. Moreover, as γ satisfies $F|_\omega$ and $var(F|_\omega) \subseteq var(\gamma) \setminus var(\omega)$, γ' satisfies F. Hence, γ' satisfies $F \cup \{C\}$. □

This alternative characterization of redundancy allows us to replace the logical implication relation by stronger polynomially decidable relations. For instance, we can replace the condition $F|_\alpha \vDash F|_\omega$ by the stronger condition $F|_\alpha \vdash_1 F|_\omega$ (likewise, we could also use relations such as "$\vdash_0$" or "$\supseteq$" instead of "$\vdash_1$"). Now, if we are given a clause C—which implicitly gives us the blocked assignment α—and a *witnessing assignment* ω, then we can check in polynomial time whether $F|_\alpha \vdash_1 F|_\omega$, implying that C is redundant w.r.t. F. We can therefore use this implication-based redundancy notion to define proof systems. A proof is then a sequence $(C_1, \omega_1), \ldots, (C_m, \omega_m)$ where the ω_i are the witnessing assignments.

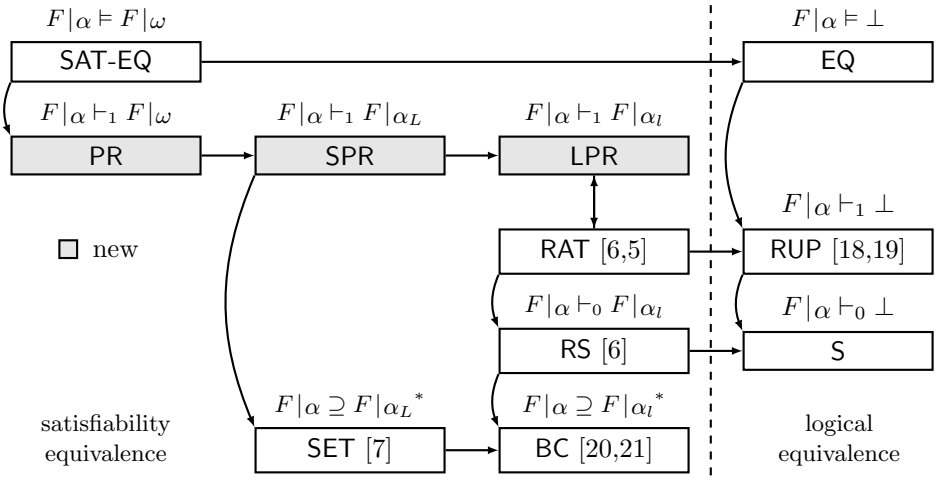


Fig. 1. Landscape of Redundancy Notions. SAT-EQ stands for all redundant clauses and EQ for implied clauses. A path from X to Y indicates that X is more general than Y. The asterisk (*) denotes that the exact characterization implies the shown one, e.g., for every set-blocked clause, the property $F|_\alpha \supseteq F|_{\alpha_L}$ holds, but not vice versa.

In the following, we use the propagation-implication relation "$\vdash_1$" to define the redundancy criteria of (1) *literal-propagation redundancy* (LPR), (2) *set-propagation redundancy* (SPR), and (3) *propagation redundancy* (PR). Basically, the three notions differ in the way we allow the witnessing assignment ω to differ from the assignment α blocked by a clause. The more freedom we give to ω, the more general the redundancy notion we obtain. We show that LPR clauses—the least general of the three—coincide with RAT. For the more general SPR clauses, we show that they generalize set-blocked clauses (SET) [7], which is not the case for LPR clauses. Finally, PR clauses are the most general ones. They give rise to an extremely powerful proof system that is still closely related to CDCL-based SAT solving. The new landscape of redundancy notions we thereby obtain is illustrated in Fig. 1. In the figure, RUP stands for the redundancy notion based on reverse unit propagation [18,19], S stands for subsumed clauses, RS for clauses with subsumed resolvents [6], and BC for blocked clauses [20,21].

As we will see, when defining proof systems based on LPR (e.g., the DRAT system) or SPR clauses, we do not need to add the explicit redundancy witnesses (i.e., the witnessing assignments ω) to a proof. In these two cases, a proof can thus just be seen as a sequence of clauses. A proof system based on SPR clauses can therefore have the same syntax as DRAT proofs, which makes it "downwards compatible". This is in contrast to a proof system based on PR clauses where, at least in general, we have to add the witnessing assignments to a proof, otherwise we cannot check the redundancy of a clause in polynomial time.

We start by introducing LPR clauses. In the following, given a (partial) assignment α and a set L of literals, we denote by α_L the assignment obtained

from α by flipping the truth values of the literals in L. If L contains only a single literal l, then we write α_l to denote $\alpha_{\{l\}}$.

