Abstract We introduce proof systems for propositional logic that admit short proofs of hard formulas as well as the succinct expression of most techniques used by modern SAT solvers. Our proof systems allow the derivation of clauses that are not necessarily implied, but which are redundant in the sense that their addition preserves satisfiability. To guarantee that these added clauses are redundant, we consider various efficiently decidable redundancy criteria which we obtain by first characterizing clause redundancy in terms of a semantic implication relationship and then restricting this relationship so that it becomes decidable in polynomial time. As the restricted implication relation is based on unit propagation—a core technique of SAT solvers—it allows efficient proof checking too. The resulting proof systems are surprisingly strong, even without the introduction of new variables—a key feature of short proofs presented in the proof-complexity literature. We demonstrate the strength of our proof systems on the famous pigeon hole formulas by providing short clausal proofs without new variables.

1 Introduction

Satisfiability (SAT) solvers are used to determine the correctness of hardware and software systems [4, 16]. It is therefore crucial that these solvers justify their claims by providing proofs that can be independently verified. This holds

Disclaimer: This is the recommended version with the intended content and typesetting—in contrast to the article published by Springer.

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

1 Department of Computer Science, The University of Texas at Austin, USA
2 Institute of Logic and Computation, TU Wien, Vienna, Austria
3 CISPA Helmholtz Center i.G., Saarland Informatics Campus, Saarbrücken, Germany
4 Institute for Formal Models and Verification, Johannes Kepler University Linz, Austria
also for various other applications that use SAT solvers. Just recently, long-standing mathematical problems were solved using SAT, including the Erdős Discrepancy Problem [21], the Pythagorean Triples Problem [13], and the computation of the fifth Schur number [10]. 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—about 200 terabytes in the case of the Pythagorean Triples Problem and even two petabytes for the fifth Schur number. As the size of proofs is influenced by the strength of their underlying proof system, the search for shorter proofs goes hand in hand with the search for stronger proof systems.

In this article, we introduce highly expressive clausal proof systems that can capture most of the techniques used by modern SAT solvers. 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 to eventually add the empty clause—which is, by definition, unsatisfiable—then the unsatisfiability of the original formula has been established.

Since the redundancy of clauses is not efficiently decidable in general, 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 [30], which is the de-facto standard in practical SAT solving, only allows the addition of so-called resolution asymmetric tautologies [18]. Given a formula and a clause, it can be decided 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 checked efficiently.

We present various new redundancy criteria by introducing a characterization of clause redundancy based on a semantic implication relationship between formulas. By replacing the implication relation in this characterization with restricted notions of implication that are computable in polynomial time, we then obtain powerful redundancy criteria that are still efficiently decidable. These redundancy criteria not only generalize earlier ones such as resolution asymmetric tautologies [18] or set-blocked clauses [19], but they are also closely related to other concepts from the literature, including autarkies [24], safe assignments [29], variable instantiation [1], and symmetry breaking [6].

Proof systems based on our new redundancy criteria turn out to be highly expressive, even without allowing the introduction of new variables. This is in contrast to resolution, which is considered relatively weak as long as the introduction of new variables via definitions—as in the stronger proof system of extended resolution [26, 9]—is not allowed. The introduction of new variables, however, has a major drawback—the search space of variables and clauses we could possibly add to a proof is clearly exponential. Finding useful clauses with new variables is therefore hard in practice and resulted only in limited success in the past [2, 23].

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 clauses added in the proof contain at most two literals. In these proofs, we add redundant clauses that are similar in nature to symmetry-breaking predicates \[6, 7\]. To verify the correctness of proofs in our new system, we implemented a proof checker. The checker is built on top of DRAT-trim \[30\], the checker used to validate the unsatisfiability results of the recent SAT competitions \[3\]. 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.

This invited article is an extended version of our CADE’17 best paper \[14\]. Apart from several small improvements throughout the article, we extended the conference version by adding Section 7, which describes further interesting properties of the redundant clauses introduced in this article. We also included a new discussion of open problems in Section 9.

2 Preliminaries

We consider propositional formulas in conjunctive normal form (CNF), which are defined as follows. A literal is either a variable \(x\) (a positive literal) or the negation \(\neg x\) of a variable \(x\) (a negative literal). The complement \(\overline{l}\) of a literal \(l\) is defined as \(\overline{l} = \neg x\) if \(l = x\) and \(\overline{l} = x\) if \(l = \neg x\). For a literal \(l\), we denote the variable of \(l\) by \(\text{var}(l)\). A clause is a disjunction of literals. If not stated otherwise, we assume that clauses do not contain complementary literals, i.e., a literal and its complement. A formula is a conjunction of clauses. Clauses can be viewed 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 function from a set of variables to the truth values 1 (true) and 0 (false). An assignment is total with respect to a formula if it assigns a truth value to all variables occurring in the formula, otherwise it is partial. We often denote assignments by the sequences of literals they satisfy. For instance, \(x \overline{y}\) denotes the assignment that makes \(x\) true and \(y\) false. We denote the domain of an assignment \(\alpha\) by \(\text{var}(\alpha)\). A literal \(l\) is satisfied by an assignment \(\alpha\) if \(\alpha\) is positive and \(\alpha(\text{var}(l)) = 1\) or if it is negative and \(\alpha(\text{var}(l)) = 0\). A literal is falsified by an assignment if its complement is satisfied by the assignment. A clause is satisfied by an assignment \(\alpha\) if it contains a literal that is satisfied by \(\alpha\). Finally, a formula is satisfied by an assignment \(\alpha\) if all its clauses are satisfied by \(\alpha\). A formula is satisfiable if there exists an assignment that satisfies it. Two formulas are logically equivalent if they are satisfied by the same total assignments; they are satisfiability equivalent if they are either both satisfiable or both unsatisfiable.

We denote the empty clause by \(\bot\) and the satisfied clause by \(\top\). Given an assignment \(\alpha\) and a clause \(C\), we define \(C | _\alpha = \top\) if \(\alpha\) satisfies \(C\), otherwise \(C | _\alpha\) denotes the result of removing from \(C\) all the literals falsified by \(\alpha\). 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 $\alpha$ 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 $\alpha$ being an assignment that satisfies a unit clause 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 on a formula $F$ yields the empty clause $\bot$, we say that it derived a conflict on $F$. For example, unit propagation derives a conflict on $F = (x \lor y) \land (\overline{y}) \land (x)$ since $F|x = (y) \land (\overline{y})$ and $F|_{x=y} = \bot$.

By $F \vdash F'$, we denote that $F$ implies $F'$, i.e., every assignment that satisfies $F$ and assigns all variables in $\text{var}(F')$ also satisfies $F'$. Furthermore, by $F \vDash F'$ we denote that for every clause $(l_1 \lor \cdots \lor l_k) \in F'$, unit propagation derives a conflict on $F \land (\overline{l_1}) \land \cdots \land (\overline{l_k})$. If $F \vDash F'$, we say that $F$ implies $F'$ via unit propagation. As an example, $(x) \land (y) \vDash (x \lor z) \land (y)$, since unit propagation derives a conflict on both $(x) \land (y) \land (\overline{x}) \land (z)$ and $(x) \land (y) \land (\overline{y})$. Similarly, $F \models F'$ 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 $F \models F'$, $F \models F'$ implies $F \models F''$, and $F \models F'$ implies $F \models F''$.

3 Clause Redundancy and Clausal Proofs

In the following, 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 [19]:

**Definition 1** A clause $C$ is redundant with respect to a formula $F$ if $F$ and $F \land C$ are satisfiability equivalent.

For instance, the clause $C = x \lor y$ is redundant with respect to the formula $F = (x \lor y)$ since $F$ and $F \land C$ are satisfiability equivalent (though they are not logically equivalent). This redundancy notion allows us to add redundant clauses to a formula without affecting its satisfiability and so it provides the basis for so-called clausal proof systems.