Definition 3. *Let F be a formula, C a clause, and α the assignment blocked by C. Then, C is* literal-propagation redundant (LPR) *w.r.t. F if it contains a literal l such that $F|_\alpha \vdash_1 F|_{\alpha_l}$.*

Example 2. Let $F = \{x \vee y,\ x \vee \overline{y} \vee z,\ \overline{x} \vee z\}$ and let C be the unit clause x. Then, $\alpha = \overline{x}$ is the assignment blocked by C, and $\alpha_x = x$. Now, consider $F|_\alpha = \{y,\ \overline{y} \vee z\}$ and $F|_{\alpha_x} = \{z\}$. Clearly, $F|_\alpha \vdash_1 F|_{\alpha_x}$ and therefore C is literal-propagation redundant w.r.t. F. □

The LPR definition is quite restrictive, as it requires the witnessing assignment α_l to disagree with α on exactly one variable. Nevertheless, this already suffices for LPR clauses to coincide with RATs [6]:

Definition 4. *Let F be a formula and C a clause. Then, C is a* resolution asymmetric tautology (RAT) *w.r.t. F if it contains a literal l such that, for every clause $D \in F_{\bar{l}}$, $F \vdash_1 C \cup (D \setminus \{\bar{l}\})$.*

Theorem 2. *A clause C is literal-propagation redundant w.r.t. a formula F if and only if it is a resolution asymmetric tautology w.r.t. F.*

Proof. For the "only if" direction, assume that C is LPR w.r.t. F, i.e., it contains a literal l such that $F|_\alpha \vdash_1 F|_{\alpha_l}$. Now, let $D \in F_{\bar{l}}$. We have to show that $F \vdash_1 C \cup (D \setminus \{\bar{l}\})$. First, note that $F|_\alpha$ is exactly the result of propagating the negated literals of C on F (i.e., applying the unit clause rule with the negated literals of C but not performing further propagations). Moreover, since α_l falsifies $\bar{l}$, it follows that $D|_{\alpha_l} \subseteq (D \setminus \{\bar{l}\})$. But then, since $F|_\alpha \vdash_1 D|_{\alpha_l}$, it must hold that $F \vdash_1 C \cup (D \setminus \{\bar{l}\})$, hence C is a RAT w.r.t. F.

For the " if" direction, assume that C is a RAT w.r.t. F, i.e., it contains a literal l such that, for every clause $D \in F_{\bar{l}}$, $F \vdash_1 C \cup (D \setminus \{\bar{l}\})$. Now, let $D|_{\alpha_l} \in F|_{\alpha_l}$ for $D \in F$. We have to show that $F|_\alpha \vdash_1 D|_{\alpha_l}$. Since α_l satisfies l and α falsifies C, D does neither contain l nor any negations of literals in C except for possibly $\bar{l}$. If D does not contain $\bar{l}$, then $D|_\alpha = D|_{\alpha_l}$ is contained in $F|_\alpha$ and hence the claim immediately follows. Assume therefore that $\bar{l} \in D$.

As argued in the proof for the other direction, propagating the negated literals of C (and no other literals) on F yields $F|_\alpha$. Therefore, since $F \vdash_1 C \cup (D \setminus \{\bar{l}\})$ and $D \setminus \{\bar{l}\}$ does not contain any negations of literals in C (which could otherwise be the reason for a unit propagation conflict that only happens because of C containing a literal whose negation is contained in $D \setminus \{\bar{l}\}$), it must be the case that $F|_\alpha \vdash_1 D \setminus \{\bar{l}\}$. Now, the only literals of $D \setminus \{\bar{l}\}$ that are not contained in $D|_{\alpha_l}$ are the ones falsified by α, but those are anyhow not contained in $F|_\alpha$. It follows that $F|_\alpha \vdash_1 D|_{\alpha_l}$ and thus C is LPR w.r.t. F. □

By allowing the witnessing assignments to disagree with α on more than only one literal, we obtain the more general notion of set-propagation-redundant clauses:

Definition 5. *Let F be a formula, C a clause, and α the assignment blocked by C. Then, C is* set-propagation redundant (SPR) *w.r.t. F if it contains a non-empty set L of literals such that $F|_\alpha \vdash_1 F|_{\alpha_L}$.*

Example 3. Let $F = \{x \vee y,\ x \vee \overline{y} \vee z,\ \overline{x} \vee z,\ \overline{x} \vee u,\ \overline{u} \vee x\}$, $C = x \vee u$, and $L = \{x, u\}$. Then, $\alpha = \overline{x}\,\overline{u}$ is the assignment blocked by C, and $\alpha_L = x\,u$. Now, consider $F|_\alpha = \{y,\ \overline{y} \vee z\}$ and $F|_{\alpha_L} = \{z\}$. Clearly, $F|_\alpha \vdash_1 F|_{\alpha_L}$ and so C is set-propagation redundant w.r.t. F. Note also that C is not literal-propagation redundant w.r.t. F. □

Since L is a subset of C, we do not need to add it (or the assignment α_L) explicitly to an SPR proof. By requiring that L must consist of the first literals of C when adding C to a proof (viewing a clause as a sequence of literals), we can ensure that the SPR property is efficiently decidable. For instance, when a proof contains the clause $l_1 \vee \cdots \vee l_n$, we first check whether the SPR property holds under the assumption that $L = \{l_1\}$. If not, we proceed by assuming that $L = \{l_1, l_2\}$, and so on until $L = \{l_1, \ldots, l_n\}$. Thereby, only linearly many candidates for L need to be checked. In contrast to LPR clauses and RATs, the notion of SPR clauses generalizes set-blocked clauses [7]:

Definition 6. *A clause C is* set-blocked (SET) *by a non-empty set $L \subseteq C$ in a formula F if, for every clause $D \in F_{\overline{L}}$, the clause $(C \setminus L) \cup \overline{L} \cup D$ contains two complementary literals.*

To show that set-propagation-redundant clauses generalize set-blocked clauses, we first characterize them as follows:

Lemma 3. *Let F be a clause, C a formula, $L \subseteq C$ a non-empty set of literals, and α the assignment blocked by C. Then, C is set-blocked by L in F if and only if, for every $D \in F$, $D|_\alpha = \top$ implies $D|_{\alpha_L} = \top$.*

Proof. For the "only if" direction, assume that there exists a clause $D \in F$ such that $D|_\alpha = \top$ but $D|_{\alpha_L} \neq \top$. Then, since α and α_L disagree only on literals in L, it follows that D contains a literal $l \in \overline{L}$ and therefore $D \in F_{\overline{L}}$. Now, α_L falsifies exactly the literals in $(C \setminus L) \cup \overline{L}$ and since it does not satisfy any of the literals in D, it follows that there exists no literal $l \in D$ such that its complement $\overline{l}$ is contained in $(C \setminus L) \cup \overline{L}$. Therefore, C is not set-blocked by L in F.