In general, given a formula $F = \{C_1, \ldots, C_m\}$, a clausal derivation of a clause $C_\alpha$ from $F$ is a sequence $(C_{m+1}, \omega_{m+1}), \ldots, (C_n, \omega_n)$ of pairs where $C_i$ is a clause and $\omega_i$, called the witness, is a string (for all $i > m$). Such a sequence gives rise to formulas $F_{m+1}, F_{m+1}, \ldots, F_n$, where $F_i = \{C_1, \ldots, C_i\}$. We call $F_i$ the accumulated formula corresponding to the $i$-th proof step. A clausal derivation is correct if every clause $C_i$ ($i > m$) is redundant with respect to the formula $F_{i-1}$ and if this redundancy can be checked in polynomial time (with respect to the size of the proof) using the witness $\omega_i$. A clausal derivation is a (refutation) proof of a formula $F$ if it derives the empty clause, i.e., if $C_n = \bot$. Clearly, since every clause-addition step preserves satisfiability, and since the empty clause is unsatisfiable, a refutation proof of $F$ certifies the unsatisfiability of $F$.

By specifying in detail what kind of redundant clauses—and corresponding witnesses—can be added to a clausal derivation, 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** \cite{30}, 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** \cite{18} (RAT, defined in the next section). As it can be efficiently checked whether a clause is a RAT with respect to a formula, and since RATs cover many types 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 \( R_1 \) is *more general* than a redundancy criterion \( R_2 \) if, whenever \( R_2 \) identifies a clause \( C \) as redundant with respect to a formula \( F \), then \( R_1 \) also identifies \( C \) as redundant with respect to \( F \). For instance, whenever a clause is subsumed in some formula, it is a RAT with respect to 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, thus giving rise to proof systems that are stronger than **DRAT**.

### 4 Clause Redundancy via Implication

In the following, we introduce a characterization of clause redundancy that reduces the question whether a clause is redundant with respect to a certain formula to a simple question of implication. The advantage of this is that we can replace the logical implication relation by 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 that falsify 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 \( F = (x \lor y) \land (x \lor z) \land (\exists y \lor z) \) and consider the (unit) clause \( C = x \) which blocks all assignments that falsify \( x \). The addition of \( C \) to \( F \) does not affect satisfiability: Let \( \alpha = x \) and \( \omega = x \). Then, \( F|_{\alpha} = (y) \land (z) \) while \( F|_{\omega} = (y \lor z) \). 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 \( \omega \) as it is under \( \alpha \). Moreover, \( \omega \) satisfies \( C \). The addition of \( C \) does therefore not affect the satisfiability of \( F \). \( \square \)

This motivates our new characterization of clause redundancy presented next. The characterization requires the existence of an assignment that satisfies the clause and so it is only applicable to non-empty clauses. Note that for a given
Theorem 1 Let \( F \) be a formula, \( C \) a non-empty clause, and \( \alpha \) the assignment blocked by \( C \). Then, \( C \) is redundant with respect to \( F \) if and only if there exists an assignment \( \omega \) such that \( \omega \) satisfies \( C \) and \( F|\alpha \models F|\omega \).

Proof For the “only if” direction, assume that \( F \) and \( F \land C \) are satisfiability equivalent. If \( F|\alpha \) is unsatisfiable, then \( F|\alpha \models F|\omega \) for every \( \omega \), hence the statement trivially holds. Assume now that \( F|\alpha \) is satisfiable, implying that \( F \) and \( F \land C \) are satisfiability equivalent, there exists an assignment \( \omega \) that satisfies both \( F \) and \( C \). Thus, since \( \omega \) satisfies \( F \), it holds that \( F|\omega = \emptyset \) and so \( F|\alpha \equiv F|\omega \).

For the “if” direction, assume that there exists an assignment \( \omega \) such that \( \omega \) satisfies \( C \) and \( F|\alpha \models F|\omega \). Now, let \( \gamma \) be a (total) assignment that satisfies \( F \) and falsifies \( C \). We show how \( \gamma \) can be turned into a satisfying assignment \( \gamma' \) of \( F \land C \). As \( \gamma \) falsifies \( C \), it coincides with \( \alpha \) on \( \text{var}(\alpha) \). Therefore, since \( \gamma \) satisfies \( F \), it must satisfy \( F|\alpha \) and since \( F|\alpha \models F|\omega \) it must also satisfy \( F|\omega \). Now, consider the following assignment:

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

Clearly, since \( \omega \) satisfies \( C \), \( \gamma' \) also satisfies \( C \). Moreover, as \( \gamma \) satisfies \( F|\omega \) and \( \text{var}(F|\omega) \subseteq \text{var}(\gamma) \setminus \text{var}(\omega) \), \( \gamma' \) satisfies \( F \). Hence, \( \gamma' \) satisfies \( F \land C \). □

This alternative characterization of redundancy allows us to replace the logical implication relation by restricted implication relations that are polynomially decidable. For instance, we can replace the condition \( F|\alpha \models F|\omega \) by the restricted condition \( F|\alpha \models F|\omega \) (likewise, we could also use relations such as “\( \models \)” or “\( \supseteq \)” instead of “\( \models \)”). Now, if we are given a clause \( C \)—which implicitly gives us the blocked assignment \( \alpha \)—and a witnessing assignment \( \omega \), then we can check in polynomial time whether \( F|\alpha \models F|\omega \), which is a sufficient condition for the redundancy of \( C \) with respect to \( F \). We can therefore use this implication-based redundancy notion to define proof systems. The witnessing assignments can then be used as witnesses in the proof.

In the following, we use the propagation-implication relation “\( \models \)” to define the redundancy criteria of

- **LPR**: literal-propagation redundancy,
- **SPR**: set-propagation redundancy, and
- **PR**: propagation redundancy.

Basically, the three notions differ in the way we allow the witnessing assignment \( \omega \) to differ from the assignment \( \alpha \) blocked by a clause. The more freedom we give to \( \omega \), 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 (SBC) \[19\], 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. 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 \[8, 28\], S stands for subsumed clauses, RS for clauses with subsumed resolvents \[18\], and BC for blocked clauses \[22, 17\].

As we will see, when defining proof systems based on LPR (e.g., the DRAT system) or SPR clauses, we do not need to explicitly add the redundancy witnesses (i.e., the witnessing assignments \(\omega\)) to a proof. Thus, LPR and SPR proofs can just be seen as a sequence of clauses. In particular, a proof system based on SPR clauses can have the same syntax as DRAT proofs, which makes it “downwards compatible”. This is in contrast to proof systems based on PR clauses, where in general witnessing assignments have to be added to a proof. Otherwise redundancy of a clause can not be checked in polynomial time.

We start by introducing LPR clauses. In the following, given a (partial) assignment \(\alpha\) and a set \(L\) of literals, we denote by \(\alpha_L\) the assignment obtained from \(\alpha\) by making all literals in \(L\) true. If \(L\) contains only a single literal, we sometimes write \(\alpha_l\) to denote \(\alpha\{l\}\). In the conference paper \[14\], we used a slightly different definition, saying that \(\alpha_L\) is obtained from \(\alpha\) by flipping the truth values of all literals in \(L\). Since we only defined \(\alpha_L\) for assignments \(\alpha\) that falsify all the literals in \(L\), nothing changes. We do, however, believe that the new notion is more intuitive.
**Definition 2** Let $F$ be a formula, $C$ a clause, and $\alpha$ the assignment blocked by $C$. Then, $C$ is literal-propagation redundant (LPR) with respect to $F$ if there exists a literal $l \in C$ such that $F|_\alpha \vdash F|_{\alpha l}$.

**Example 2** Let $F = (x \lor y) \land (x \lor \neg y) \land (\neg x \lor z)$ and let $C$ be the unit clause $x$. Then, $\alpha = \pi$ is the assignment blocked by $C$, and $\alpha x = x$. Now, consider $F|\alpha = (y) \land (\neg y \lor z)$ and $F|\alpha x = (z)$. Clearly, $F|\alpha \vdash F|\alpha x$ and therefore $C$ is literal-propagation redundant with respect to $F$. □

The LPR definition is quite restrictive since it requires the witnessing assignment $\alpha l$ to disagree with $\alpha$ on exactly one variable. Nevertheless, this already suffices for LPR clauses to coincide with RATs [13].

**Definition 3** Let $F$ be a formula and $C$ a clause. Then, $C$ is a resolution asymmetric tautology (RAT) with respect to $F$ if there exists a literal $l \in C$ such that, for every clause $D \in F_T$, $F \vdash C \cup (D \setminus \{l\})$.

**Theorem 2** A clause $C$ is LPR with respect to a formula $F$ if and only if it is a RAT with respect to $F$.

**Proof** For the “only if” direction, assume that $C$ is LPR with respect to $F$, i.e., $C$ contains a literal $l$ such that $F|_\alpha \vdash F|_{\alpha l}$. Now, let $D \in F_T$. We have to show that $F|_\alpha C \cup (D \setminus \{l\})$. 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 $\alpha l$ falsifies $\overline{I}$, it follows that $D|_{\alpha l} \subseteq (D \setminus \{l\})$. But then, since $F|_{\alpha l} \vdash D|_{\alpha l}$, it must hold that $F|_\alpha C \cup (D \setminus \{l\})$, hence $C$ is a RAT with respect to $F$.

For the “if” direction, assume that $C$ is a RAT with respect to $F$, i.e., $C$ contains a literal $l$ such that, for every clause $D \in F_T$, $F|_{\alpha l} C \cup (D \setminus \{l\})$. Now, let $D|_{\alpha l} \in F|_{\alpha l}$ for $D \in F$. We have to show that $F|_{\alpha l} \vdash D|_{\alpha l}$. Since $\alpha l$ satisfies $l$ and $\alpha$ falsifies $C$, $D$ does neither contain $l$ nor any negations of literals in $C$ except for possibly $\overline{l}$. If $D$ does not contain $\overline{l}$, then $D|_{\alpha} = D|_{\alpha l}$ as contained in $F|_{\alpha}$ and hence the claim immediately follows.

Assume therefore that $\overline{l} \in D$. As argued for the other direction, propagating the negated literals of $C$ (and no other literals) on $F$ yields $F|_{\alpha l}$. Therefore, since $F|_{\alpha l} C \cup (D \setminus \{l\})$ and $D \setminus \{l\}$ 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 $D \setminus \{l\}$), it must be the case that $F|_{\alpha l} D \setminus \{l\}$. Now, the only literals of $D \setminus \{l\}$ that are not contained in $D|_{\alpha l}$ are the ones falsified by $\alpha$, but those are anyhow not contained in $F|_{\alpha}$. Hence, $F|_{\alpha l} \vdash D|_{\alpha l}$ and thus $C$ is LPR with respect to $F$. □

By allowing the witnessing assignments to disagree with $\alpha$ on more than one literal, we obtain the more general notion of set-propagation-redundant clauses, which we introduce next. In the following, for a set $L$ of literals, we define $\overline{L} = \{\overline{l} \mid l \in L\}$.
Definition 4 Let $F$ be a formula, $C$ a clause, and $\alpha$ the assignment blocked by $C$. Then, $C$ is set-propagation redundant (SPR) with respect to $F$ if there exists a non-empty set $L \subseteq C$ of literals such that $F|_\alpha \models F|_{\alpha_L}$.

Example 3 Let $F = (x \lor y) \land (x \lor \neg y \lor z) \land (\neg y \lor u) \land (\neg u \lor x)$, $C = x \lor u$, and $L = \{x, u\}$. Then, $\alpha = \neg y \land \neg u$ is the assignment blocked by $C$, and $\alpha_L = x \lor u$. Now, consider $F|_\alpha = (y) \land (\neg y \lor z)$ and $F|_{\alpha_L} = (z)$. Clearly, $F|_\alpha \not\models F|_{\alpha_L}$ and so $C$ is set-propagation redundant with respect to $F$. Note also that $C$ is not literal-propagation redundant with respect to $F$. \qed

Since $L$ is a subset of $C$, we do not need to add it (or the assignment $\alpha_L$) 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 $l_1 \lor \cdots \lor l_k$, 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 $L = \{l_1, \ldots, l_k\}$. 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 [19]:

Definition 5 A clause $C$ is set-blocked (SBC) by a non-empty set $L \subseteq C$ in a formula $F$ if, for every clause $D \in F_L$, the clause $(C \setminus L) \cup \bar{L} \cup D$ 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 $\alpha$ 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 $\alpha$ and $\alpha_L$ disagree only on literals in $L$, it follows that $D$ contains a literal $l \in L$ and thus $D \in F_L$. Now, $\alpha_L$ falsifies exactly the literals in $(C \setminus L) \cup \bar{L}$ 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 $\bar{l}$ is contained in $(C \setminus L) \cup \bar{L}$. Therefore, $C$ is not SBC by $L$ in $F$.

For the “if” direction, assume that $C$ is not SBC by $L$ in $F$, i.e., there exists a clause $D \in F_L$ such that $(C \setminus L) \cup \bar{L} \cup D$ does not contain complementary literals. Now, $D|_\alpha = \top$ since $\alpha$ falsifies $L$ and $D \cap L \neq \emptyset$. Since $D$ contains no literal $l$ such that $\bar{l} \in (C \setminus L) \cup \bar{L}$ and since $\alpha_L$ falsifies exactly the literals in $(C \setminus L) \cup \bar{L}$, it follows that $\alpha_L$ does not satisfy $D$, hence $D|_{\alpha_L} \neq \top$. \qed

Theorem 4 If a clause $C$ is set-blocked by a set $L$ in a formula $F$, it is set-propagation redundant with respect to $F$.

Proof Assume that $C$ is set-blocked by $L$ in $F$. We show that $F|_\alpha \supseteq F|_{\alpha_L}$, which implies that $F|_\alpha \models F|_{\alpha_L}$, and therefore that $C$ is set-propagation
redundant with respect to \( F \). 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 \( D|\alpha_L \not\in F|\alpha_L \). Second, observe that \( D \) can also not be contained in \( F_L \), since that would imply that \( D|\alpha = \top \) and thus, by Lemma 3, \( D|\alpha_L = \top \). Therefore, \( D \not\in F_L \cup F_L \) 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. As there are resolution asymmetric tautologies that are not set-blocked (and vice versa) \([19]\), it follows that set-propagation-redundant clauses are actually a strict generalization of these two kinds of clauses.

Note that \( F|\alpha \models F|\alpha_L \) is equivalent to \( F|\alpha_L \models F|\alpha \). To see this, observe that if a clause \( D|\alpha_L \in F|\alpha_L \) contains no literals from \( L \), then \( \alpha_L \) does not assign any of its literals, in which case \( D|\alpha_L \) is also contained in \( F|\alpha \). We therefore do not need to check for every \( D|\alpha_L \in F|\alpha_L \) whether \( F|\alpha \models D|\alpha \).

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 6** Let \( F \) be a formula, \( C \) a clause, and \( \alpha \) the assignment blocked by \( C \). Then, \( C \) is propagation redundant (PR) with respect to \( F \) if there exists an assignment \( \omega \) such that \( \omega \) satisfies \( C \) and \( F|\alpha \models F|\omega \).

**Example 4** Let \( F = (x \lor y) \land (\overline{x} \lor y) \land (\overline{x} \lor z) \), \( C = x \), and let \( \omega = x \leftrightarrow z \) 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, \( F|\alpha \models F|\omega \) and so \( C \) is PR with respect to \( F \). 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 \( F|\alpha \models z \). The fact that \( \omega \) satisfies \( z \) is crucial for ensuring propagation redundancy.

Since the witnessing assignments \( \omega \) 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.

We can now explicitly define the PR proof system as an instance of a clausal proof system as defined on page 4.