For the "if" direction, assume that C is not set-blocked by L in F, i.e., there exists a clause $D \in F_{\overline{L}}$ such that $(C \setminus L) \cup \overline{L} \cup D$ does not contain complementary literals. Clearly, $D|_\alpha = \top$ since α falsifies L and $D \cap \overline{L} \neq \emptyset$. Now, since D contains no literal l such that $\overline{l} \in (C \setminus L) \cup \overline{L}$ and since α_L falsifies exactly the literals in $(C \setminus L) \cup \overline{L}$, it follows that α_L does not satisfy D, hence $D|_{\alpha_L} \neq \top$. □

Theorem 4. *If a clause C is set-blocked by a set L in a formula F, it is set-propagation redundant w.r.t. F.*

Proof. Assume that C is set-blocked by L in F. We show that $F|_\alpha \supseteq F|_{\alpha_L}$, which implies that $F|_\alpha \vdash_1 F|_{\alpha_L}$, and therefore that C is set-propagation redundant w.r.t. F. Let $D|_{\alpha_L} \in F|_{\alpha_L}$. First, note that D cannot be contained

in F_L, for otherwise $D|\alpha_L = \top$ and thus $D|\alpha_L \notin F|\alpha_L$. Second, observe that D can also not be contained in $F_{\overline{L}}$, since that would imply that $D|\alpha = \top$ and thus, by Lemma 3, $D|\alpha_L = \top$. Therefore, $D \notin F_L \cup F_{\overline{L}}$ and so $D|\alpha = D|\alpha_L$. But then, $D|\alpha_L \in F|\alpha$. It follows that $F|\alpha \supseteq F|\alpha_L$. □

We thus know that set-propagation-redundant clauses generalize both resolution asymmetric tautologies and set-blocked clauses. Since there exist resolution asymmetric tautologies that are not set-blocked (and vice versa) [7], it follows that set-propagation-redundant clauses are actually a *strict* generalization of these two kinds of clauses.

By giving practically full freedom to the witnessing assignments, i.e., by only requiring them to satisfy C, we finally arrive at propagation-redundant clauses, the most general of the three redundancy notions:

Definition 7. *Let F be a formula, C a clause, and α the assignment blocked by C. Then, C is* propagation redundant (PR) *w.r.t. F if there exists an assignment ω such that ω satisfies C and $F|\alpha \vdash_1 F|\omega$.*

Example 4. Let $F = \{x \vee y,\ \overline{x} \vee y,\ \overline{x} \vee z\}$, $C = x$, and let $\omega = x\,z$ be the witnessing assignment. Then, $\alpha = \overline{x}$ is the assignment blocked by C. Now, consider $F|\alpha = \{y\}$ and $F|\omega = \{y\}$. Clearly, unit propagation with the negated literal $\overline{y}$ of the unit clause $y \in F|\omega$ derives a conflict on $F|\alpha$. Therefore, $F|\alpha \vdash_1 F|\omega$ and so C is propagation redundant w.r.t. F. Note that C is not set-propagation redundant because for $L = \{x\}$, we have $\alpha_L = x$ and so $F|\alpha_L$ contains the two unit clauses y and z, but it does not hold that $F|\alpha \vdash_1 z$. The fact that ω satisfies z is crucial for ensuring propagation redundancy. □

Since the witnessing assignments ω are allowed to assign variables that are not contained in C, we need—at least in general—to add them to a proof to guarantee that redundancy can be efficiently checked. In the next section, we illustrate the power of a proof system that is based on the addition of PR clauses.

5 Short Proofs of the Pigeon Hole Principle

In a landmark paper, Haken [13] showed that pigeon hole formulas cannot be refuted by resolution proofs that are of polynomial size w.r.t. the size of the formulas. In contrast, by using the stronger proof system of *extended resolution*, Cook [22] proved that one can actually refute pigeon hole formulas in polynomial size. What distinguishes extended resolution from general resolution is that it allows for the introduction of new variables via definitions. Cook showed how the introduction of such definitions helps to reduce a pigeon hole formula of size n to a pigeon hole formula of size $n - 1$ over new variables. The problem with the introduction of new variables, however, is that the search space of possible variables—and therefore clauses—that could be added to a proof is infinite.