**Definition 7** Given a formula \( F = \{C_1, \ldots, C_m\} \), a PR derivation of a clause \( C_n \) from \( F \) is a sequence \( (C_{m+1}, \omega_{m+1}), \ldots, (C_n, \omega_n) \) where for every pair \( (C_i, \omega_i) \), one of the following holds: (1) \( \omega_i \) is an assignment that satisfies \( C_i \) and \( F_{i-1}|\alpha_i \models F_{i-1}|\omega_i \) with \( \alpha_i \) being the assignment blocked by \( C_i \), or (2) \( C_n = \bot \) and \( F_{n-1} \models \bot \). A PR derivation of \( \bot \) from \( F \) is a PR proof of \( F \).

The LPR proof system and the SPR proof system are defined accordingly. Note that in the definition above we treat the empty clause separately because only non-empty clauses can be propagation redundant. If we allow the mentioned proof systems to delete arbitrary clauses, we obtain the proof systems DLPR, DSPR, and DPR. We will not consider deletion in the rest of the article.
5 Short Proofs of the Pigeon Hole Principle

In a landmark article, Haken [9] showed that pigeon hole formulas cannot be refuted by resolution proofs that are of polynomial size with respect to the size of the formulas. In contrast, Cook [5] proved that there are actually polynomial-size refutations of the pigeon hole formulas in the stronger proof system of extended resolution. What distinguishes extended resolution from general resolution is that it allows 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 exponential.

In the following, we illustrate how the PR proof system admits short proofs of pigeon hole formulas without the need for introducing new variables. This shows that the PR system is strictly stronger than the resolution calculus, even when we forbid the introduction of new variables. A pigeon hole formula \( PHP_n \) intuitively encodes that \( n + 1 \) pigeons have to be assigned to \( n \) holes such that no hole contains more than one pigeon. In the encoding, a variable \( x_{p,h} \) intuitively denotes that pigeon \( p \) is assigned to hole \( h \):

\[
PHP_n := \bigwedge_{1 \leq p \leq n+1} (x_{p,1} \lor \cdots \lor x_{p,n}) \land \bigwedge_{1 \leq p < q \leq n+1} \bigwedge_{1 \leq h \leq n} (x_{p,h} \lor x_{q,h})
\]

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 \( x_{p,1} \lor \cdots \lor x_{p,n-1} \) for \( 1 \leq p \leq n \).

Following Haken [9], we use array notation for clauses: Every clause is represented by an array of \( n + 1 \) columns and \( n \) rows. An array contains a “+” (“−”) in the \( p \)-th column and \( h \)-th row if and only if the variable \( x_{p,h} \) occurs positively (negatively, respectively) in the corresponding clause. Representing \( PHP_n \) in array notation, we have for every clause \( x_{p,1} \lor \cdots \lor x_{p,n} \), an array in which the \( p \)-th column is filled with “+”. Moreover, for every clause \( x_{p,h} \lor x_{q,h} \), we have an array that contains two “−” in row \( h \)—one in column \( p \) and the other in column \( q \). For instance, \( PHP_3 \) is given in array notation as follows:

\[
\begin{array}{cccc}
1 & + & + & + \\
2 & + & + & + \\
3 & + & + & + \\
\end{array}
\quad
\begin{array}{cccc}
1 & + & + & + \\
2 & + & + & + \\
3 & + & + & + \\
\end{array}
\quad
\begin{array}{cccc}
1 & + & + & + \\
2 & + & + & + \\
3 & + & + & + \\
\end{array}
\quad
\begin{array}{cccc}
1 & + & + & + \\
2 & + & + & + \\
3 & + & + & + \\
\end{array}
\quad
\begin{array}{cccc}
1 & + & + & + \\
2 & + & + & + \\
3 & + & + & + \\
\end{array}
\quad
\begin{array}{cccc}
1 & + & + & + \\
2 & + & + & + \\
3 & + & + & + \\
\end{array}
\]

\[\text{1} \quad \text{2} \quad \text{3} \quad \text{4} \quad \text{1} \quad \text{2} \quad \text{3} \quad \text{4} \quad \text{1} \quad \text{2} \quad \text{3} \quad \text{4} \quad \text{1} \quad \text{2} \quad \text{3} \quad \text{4} \quad \text{1} \quad \text{2} \quad \text{3} \quad \text{4}
\]

We changed the definition of \( PHP_n \) from putting \( n \) pigeons into \( n - 1 \) holes to putting \( n + 1 \) pigeons into \( n \) holes in order to fix a discrepancy with the formulas in the evaluation.
We illustrate the general idea for reducing a pigeon hole formula $PHP_n$ to the smaller $PHP_{n-1}$ on the concrete formula $PHP_3$. It should, however, become clear from our explanation that the procedure works for every $n > 1$. If we want to reduce $PHP_3$ to $PHP_2$, we have to derive the following three clauses:

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & + & 1 \\
3 & + & + & 1 \\
\end{array}
\]

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 $PHP_2$. The key observation is that a “+” in the last row of the $p$-th column can be removed with the help of so-called “diagonal clauses” of the form $\forall p,n \leq h \neg x_{p,n} \lor \forall p,n+1 \neg x_{p,n}$, or, in other words, how they help us to resolve with $PHP$. We illustrate the general idea for reducing a pigeon hole formula $PHP$ as follows: 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 $\pi_{2,3}$. The clauses are given in array notation as follows:

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & - & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & + & 1 \\
3 & + & + & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & - & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

Finally, we resolve $\pi_{2,3}$ with $x_{2,1} \lor x_{2,2} \lor x_{2,3}$ to obtain the desired clause $x_{2,1} \lor x_{2,2}$. 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 $\pi_{2,3}$. The clauses are given in array notation as follows:

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & - & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & + & - & 1 \\
3 & - & - & 1 \\
\end{array}
\]
We can then resolve $\tau_{2,3}$ with $x_{2,1} \lor x_{2,2} \lor x_{2,3}$ to obtain $x_{2,1} \lor x_{2,2}$:

This should illustrate how a clause of the form $x_{p,1} \lor \cdots \lor x_{p,n}$ ($1 \leq p \leq n$) can be reduced to a clause $x_{p,1} \lor \cdots \lor x_{p,n-1}$. By repeating this procedure for every column $p$ with $1 \leq p \leq n$, 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 $\tau_{2,3}$ with the clause $x_{2,1} \lor x_{2,2} \lor x_{2,3}$, 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 with respect to the pigeon hole formula. To do so, we show that for every assignment $\alpha = x_{p,n} x_{n+1,h}$ that is blocked by a diagonal clause $\tau_{p,n} \lor \tau_{n+1,h}$, it holds that for the assignment $\omega = \tau_{p,n} \tau_{n+1,h} x_{p,h} x_{n+1,n}$, $PHP_n | \alpha = PHP_n | \omega$, implying that $PHP_n | \alpha \nvdash PHP_n | \omega$. 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 $PHP_3$. We now use array notation also for assignments, i.e., a “$+$” (“$-$”) in column $p$ and row $h$ denotes that the assignment makes variable $x_{p,h}$ true (false, respectively). Consider, for instance, the diagonal clause $D_2 = \tau_{2,3} \lor \tau_{4,2}$ that blocks $\alpha = x_{2,3} x_{4,2}$. The corresponding witnessing assignment $\omega = \tau_{2,3} \tau_{4,2} x_{2,2} x_{4,3}$ can be seen as a “rectangle” with two “$-$” in the corners of one diagonal and two “$+$” in the other corners:

To see that $PHP_3 | \alpha$ and $PHP_3 | \omega$ coincide on clauses $x_{p,1} \lor \cdots \lor x_{p,n}$, consider that whenever $\alpha$ and $\omega$ 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 $PHP_3$. For instance, in the following example, both $\alpha$ and $\omega$ satisfy $x_{2,1} \lor x_{2,2} \lor x_{2,3}$ while both do not assign a variable of the clause $x_{3,1} \lor x_{3,2} \lor x_{3,3}$:

To see that $PHP_3 | \alpha$ and $PHP_3 | \omega$ coincide on clauses of the form $\tau_{p,h} \lor \tau_{q,h}$, consider the following: If $\alpha$ falsifies a literal of $\tau_{p,h} \lor \tau_{q,h}$, then the resulting clause is a unit clause for which one of the two literals is not assigned by $\alpha$.
(since \( \alpha \) does not assign two variables in the same row). Now, one can show that the same unit clause is also contained in \( \text{PHP}_3 | \omega \), 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 \( \omega = x_{2,3} x_{4,2} x_{2,4} x_{4,3} \) from above.

The assignment \( \alpha \) turns the clause \( C = x_{3,2} \vee x_{4,2} \) into the unit \( C | \alpha = x_{4,2} \). The same clause is contained in \( \text{PHP}_3 | \omega \), as it is obtained from \( C' = x_{2,2} \vee x_{3,2} \) since \( C' | \omega = C | \alpha = x_{3,2} \):

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
\alpha & + & 1 & 2 & 3 & 4 \\
\hline
C & -- & 1 & 2 & 3 & 4 \\
\hline
C | \alpha = C' | \omega & - & 1 & 2 & 3 & 4 \\
\hline
C' & -- & 1 & 2 & 3 & 4 \\
\hline
\omega & - & 1 & 2 & 3 & 4 \\
\end{array}
\]

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 \( \text{PHP}_n \) to \( \text{PHP}_{n-1} \). Then, the assignments \( \alpha \) and \( \omega \) under consideration only assign variables in \( \text{PHP}_n \). In contrast, the unit and diagonal clauses used for reducing \( \text{PHP}_{n+1} \) to \( \text{PHP}_n \) (or earlier ones) are only defined on variables outside of \( \text{PHP}_n \). They are therefore contained in both \( \text{PHP}_n | \alpha \) and \( \text{PHP}_n | \omega \). We can also ignore earlier unit and diagonal clauses over variables in \( \text{PHP}_n \), i.e., clauses used for reducing an earlier column or other diagonal clauses for the current column: If \( \alpha \) assigns one of their variables, then \( \omega \) satisfies them and so they are not in \( \text{PHP}_n | \omega \).

Finally, we want to mention that short \( \text{SPR} \) proofs (without new variables) of the pigeon hole formulas can be constructed by first adding \( \text{SPR} \) clauses of the form \( x_{p,n} \vee x_{n+1,h} \vee x_{p,h} \vee x_{n+1,n} \) and then turning them into diagonal clauses using resolution. We left these proofs out since they are twice as large as the \( \text{PR} \) proofs and their explanation is less intuitive. A recent result shows that the conversion of a \( \text{PR} \) proof into a \( \text{DRAT} \) proof requires only one auxiliary variable \[11\]. We can thus construct short \( \text{DRAT} \) proofs without new variables from short \( \text{PR} \) proofs without new variables. To do so, we first eliminate a single variable from the original formula and then reuse that variable in the conversion algorithm. This is only possible because \( \text{DRAT} \) allows clause deletion. We consider it unlikely that there exist short proofs for the pigeon hole formulas in the \( \text{RAT/LPR} \) proof system, where no deletions are allowed.

6 Evaluation

We implemented a \( \text{PR} \) proof checker on top of \( \text{DRAT-trim} \) \[30\]. The tool, formulas, and proofs are available at [https://www.cs.utexas.edu/~marijn/pr](https://www.cs.utexas.edu/~marijn/pr). Fig. 2 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 \( O(n^3) \), where \( n \) is the size of the final formula. The reason for this is that there are \( n - m \) iterations of the outer for-loop and for each of these iterations, the inner for-loop is performed \( |F_i| \) times, i.e., once for every clause in \( F_i \). Given that \( F_i \) contains \( i \) clauses,
we know that the size of \( F \) is bounded by \( n \). It follows that the inner for-loop is performed \( mn \) times. Now, there is a unit propagation test in the inner if-statement: If \( k \) is the maximal clause size and \( n \) is an upper bound for the size of the formula, then the complexity of unit propagation is known to be at most \( kn \). Hence, the overall worst-case complexity of the algorithm is bounded by \( mkn^2 = O(n^3) \).

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, it is an NP-complete problem to decide if 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 \( \omega_i \) 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. 3 shows the formula and the PR proof of our running example PHP.

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 allows 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. Notice that the PR proofs do not introduce new variables and that they contain fewer clauses than their corresponding formulas. The DRAT

\[
\text{PRcheck (formula } F_m = C_1, \ldots, C_m; \text{ PR proof } (C_{m+1}, \omega_{m+1}), \ldots, (C_n, \omega_n) )
\]

\[
\begin{align*}
\text{for } i \in \{m + 1, \ldots, n\} \text{ do} & \\
\quad \text{for } D \in F_{i-1} \text{ do} & \\
\quad \quad \text{if } D \mid \omega_i \neq \top \text{ and } (D \mid \alpha_i = \top \text{ or } D \mid \omega_i \subset D \mid \alpha_i) \text{ then} & \\
\quad \quad \quad \text{if } F_{i-1} \mid \alpha_i \not\models D \mid \omega_i \text{ then return failure} & \\
\quad \quad \text{F}_i := F_{i-1} \cup \{C_i\} & \\
\text{return success} & 
\end{align*}
\]

Fig. 2 Pseudo Code of the PR-Proof Checking Algorithm.
CNF Formula | DIMACS File | PR Proof File | Lemmas
---|---|---|---
$x_{1,1} \lor x_{1,2} \lor x_{1,3}$ | p cnf 12 22 | -3 -10 -3 -10 1 12 0 | $\tau_{1,3} \lor \tau_{4,1}$
$x_{2,1} \lor x_{2,2} \lor x_{2,3}$ | | -3 -11 -3 -11 2 12 0 | $\tau_{1,3} \lor \tau_{4,2}$
$x_{3,1} \lor x_{3,2} \lor x_{3,3}$ | | -3 | $\tau_{1,3}$
$x_{4,1} \lor x_{4,2} \lor x_{4,3}$ | | -6 -10 -6 -10 4 12 0 | $\tau_{2,3} \lor \tau_{4,1}$
| | -6 | $\tau_{3,3}$
$\tau_{1,1} \lor \tau_{2,1}$ | | -9 -10 -9 -10 7 12 0 | $\tau_{1,3} \lor \tau_{4,1}$
$\tau_{1,2} \lor \tau_{2,2}$ | | -9 -11 -9 -11 8 12 0 | $\tau_{3,3} \lor \tau_{4,2}$
$\tau_{1,3} \lor \tau_{2,3}$ | | -9 | $\tau_{1,3}$
$\tau_{1,1} \lor \tau_{3,1}$ | | -2 -5 0 | $\tau_{1,2}$
$\tau_{1,2} \lor \tau_{3,2}$ | | -5 | $ot$
$\tau_{1,3} \lor \tau_{3,3}$ | | -5 | $\tau_{1,2}$
... | ... | ... | ...
this, we show how propagation-redundant clauses can be shortened based on unit propagation. Finally, we present an observation that clarifies the relationship between propagation-redundant clauses and their corresponding witnessing assignments.