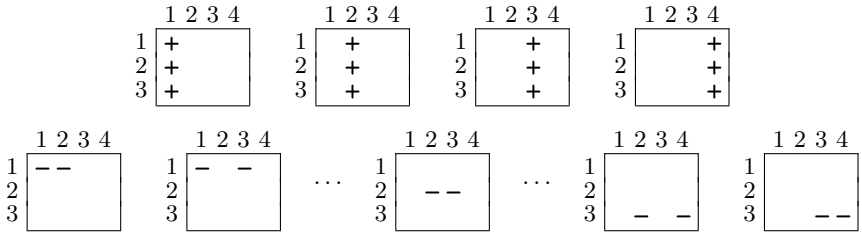
In this section, we illustrate how a clausal proof system that allows the addition of PR clauses can yield short proofs of pigeon hole formulas without the

need for introducing new variables. This shows that a proof system based on PR clauses is strictly stronger than the resolution calculus, even when we forbid the introduction of new variables. To recap, a pigeon hole formula PHP_n intuitively encodes that n pigeons have to be assigned to $n-1$ holes such that no hole contains more than one pigeon. In the encoding, a variable $x_{i,k}$ intuitively denotes that pigeon i is assigned to hole k:

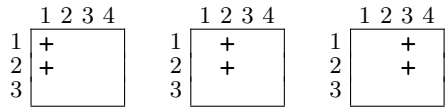
$$PHP_n := \bigwedge_{1 \le i \le n} (x_{i,1} \vee \cdots \vee x_{i,n-1}) \wedge \bigwedge_{1 \le i < j \le n} \bigwedge_{1 \le k \le n-1} (\overline{x}_{i,k} \vee \overline{x}_{j,k})$$

Clearly, pigeon hole formulas are unsatisfiable. The main idea behind our approach is similar to that of Cook, namely to reduce a pigeon hole formula PHP_n to the smaller PHP_{n-1}. The difference is, that in our case, PHP_{n-1} is still defined on the same variables as PHP_n. Therefore, reducing PHP_n to PHP_{n-1} boils down to deriving the clauses $x_{i,1} \vee \cdots \vee x_{i,n-2}$ for $1 \le i \le n-1$.

Following Haken [13], we use array notation for clauses: Every clause is represented by an array of n columns and $n-1$ rows. An array contains a "+" ("−") in the i-th column and k-th row if and only if the variable $x_{i,k}$ occurs positively (negatively, respectively) in the corresponding clause. Representing PHP_n in array notation, we have for every clause $x_{i,1} \vee \cdots \vee x_{i,n-1}$, an array in which the i-th column is filled with "+". Moreover, for every clause $\overline{x}_{i,k} \vee \overline{x}_{j,k}$, we have an array that contains two "−" in row k—one in column i and the other in column j. For instance, PHP_4 is given in array notation as follows:

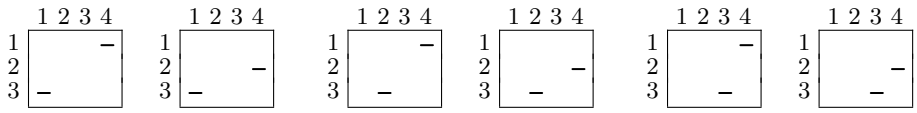


We illustrate the general idea for reducing a pigeon hole formula PHP_n to the smaller PHP_{n-1} on the concrete formula PHP_4. It should, however, become clear from our explanation that the procedure works for every $n > 1$. If we want to reduce PHP_4 to PHP_3, we have to obtain the following three clauses:



We can do so, by removing the "+" from the last row of every column full of "+", except for the last column, which can be ignored as it is not contained in PHP_3. The key observation is, that a "+" in the last row of the i-th column can be removed with the help of so-called "diagonal clauses" of the form $\overline{x}_{i,n-1} \vee \overline{x}_{n,k}$ $(1 \le k \le n-2)$. We are allowed to add these diagonal clauses since they are, as we will show, propagation redundant w.r.t. PHP_n. The arrays below represent

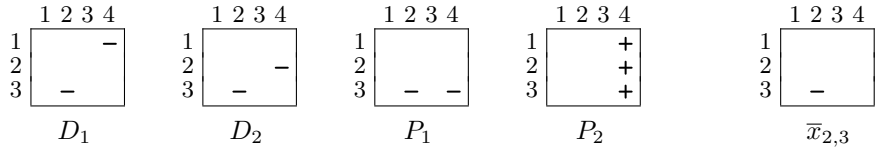
the diagonal clauses to remove the "+" from the last row of the first (left), second (middle), and third column (right):



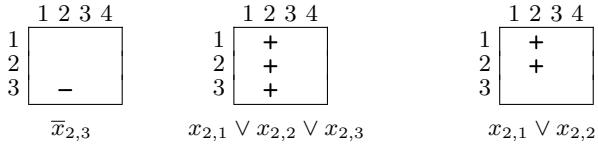
We next show how exactly these diagonal clauses allow us to remove the bottom "+" from a column full of "+", or, in other words, how they help us to remove the literal $x_{i,n-1}$ from a clause $x_{i,1} \vee \cdots \vee x_{i,n-1}$ $(1 \leq i \leq n-1)$. Consider, for instance, the clause $x_{2,1} \vee x_{2,2} \vee x_{2,3}$ in PHP_4. Our aim is to remove the literal $x_{2,3}$ from this clause. Before we explain the procedure, we like to remark that proof systems based on propagation redundancy can easily simulate resolution: Since every resolvent of clauses in a formula F is implied by F, the assignment α blocked by the resolvent must falsify F and thus $F|_\alpha \vdash_1 \bot$. We explain our procedure textually before we illustrate it in array notation:

First, we add the diagonal clauses $D_1 = \overline{x}_{2,3} \vee \overline{x}_{4,1}$ and $D_2 = \overline{x}_{2,3} \vee \overline{x}_{4,2}$ to PHP_4. After this, we can derive the unit clause $\overline{x}_{2,3}$ by resolving the two diagonal clauses D_1 and D_2 with the original pigeon hole clauses $P_1 = \overline{x}_{2,3} \vee \overline{x}_{4,3}$ and $P_2 = x_{4,1} \vee x_{4,2} \vee x_{4,3}$ as follows: We resolve D_1 with P_2 to obtain $\overline{x}_{2,3} \vee x_{4,2} \vee x_{4,3}$. Then, we resolve this clause with D_2 to obtain $\overline{x}_{2,3} \vee x_{4,3}$, which we resolve with P_1 to obtain $\overline{x}_{2,3}$. Note that our proof system actually allows us to add $\overline{x}_{2,3}$ immediately without carrying out all the resolution steps explicitly. Finally, we resolve $\overline{x}_{2,3}$ with $x_{2,1} \vee x_{2,2} \vee x_{2,3}$ to obtain the desired clause $x_{2,1} \vee x_{2,2}$.