Our strategy for showing that the superclauses of LPR, SPR, and PR clauses are also LPR, SPR, and PR is as follows: Assume, for instance, that \( C \) is a PR clause with respect to \( F \) and let \( \alpha \) be the assignment blocked by \( C \). We then know that there exists an assignment \( \omega \) such that \( F \models_\alpha \models F \models_\omega \). To show that every superclause \( C' \) of \( C \) is also propagation redundant with respect to \( F \), we extend \( \alpha \) so that it becomes the assignment \( \alpha' \) blocked by \( C' \). After this, we extend \( \omega \) to an assignment \( \omega' \) such that \( F \models_\alpha' \models F \models_\omega' \). The following example shows that we cannot simply extend \( \alpha \) without extending \( \omega \):

**Example 5** Let \( F = (x \lor y) \land (\pi \lor y) \) and let \( \alpha = x \) and \( \omega = \pi \). Then, \( F \models_\alpha \) and \( F \models_\omega \) both contain only the unit clause \( y \) and so it holds that \( F \models_\alpha \models F \models_\omega \). If we only extend \( \alpha \) to \( \alpha y \), then \( F \models_\alpha y \models F \models_\omega \). However, if we also extend \( \omega \) to \( \omega y \), then we again have \( F \models_\alpha y \models F \models_\omega y \). \( \square \)

In the following, we present several statements that help us extend \( \omega \) in the right way. We start with a simple observation about the relation between unit propagation and variable assignments. If a formula is unsatisfiable, then the formula remains unsatisfiable after we assign some of its variables. A similar property holds when unsatisfiability can be shown by unit propagation: If unit propagation derives a conflict on a formula, then we can assign truth values to arbitrary variables of the formula and unit propagation will still derive a conflict. This property will be useful below when we prove other properties about propagation redundancy:

**Proposition 5** If unit propagation derives a conflict on a formula \( F \), then it derives a conflict on \( F \models x \) for every literal \( x \).

**Proof** Assume unit propagation derives a conflict on \( F \). Then, there must exist a sequence \( C_1, \ldots, C_k \) of clauses from \( F \) such that \( C_1 \) is the unit clause \( a_1 \), \( C_2 | a_1 \) is the unit clause \( a_2 \), and so on until \( C_{k-1} | a_1 \ldots a_{n-2} = a_{k-1} \), and finally \( C_k | a_1 \ldots a_{k-1} = \bot \). Now, if the variable \( \text{var}(x) \) does not occur in any of the clauses \( C_1, \ldots, C_k \), then \( C_i \models x = C_i \) for each \( i \in 1, \ldots, n \) and thus unit propagation derives a conflict on \( F \models x \). Assume now that \( \text{var}(x) \) occurs in \( C_1, \ldots, C_k \) and let \( C_i \) be the clause with the smallest \( i \) such that \( \text{var}(x) \in C_i \). Then, \( C_1, \ldots, C_{i-1} \in F \models x \) and so unit propagation derives the unit clauses \( a_1, \ldots, a_{i-1} \) on \( F \models x \). We proceed by a case distinction.

\( x \in C_i \): In this case, \( C_i \notin F \models x \), but we know that \( C_i | a_1 \ldots a_{i-1} = a_i = x \) since \( \text{var}(x) \) cannot occur in \( a_1, \ldots, a_{i-1} \). But then the assignment \( a_1 \ldots a_{i-1}a_i \), derived by unit propagation on \( F \), is the assignment \( a_1 \ldots a_{i-1}x \), derived by unit propagation on \( F \models x \). Hence, unit propagation on \( F \models x \) derives a conflict using the clauses \( C_1 \models x, \ldots, C_{i-1} \models x, C_{i+1} \models x, \ldots, C_k \models x \).

\( \pi \in C_i \): In that case, \( C_i | a_1 \ldots a_{i-1} \) must be the unit clause \( \pi \). It follows that \( C_i \subseteq \pi \lor \cdots \lor \pi_{i-1} \lor \pi \) and thus \( C_i \models x \subseteq \pi \lor \cdots \lor \pi_{i-1} \lor \pi \). Hence, unit propagation
on $F|_x$ derives a conflict with the clauses $C_1, C_2, \ldots, C_i|_x$ since it derives all the unit clauses $a_1, \ldots, a_{i-1}$. \hfill \Box

In Example 5, we presented a formula $F$ with two assignments $\alpha$ and $\omega$ such that $F|_\alpha \vdash F|_\omega$. After extending $\alpha$ to $\alpha x$, however, we could observe that $F|_{\alpha x} \not\vdash F|_\omega$. The problem in the example is that in contrast to $F|_\alpha$, the formula $F|_{\alpha x}$ does not contain the unit clause $x$ anymore while $F|_\omega$ still contains $x$ as a clause. Hence, $F|_{\alpha x}$ does not imply $F|_\omega$. However, as the next statement tells us, it is guaranteed that $F|_{\alpha x}$ implies all those clauses of $F|_\omega$ that contain neither $x$ nor $\overline{x}$.

In the rest of this section, given a clause $C = (c_1 \lor \cdots c_a)$, we write $\neg C$ for the conjunction $\neg \overline{c_1} \land \cdots \land \neg \overline{c_a}$ of unit clauses. In the following statement, the requirement that $\alpha$ must not falsify $x$ makes sure that $\alpha x$ is well-defined:

**Lemma 6** Let $F$ be formula, let $\alpha, \omega$ be assignments such that $F|_\alpha \vdash F|_\omega$, and let $x$ be a literal that is not falsified by $\alpha$. Then, $F|_{\alpha x} \vdash D|_\omega$ for every clause $D|_\omega \in F|_\omega$ such that $\text{var}(x) \notin \text{var}(D|_\omega)$.

**Proof** Assume that $F|_\alpha \vdash F|_\omega$ and let $D|_\omega \in F|_\omega$ be a clause such that $\text{var}(x) \notin \text{var}(D|_\omega)$. Since $F|_\alpha \vdash D|_\omega$, we know that unit propagation derives a conflict on $F|_\alpha \land \neg(D|_\omega)$, with $\neg(D|_\omega)$ being the conjunction of the negated literals of $D|_\omega$. Since $\text{var}(x) \notin \text{var}(D|_\omega)$, it follows that $D|_\omega = D|_{\omega x}$. Thus, $(F|_{\alpha x} \land \neg(D|_\omega))|_x = F|_{\alpha x} \land \neg(D|_\omega)$. But then, since unit propagation derives a conflict on $F|_{\alpha x} \land \neg(D|_\omega)$, we know, by Proposition 5, that unit propagation derives a conflict on $F|_{\alpha x} \land \neg(D|_\omega)$. It follows that $F|_{\alpha x} \vdash D|_\omega$. \hfill \Box

Using Lemma 6, we can show that every literal that is neither falsified by $\alpha$ nor by $\omega$ can just be appended to both $\alpha$ and $\omega$:

**Lemma 7** Let $F$ be formula, $\alpha$ and $\omega$ assignments, and $x$ a literal that is neither falsified by $\alpha$ nor by $\omega$. Then, $F|_{\alpha x} \vdash F|_{\omega x}$.

**Proof** Suppose $F|_{\alpha x} \vdash F|_{\omega x}$ and let $D|_{\omega x} \in F|_{\omega x}$ for $D \in F$. We show that $F|_{\alpha x} \vdash D|_{\omega x}$. If $\overline{x} \in D|_\omega$, then $D|_\omega = D|_{\omega x} \lor \overline{x}$. Hence, unit propagation derives a conflict on $F|_{\alpha x} \land \neg(D|_{\omega x}) \lor x$, with $\neg(D|_{\omega x})$ being the conjunction of the negated literals of $D|_{\omega x}$. But then unit propagation derives a conflict on $F|_{\alpha x} \land \neg(D|_{\omega x})$ and thus $F|_{\alpha x} \vdash D|_{\omega x}$. If $\overline{x} \notin D|_\omega$, then $D|_{\omega x} = D|_\omega$. Now, we know that $F|_{\alpha x} \vdash D|_\omega$ and hence $F|_{\alpha x} \vdash D|_{\omega x}$. Thus, by Lemma 6, it follows that $F|_{\alpha x} \vdash D|_{\omega x}$. \hfill \Box

Assume that a clause $C$ is LPR with respect to a formula $F$. Let $D$ be a super-clause of $C$, $\alpha$ be the assignment blocked by $C$, and $\alpha x_1 \ldots x_k$ the assignment blocked by $D$. Then, we know that there exists a literal $l \in C$ such that $F|_\alpha \vdash F|_{\alpha l}$. But then Lemma 7 tells us that $F|_{\alpha x_1 \ldots x_k} \vdash F|_{\alpha x_1 \ldots x_k}$ and thus $D$ is LPR with respect to $F$. The same argument applies to SPR clauses (but not to PR clauses in general) and thus we get:

**Theorem 8** If a clause $C$ is LPR (SPR) with respect to $F$, then every super-clause of $C$ is LPR (SPR, respectively) with respect to $F$. 
To show that the corresponding statement also holds for PR clauses that are not SPR clauses, we need to show some additional properties of PR clauses. The next statement, which is a simple consequence of Lemma 6, tells us that the extension of $\alpha$ to $\alpha x$ is harmless if $\omega$ already falsifies $x$:

**Lemma 9** Let $F$ be formula, let $\alpha$ and $\omega$ be assignments, and let $x$ be a literal that is not falsified by $\alpha$. Then, $F|\alpha \models F|\omega \tau$ implies $F|\alpha x \models F|\omega \tau$.

**Proof** Assume that $\text{var}(x)$ occurs in a clause $D \in F$. If $x \in D$, then $D|\omega \tau$ does not contain $x$. If $\tau \in D$, then $D$ is satisfied by $\omega \tau$ and thus $D|\omega \tau \notin F|\omega \tau$. Thus, by Lemma 6, $F|\alpha x \models F|\omega \tau$. \qed

Putting everything together, we can now show that we can always extend $\alpha$ if we just extend $\omega$ accordingly:

**Lemma 10** Let $F$ be a formula, let $\alpha$ and $\omega$ be two assignments such that $F|\alpha \models F|\omega$, and let $\alpha'$ be an assignment such that $\alpha \subseteq \alpha'$. Then, there exists an assignment $\omega'$ such that $\omega \subseteq \omega'$ and $F|\alpha' \models F|\omega'$.

**Proof** Suppose $\alpha'$ is an assignment such that $\alpha \subseteq \alpha'$. Then, $\alpha'$ is of the form $\alpha x_1 \ldots x_k$ where $n \in 0, \ldots, n$. Now, starting with $x_1$, we step-wise extend $\alpha$ with $x_1, \ldots, x_k$ to finally obtain $\alpha x_1 \ldots x_k$. We just have to extend $\omega$ to an assignment $\omega_k$ accordingly to ensure that $F|\alpha x_1 \ldots x_k \models F|\omega_k$. We start with $\omega_0 = \omega$ and proceed as follows for $i \in 1, \ldots, n$: If $\text{var}(x_i) \in \text{var}(\omega)$, define $\omega_i = \omega_{i-1}$. In contrast, if $\text{var}(x_i) \notin \text{var}(\omega)$, define $\omega_i = \omega_{i-1} x_i$.

By a simple induction on $i$ we can now show $F|\alpha x_1 \ldots x_i \models F|\omega_i$ for every $i \in 0, \ldots, k$: The base case, $F|\alpha \models F|\omega$, holds by assumption. The induction hypothesis states that $F|\alpha x_1 \ldots x_{i-1} \models F|\omega_{i-1}$. Now, for the induction step, if $\omega(x_i) = 0$, then $\omega_i$ is of the form $\omega_{i-1} \tau_i$. In this case, by Lemma 9, $F|\alpha x_1 \ldots x_i \models F|\omega_{i-1} \tau_i$. If $\text{var}(x_i) \notin \text{var}(\omega)$ or $\omega(x_i) = 1$, then $\omega_i$ is of the form $\omega_{i-1} x_i$. In that case, by Lemma 7, $F|\alpha x_1 \ldots x_i \models F|\omega_{i-1} x_i$. \qed

As immediate consequence we obtain one of our main statements:

**Theorem 11** If a clause $C$ is PR with respect to $F$, then every superclause of $C$ is PR with respect to $F$.

Next, we show that we can remove certain literals from propagation-redundant clauses without violating their property of being propagation redundant. The idea is as follows: Let $C$ be a clause that is propagation redundant with respect to a formula $F$ and let $\alpha x_1 \ldots x_k$ be the assignment blocked by $C$. Now, if unit propagation on $F|\alpha$ derives the unit clauses $x_1, \ldots, x_k$, then the clause that blocks only $\alpha$—and not the whole assignment $\alpha x_1 \ldots x_k$—is propagation redundant with respect to $F$ too. To show this, we first introduce the notion of propagation extensions. Note that in the following definition, by a *consistent* set of unit clauses, we mean a set of unit clauses that does not contain two complementary unit clauses $x$ and $\bar{x}$.
Definition 8 Let $F$ be a formula, let $\alpha$ be an assignment, and let $\{x_1, \ldots, x_k\}$ be a consistent set of unit clauses derived by unit propagation on $F|_\alpha$. Then, $\alpha x_1 \ldots x_k$ is a propagation extension of $\alpha$ on $F$.

Example 6 Let $F = (x \lor y) \land (\neg y \lor z)$ and let $\alpha = \neg z$. Unit propagation on $F|_\alpha$ derives the unit clauses $y$ and $z$. Hence, the assignments $\alpha y$, $\alpha z$, and $\alpha y z$ are propagation extensions of $\alpha$ on $F$. Now consider the formula $F \land \neg z$. Then, unit propagation on $(F \land \neg z)|_\alpha$ derives the unit clauses $y$, $z$, and $\neg z$. Thus, the propagation extensions of $\alpha$ on $F \land \neg z$ are the assignments $\alpha z$ and $\alpha y z$ as well as all the propagation extensions of $\alpha$ on $F$.  

If $F|_{\alpha^+} \models F|_\omega$ for some propagation extension $\alpha^+$ of an assignment $\alpha$, we can simply shorten $\alpha^+$ to $\alpha$ without modifying $\omega$ and it will still hold that $F|_{\alpha} \models F|_\omega$.

Lemma 12 Let $F$ be a formula, $\alpha$ an assignment, and $\alpha^+$ a propagation extension of $\alpha$ on $F$. Then, $F|_{\alpha} \models F|_\omega$ if and only if $F|_{\alpha^+} \models F|_\omega$.

Proof The “only if” direction is an immediate consequence of Lemma 10. For the “if” direction, assume that $F|_{\alpha^+} \models F|_\omega$ and let $D|_\omega \in F|_\omega$. We know that unit propagation derives a conflict on $F|_{\alpha^+} \land \neg (D|_\omega)$ where $\neg (D|_\omega)$ is the conjunction of the negated literals of $D|_\omega$. Since unit propagation derives $F|_{\alpha^+}$ from $F|_{\alpha}$, it follows that unit propagation derives a conflict on $F|_{\alpha} \land \neg (D|_\omega)$. Hence, $F|_{\alpha} \models D|_\omega$ and thus $F|_{\alpha} \models F|_\omega$. \hfill $\square$

Using Lemma 12 we can now show that the removal of propagated literals from PR clauses is harmless:

Theorem 13 Let $C$ be a clause that is PR with respect to a formula $F$ and let $\alpha^+$ be the assignment blocked by $C$. If $\alpha^+$ is a propagation extension of an assignment $\alpha$ on $F$, then the clause that blocks $\alpha$ is PR with respect to $F$.

Proof Assume that $\alpha^+$ is a propagation extension of an assignment $\alpha$ on $F$ and let $C^-$ be the clause that blocks $\alpha$. Then, $\alpha^+$ is of the form $\alpha x_1 \ldots x_k$ where $x_1, \ldots, x_k$ are all the literals derived by unit propagation on $F|_\alpha$. Since $C$ is propagation redundant with respect to $F$, we know that there exists some assignment $\omega$ such that $\omega$ satisfies $C$ and $F|_{\alpha x_1 \ldots x_k} \models F|_\omega$. Hence, by Lemma 12 $F|_{\alpha} \models F|_\omega$. Now, if $\omega$ satisfies $C^-$, then $C^-$ is propagation redundant with respect to $F$.