We next illustrate this procedure in array notation. We start by visualizing the clauses D_1, D_2, P_1, and P_2 that can be resolved to yield the clause $\overline{x}_{2,3}$. The clauses are given in array notation as follows:



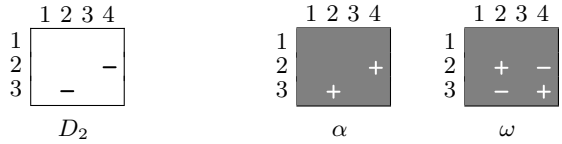
We can then resolve $\overline{x}_{2,3}$ with $x_{2,1} \vee x_{2,2} \vee x_{2,3}$ to obtain $x_{2,1} \vee x_{2,2}$:



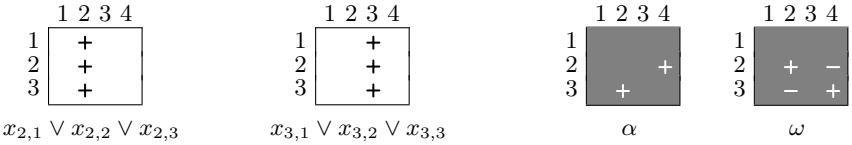
This should illustrate the general idea of how to reduce a clause of the form $x_{i,1} \vee \ldots x_{i,n-1}$ $(1 \leq i \leq n-1)$ to a clause $x_{i,1} \vee \ldots x_{i,n-2}$. By repeating this procedure for every column i with $1 \leq i \leq n-1$, we can thus reduce a pigeon hole formula PHP_n to a pigeon hole formula PHP_{n-1} without introducing new variables. Note that the last step, in which we resolve the derived unit clause $\overline{x}_{2,3}$ with the clause $x_{2,1} \vee x_{2,2} \vee x_{2,3}$, is actually not necessary for a valid PR proof of a pigeon hole formula, but we added it to simplify the presentation.

It remains to show that the diagonal clauses are indeed propagation redundant w.r.t. the pigeon hole formula. To do so, we show that for every assignment $\alpha = x_{i,n-1}\, x_{n,k}$ that is blocked by a diagonal clause $\overline{x}_{i,n-1} \vee \overline{x}_{n,k}$, it holds that for the assignment $\omega = \overline{x}_{i,n-1}\, \overline{x}_{n,k}\, x_{i,k}\, x_{n,n-1}$, $PHP_n|\alpha = PHP_n|\omega$, implying that $PHP_n|\alpha \vdash_1 PHP_n|\omega$. We also argue why other diagonal and unit clauses can be ignored when checking whether a new diagonal clause is propagation redundant.

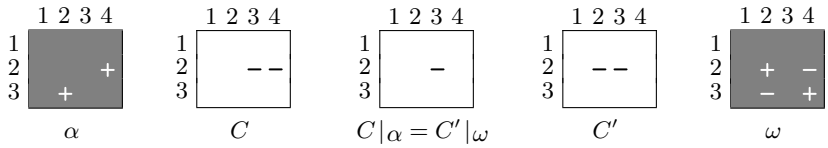
We again illustrate the idea on PHP_4. From now on, we use array notation also for assignments, i.e., a "+" ("−") in column i and row k denotes that the assignment assigns 1 (0, respectively) to variable $x_{i,k}$. Consider, for instance, the diagonal clause $D_2 = \overline{x}_{2,3} \vee \overline{x}_{4,2}$ that blocks $\alpha = x_{2,3}\, x_{4,2}$. The corresponding witnessing assignment $\omega = \overline{x}_{2,3}\, \overline{x}_{4,2}\, x_{2,2}\, x_{4,3}$ can be seen as a "rectangle" with two "−" in the corners of one diagonal and two "+" in the other corners:



To see that $PHP_4|\alpha$ and $PHP_4|\omega$ coincide on clauses $x_{i,1} \vee \cdots \vee x_{i,n-1}$, consider that whenever α and ω assign a variable of such a clause, they both satisfy the clause (since they both have a "+" in every column in which they assign a variable) and so they both remove it from PHP_4. For instance, in the following example, both α and ω satisfy $x_{2,1} \vee x_{2,2} \vee x_{2,3}$ while both do not assign a variable of the clause $x_{3,1} \vee x_{3,2} \vee x_{3,3}$:



To see that $PHP_4|\alpha$ and $PHP_4|\omega$ coincide on clauses of the form $\overline{x}_{i,k} \vee \overline{x}_{j,k}$, consider the following: If α falsifies a literal of $\overline{x}_{i,k} \vee \overline{x}_{j,k}$, then the resulting clause is a unit clause for which one of the two literals is not assigned by α (since α does not assign two variables in the same row). Now, one can show that the same unit clause is also contained in $PHP_4|\omega$, where it is obtained from another clause: Consider, for example, again the assignment $\alpha = x_{2,3}\, x_{4,2}$ and the corresponding witnessing assignment $\omega = \overline{x}_{2,3}\, \overline{x}_{4,2}\, x_{2,2}\, x_{4,3}$ from above. The assignment α turns the clause $C = \overline{x}_{3,2} \vee \overline{x}_{4,2}$ into the unit clause $C|\alpha = \overline{x}_{3,2}$. The same clause is contained in $PHP_4|\omega$, as it is obtained from $C' = \overline{x}_{2,2} \vee \overline{x}_{3,2}$ since $C'|\omega = C|\alpha = \overline{x}_{3,2}$:



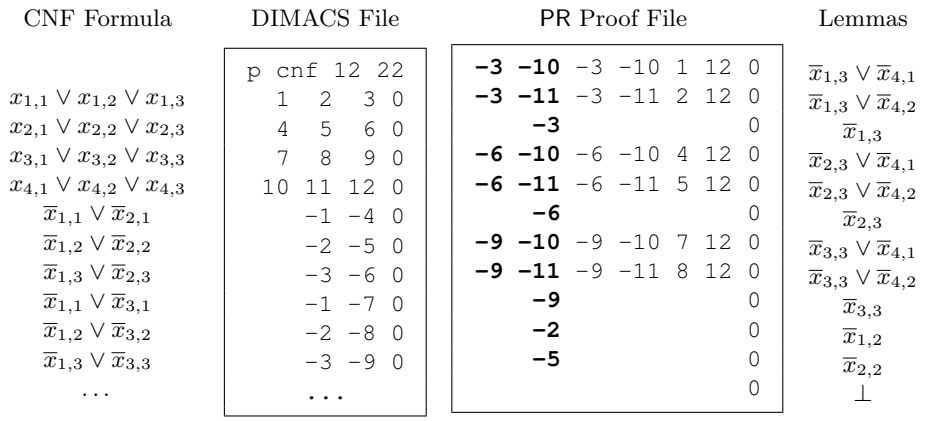


Fig. 2. Left, ten clauses of PHP_4 using the notation as elsewhere in this paper and next to it the equivalent representation of these clauses in the DIMACS format used by SAT solvers. Right, the full PR refutation consisting of clause-witness pairs. A repetition of the first literal indicates the start of the optional witness.

Note that diagonal clauses and unit clauses that have been derived earlier can be ignored when checking whether the current one is propagation redundant. For instance, assume we are currently reducing PHP_n to PHP_{n-1}. Then, the assignments α and ω under consideration only assign variables in PHP_n. In contrast, the unit and diagonal clauses used for reducing PHP_{n+1} to PHP_n (or earlier ones) are only defined on variables outside of PHP_n. They are therefore contained in both $PHP_n|\alpha$ and $PHP_n|\omega$. We can also ignore earlier unit and diagonal clauses over variables in PHP_n, i.e., clauses used for reducing an earlier column or other diagonal clauses for the current column: Whenever α assigns one of their variables, then ω satisfies them and so they are not in $PHP_n|\omega$.

Finally, we want to mention that one can also construct short SPR proofs (without new variables) of the pigeon hole formulas by first adding SPR clauses of the form $\overline{x}_{i,n-1} \vee \overline{x}_{n,k} \vee x_{i,k} \vee x_{n,n-1}$ and then turning them into diagonal clauses using resolution. We left these proofs out since they are twice as large as the PR proofs and their explanation is less intuitive. For DRAT, we consider it unlikely that such proofs exist.

6 Evaluation

We implemented a PR proof checker[4] on top of `DRAT-trim` [5]. Fig. 3 shows the pseudo code of the checking algorithm. The first "if" statement is not necessary but significantly improves the efficiency of the algorithm. The worst-case complexity of the algorithm is $\mathcal{O}(m^3)$, where m is the number of clauses in a proof. The reason for this is that there are m iterations of the outer for-loop and for

[4] The checker, benchmark formulas, and proofs are available at `http://www.cs.utexas.edu/~marijn/pr/`

```
PRcheck (CNF formula F; PR proof (C_1, ω_1), ..., (C_m, ω_m))
  for i ∈ {i, ..., m} do
    for D ∈ F do
      if D|_{ω_i} ≠ ⊤ and (D|_{α_i} = ⊤ or D|_{ω_i} ⊂ D|_{α_i}) then
        if F|_{α_i} ⊬_1 D|_{ω_i} then return failure
    F := F ∪ {C_i}
  return success
```

Fig. 3. Pseudo Code of the PR-Proof Checking Algorithm.

each of these iterations, the inner for-loop is performed $|F|$ times (i.e., once for every clause in F). Given that F contains n clauses at the start of the algorithm, we know that the size of F is bounded by $m+n$ (the original n clauses of F plus the m clauses of the proof that are added to F by the algorithm). It follows that the inner for-loop is performed $m(m+n)$ times. Now, there is a unit propagation test in the inner if-statement: If k is the maximal clause size and $m+n$ is an upper bound for the size of the formula, then the complexity of unit propagation is known to be at most $k(m+n)$. Hence, the overall worst-case complexity of the algorithm is bounded by $m(m+n)k(m+n) = \mathcal{O}(m^3)$.

This complexity is the same as for RAT-proof checking. In fact, the pseudo-code for RAT-proof checking and PR-proof checking is the same apart from the first if-statement, which is always true in the worst case, both for RAT and PR. Although the theoretical worst-case complexity makes proof checking seem very expensive, it can be done quite efficiently in practice: For the RAT proofs produced by solvers in the SAT competitions, we observed that the runtime of proof checking is close to linear with respect to the sizes of the proofs.

Moreover, we want to highlight that verifying the PR property of a clause is relatively easy as long as a witnessing assignment is given. For an arbitrary clause *without* a witnessing assignment, however, we conjecture that it is an NP-complete problem to decide whether the clause is PR. We therefore believe that in general, the verification of PR proofs is simpler than the actual solving/proving.

The format of PR proofs is an extension of DRAT proofs: the first numbers of line i denote the literals in C_i. Positive numbers refer to positive literals, and negative numbers refer to negative literals. In case a witness ω_i is provided, the first literal in the clause is repeated to denote the start of the witness. Recall that the witness always has to satisfy the clause. It is therefore guaranteed that the witness and the clause have at least one literal in common. Our format requires that such a literal occurs at the first position of the clause and of the witness. Finally, `0` marks the end of a line. Fig. 2 shows the formula and the PR proof of our running example PHP_4.

Table 1 compares our PR proofs with existing DRAT proofs of the pigeon hole formulas and of formulas from another challenging benchmark suite of the SAT competition that allow two pigeons per hole. For the latter suite, PR proofs can be constructed in a similar way as those of the classical pigeon hole formulas.

Table 1. The sizes (in terms of the number of variables and clauses) of pigeon hole formulas (top) and two-pigeons-per-hole formulas (bottom) as well as the sizes and validation times (in seconds) for their PR proofs (as described in Section 5) and their DRAT proofs (based on symmetry breaking [23]).

	input		PR proofs			DRAT proofs		
formula	#var	#cls	#var	#cls	time	#var	#cls	time
hole10.cnf	110	561	110	385	0.17	440	3,685	0.22
hole11.cnf	132	738	132	506	0.18	572	5,236	0.23
hole12.cnf	156	949	156	650	0.19	728	7,228	0.27
hole13.cnf	182	1,197	182	819	0.21	910	9,737	0.34
hole20.cnf	420	4,221	420	2,870	0.40	3,080	49,420	2.90
hole30.cnf	930	13,981	930	9,455	2.57	99,20	234,205	61.83
hole40.cnf	1,640	32,841	1,640	22,140	13.54	22,960	715,040	623.29
hole50.cnf	2,550	63,801	2,550	42,925	71.72	44,200	1,708,925	3,158.17
tph8.cnf	136	5,457	136	680	0.32	3,520	834,963	5.47
tph12.cnf	300	27,625	300	2,300	1.81	11,376	28,183,301	1,396.92
tph16.cnf	528	87,329	528	5,456	11.16	not available, too large		
tph20.cnf	820	213,241	820	10,660	61.69	not available, too large		

Notice that the PR proofs do not introduce new variables and that they contain fewer clauses than their corresponding formulas. The DRAT proof of PHP_n contains a copy of the formula PHP_k for each $k < n$. Checking PR proofs is also more efficient, as they are more compact.

7 Related Work

In this section, we shortly discuss how the concepts in this paper are related to *variable instantiation* [10], *autarkies* [8], *safe assignments* [9], and *symmetry breaking* [11]. If, for some literal l, it is possible to show $F|_{\bar{l}} \vDash F|_l$, then *variable instantiation*, as described by Andersson *et al.* [10], allows to assign the literal l in the formula F to 1. Analogously, we identify the unit clause l as redundant.

As presented by Kleine Büning and Kullmann [8], an assignment ω is an *autarky* for a formula F if it satisfies all clauses of F that contain a literal to which ω assigns a truth value. If an assignment ω is an autarky for a formula F, then F is satisfiability equivalent to $F|_\omega$. Similarly, propagation redundancy PR allows us to add all the unit clauses falsified by an autarky, with the autarky serving as a witness: Let ω be an autarky for some formula F, $C = \bar{l}$ for a literal l falsified by ω, and α the assignment blocked by C. Notice that $F|_\alpha \supseteq F|_\omega$ and thus C is propagation redundant w.r.t. F.

According to Weaver and Franco [9], an assignment ω is considered *safe* if, for every assignment α with $var(\alpha) = var(\omega)$, it holds that $F|_\alpha \vDash F|_\omega$. If an assignment ω is safe, then $F|_\omega$ is satisfiability equivalent to F. In a similar fashion, our approach allows us to block all the above-mentioned assignments $\alpha \neq \omega$. Through this, we obtain a formula that is logically equivalent to $F|_\omega$. Note that

safe assignments generalize autarkies and variable instantiation. Moreover, while safe assignments only allow the application of an assignment ω to a formula F if $F|_\alpha \vDash F|_\omega$ holds for *all* assignments $\alpha \neq \omega$, our approach enables us to block an assignment α as soon as $F|_\alpha \vDash F|_\omega$.

Finally, symmetry breaking [11] can be expressed in the DRAT proof system [23] but existing methods introduce many new variables and duplicate the input formula multiple times. It might be possible to express symmetry breaking without new variables in the PR proof system. For one important symmetry, row-interchangeability [16], the symmetry breaking using PR without new variables appears similar to the method we presented for the pigeon hole formulas.

8 Conclusion

Based on an implication relation between a formula and itself under different partial assignments, we obtain a clean and simple characterization of the most general notion of clause redundancy considered in the literature so far. Replacing the implication relation by stronger notions of implication, e.g., the superset relation or implication through unit propagation, gives then rise to various polynomially checkable redundancy criteria. One variant yields a proof system that turns out to coincide with the well-known DRAT, while we conjecture the proof systems produced by the other two variants to be much more powerful. We showed that these more general variants admit short clausal proofs for the famous pigeon hole formulas, without the need to introduce new variables. Experiments show that our proofs are much more compact than existing clausal proofs and also much faster to check. Our new proof systems simulate many other concepts from the literature very concisely, including autarkies, variable instantiation, safe assignments, and certain kinds of symmetry reasoning.

Interesting future work includes the separation of our new proof systems from the DRAT proof system on the lower end and from extended resolution on the upper end, under the additional restriction that our proof systems and DRAT do not introduce new variables. The relation to extended resolution is a particularly interesting aspect from the proof complexity point of view. Other open questions are related to the space and width bounds of the smallest PR proofs, again without new variables, for well-known other hard problems such as Tseitin formulas [12,24] or pebbling games [25]. On the practical side, we want to implement a formally verified proof checker for PR proofs. Moreover, we want to pursue some preliminary ideas for automatically generating short PR proofs during actual SAT solving: Our initial plan is to enumerate unit and binary clauses and to add them to a formula if they are propagation redundant. We already have a prototype implementation which is able to find short proofs of pigeon hole formulas, but we are still searching for efficient heuristics that help solvers with finding short PR clauses in general formulas.

References

1. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design **19**(1) (2001) 7–34
2. Ivančić, F., Yang, Z., Ganai, M.K., Gupta, A., Ashar, P.: Efficient SAT-based bounded model checking for software verification. Theoretical Computer Science **404**(3) (2008) 256–274
3. Konev, B., Lisitsa, A.: Computer-aided proof of Erdős discrepancy properties. Artificial Intelligence **224**(C) (July 2015) 103–118
4. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the Boolean Pythagorean Triples problem via Cube-and-Conquer. In: Proc. of the 19th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2016). Volume 9710 of LNCS., Cham, Springer (2016) 228–245
5. Wetzler, N.D., Heule, M.J.H., Hunt Jr., W.A.: DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In: Proc. of the 17th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2014). Volume 8561 of LNCS., Cham, Springer (2014) 422–429
6. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Proc. of the 6th Int. Joint Conference on Automated Reasoning (IJCAR 2012). Volume 7364 of LNCS., Heidelberg, Springer (2012) 355–370
7. Kiesl, B., Seidl, M., Tompits, H., Biere, A.: Super-blocked clauses. In: Proc. of the 8th Int. Joint Conference on Automated Reasoning (IJCAR 2016). Volume 9706 of LNCS., Cham, Springer (2016) 45–61
8. Kleine Büning, H., Kullmann, O.: Minimal unsatisfiability and autarkies. In Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. IOS Press (2009) 339–401
9. Weaver, S., Franco, J.V., Schlipf, J.S.: Extending existential quantification in conjunctions of BDDs. JSAT **1**(2) (2006) 89–110
10. Andersson, G., Bjesse, P., Cook, B., Hanna, Z.: A proof engine approach to solving combinational design automation problems. In: Proc. of the 39th Annual Design Automation Conference (DAC 2002), ACM (2002) 725–730
11. Crawford, J., Ginsberg, M., Luks, E., Roy, A.: Symmetry-breaking predicates for search problems. In: Proc. of the 5th Int. Conference on Principles of Knowledge Representation and Reasoning (KR 1996), Morgan Kaufmann (1996) 148–159
12. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Mathematics and Mathematical Logic **2** (1968) 115–125
13. Haken, A.: The intractability of resolution. Theoretical Computer Science **39** (1985) 297–308
14. Audemard, G., Katsirelos, G., Simon, L.: A restriction of extended resolution for clause learning sat solvers. In: Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010), AAAI Press (2010)
15. Manthey, N., Heule, M.J.H., Biere, A.: Automated reencoding of boolean formulas. In: Proc. of the 8th Int. Haifa Verification Conference (HVC 2012). Volume 7857 of LNCS., Heidelberg, Springer (2013)
16. Devriendt, J., Bogaerts, B., Bruynooghe, M., Denecker, M.: Improved static symmetry breaking for SAT. In: Proc. of the 19th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2016). Volume 9710 of LNCS., Cham, Springer (2016) 104–122
17. Balyo, T., Heule, M.J.H., Järvisalo, M.: SAT competition 2016: Recent developments. To appear in: Proc. of the 31st AAAI Conference on Artificial Intelligence (AAAI 2017), AAAI Press (2017)

18. Goldberg, E.I., Novikov, Y.: Verification of proofs of unsatisfiability for CNF formulas. In: Proc. of the Conference on Design, Automation and Test in Europe (DATE 2003), IEEE Computer Society (2003) 10886–10891
19. Van Gelder, A.: Producing and verifying extremely large propositional refutations. Annals of Mathematics and Artificial Intelligence **65**(4) (2012) 329–372
20. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics **96-97** (1999) 149–176
21. Järvisalo, M., Biere, A., Heule, M.J.H.: Simulating circuit-level simplifications on CNF. Journal on Automated Reasoning **49**(4) (2012) 583–619
22. Cook, S.A.: A short proof of the pigeon hole principle using extended resolution. SIGACT News **8**(4) (October 1976) 28–32
23. Heule, M.J.H., Hunt Jr., W.A., Wetzler, N.D.: Expressing symmetry breaking in DRAT proofs. In: Proc. of the 25th Int. Conference on Automated Deduction (CADE 2015). Volume 9195 of LNCS., Cham, Springer (2015) 591–606
24. Urquhart, A.: The complexity of propositional proofs. The Bulletin of Symbolic Logic **1**(4) (1995) 425–467
25. Nordström, J.: A simplified way of proving trade-off results for resolution. Information Processing Letters **109**(18) (August 2009) 1030–1035