Assume thus that $\omega$ does not satisfy $C^-$. Then, since $\omega$ satisfies $C$, it must falsify a literal in $x_1, \ldots, x_k$. Let $x_i$ be the first literal (i.e., the one with the smallest index) of $x_1, \ldots, x_k$ that is falsified by $\omega$. Then, there exists a clause $D \in F$ such that $D|_{\alpha x_1 \ldots x_{i-1}}$ is the unit clause $x_i$ and thus $\omega$ falsifies $D$. Hence, $F|_{\alpha} \models \bot$. But then, $C^-$ is trivially PR with respect to $F$ since it holds for every assignment $\tau$ (and in particular for every $\tau$ that satisfies $C$) that $F|_{\alpha} \models F|_\tau$. \hfill $\square$
We can prove a corresponding result about LPR clauses: If we remove propagated literals from an LPR clause, then the resulting clause is also an LPR clause. As we will see later, this is not the case for SPR clauses. We start with two lemmas:

**Lemma 14** Let $F$ be a formula, $\alpha$ and $\omega$ assignments, and $x$ a literal such that $F|\alpha \models x$. Then, $F|\alpha \models F|\omega x$ implies $F|\alpha \models F|\omega$.

**Proof** Suppose to the contrary that there exists a clause $D|\omega \in F|\omega$ such that $F|\alpha \not\models D|\omega$. Then, $D$ must contain $x$, for otherwise $\omega x$ would not satisfy $D$, which would in turn imply $F|\alpha \not\models F|\omega x$. Therefore, the clause $\neg(D|\omega)$, being the conjunction of the negated literals of $D|\omega$, must contain $\overline{x}$. But then, since unit propagation on $F|\alpha$ derives $x$, it follows that unit propagation derives a conflict on $F|\alpha \land \neg(D|\omega)$. Hence, $F|\alpha \not\models D|\omega$ and thus $F|\alpha \not\models F|\omega$. \qed

**Lemma 15** Let $F$ be a formula, $\alpha$ an assignment, $\alpha^+$ a propagation extension of $\alpha$ on $F$, and $l$ a literal. Then, $F|\alpha^+ \models F|\alpha^+_l$ implies $F|\alpha \models F|\alpha_l$.

**Proof** Assume that $\alpha^+$ is a propagation extension of $\alpha$. Then, $\alpha^+$ is of the form $\alpha \tau$ where $\tau$ make the literals true that have been derived by unit propagation on $F|\alpha$. To show that $F|\alpha \models F|\alpha_l$, we distinguish two cases:

$\var(l) \in \var(\alpha)$: In this case, $F|\alpha \tau \models F|\alpha^+_l$ and thus $F|\alpha \models F|\alpha^+_l$. Now, since $F|\alpha \models x$ for every literal $x$ satisfied by $\tau$, we use Lemma 14 to repeatedly remove from $\alpha^+_l$ all assignments made by $\tau$ to obtain $F|\alpha \models F|\alpha_l$.

$\var(l) \in \var(\tau)$: In that case, $F|\alpha \tau \models F|\alpha^+_l$. Since unit propagation on $F|\alpha$ derives $l$, there exists a clause $D \in F$ such that $\alpha \tau_l$—which satisfies $l$—falsifies $D$. Hence, $D|\alpha^+_l = \bot$, and since $F|\alpha \tau \models D|\alpha^+_l$, it follows that unit propagation derives a conflict on $F|\alpha \tau$. But then unit propagation must derive a conflict on $F|\alpha$ and thus $F|\alpha$ implies every clause via unit propagation. We thus conclude that $F|\alpha \models F|\alpha_l$. \qed

The following Theorem is now an immediate consequence of Lemma 15:

**Theorem 16** Let $C$ be a clause that is LPR with respect to a formula $F$ and let $\alpha^+$ be the assignment blocked by $C$. If $\alpha^+$ is a propagation extension of an assignment $\alpha$ on $F$, then the clause that blocks $\alpha$ is LPR with respect to $F$.

The corresponding property does not hold for SPR clauses, as illustrated by the following example:

**Example 7** Let $F = (\pi \lor y) \land (x \lor \overline{y})$, $C = x \lor y$, and $L = \{x, y\}$. Then, $\alpha^+ = \pi \overline{y}$, which is the assignment blocked by $C$, is a propagation extension of the assignment $\alpha = \pi$. Moreover, since $F|\alpha^+_L$ contains no clauses, we have $F|\alpha^+_L \models F|\alpha^+_L_y$ and thus $C$ is set-propagation redundant with respect to $F$. However, $F|\alpha \not\models F|\alpha_x$ and thus the subclause $x$ of $C$ is not set-propagation redundant with respect to $F$. \qed
We conclude this section with an observation about the witnessing assignments of propagation-redundant clauses. In the case of propagation redundancy, the domain of the witnessing assignment $\omega$ is not constrained to a particular set of variables. Thus, if we are given a clause $C$ and we want to find a corresponding assignment $\omega$ that witnesses the propagation redundancy of $C$, we would have to consider assignments over all possible sets of variables. It turns out, however, that there is always a witnessing assignment that assigns all variables occurring in $C$, and possibly more. Thus, if $\alpha$ is the assignment blocked by $C$, we only need to consider assignments $\omega$ such that $\text{var}(\alpha) \subseteq \text{var}(\omega)$. The reason for this is that we can extend every witness $\omega$ to the variables of $\text{var}(\alpha)$:

**Proposition 17** Let $F$ be a formula, $\alpha$ and $\omega$ assignments, and $x$ a literal such that $\text{var}(x) \in \text{var}(\alpha) \setminus \text{var}(\omega)$. Then, $F|_\alpha \models F|_\omega$ implies $F|_\alpha \models F|_{\omega x}$.

*Proof* Let $D|_{\omega x} \in F|_{\omega x}$. We show that $F|_\alpha \models D|_{\omega x}$. Clearly, $x$ is not contained in $D$ for otherwise $D|_{\omega x} = \top$. Therefore, the only possible difference between $D|_{\omega}$ and $D|_{\omega x}$ is that $\pi$ is contained in $D|_{\omega}$ but not in $D|_{\omega x}$. Now, since $\text{var}(x) \in \text{var}(\alpha)$, we know that $\text{var}(x) \notin F|_\alpha$. But then, $F|_\alpha \models D|_{\omega x}$ if and only if $F|_\alpha \models D|_{\omega}$. It thus follows that $F|_\alpha \models F|_{\omega x}$. \hfill $\square$

### 8 Related Work

Here, we discuss how the concepts in this article are related to *variable instantiation* [1], *autarkies* [24], *safe assignments* [29], and *symmetry breaking* [6]. If $F|_\pi \models F|_x$ holds for some literal $x$, then *variable instantiation*, as described by Andersson *et al.* [1], allows to make the literal $x$ true in the formula $F$. Analogously, our redundancy notion identifies the clause $x$ as redundant.

As presented by Monien and Speckenmeyer [24], an assignment $\omega$ is an *autarky* for a formula $F$ if it satisfies all clauses of $F$ that contain a literal to which $\omega$ assigns a truth value. If an assignment $\omega$ 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 satisfied by an autarky, with the autarky serving as a witness of $\omega$. Let $\omega$ be an autarky for some formula $F$, $C = x$ for a literal $x$ satisfied by $\omega$, and $\alpha = \pi$ the assignment blocked by $C$. Notice that $F|_\alpha \models F|_{\omega}$ and thus $C$ is PR with respect to $F$.

According to Weaver and Franco [29], an assignment $\omega$ is considered *safe* if, for every assignment $\alpha$ with $\text{var}(\alpha) = \text{var}(\omega)$, it holds that $F|_\alpha \models F|_{\omega}$. If an assignment $\omega$ 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 $\omega$ to a formula $F$ if $F|_\alpha \models F|_{\omega}$ holds for all assignments $\alpha \neq \omega$, our approach enables us to block an assignment $\alpha$ as soon as $F|_\alpha \models F|_{\omega}$.

---

2 In the conference version [14] of this article, we wrongly stated that we can add all *falsified* unit clauses instead of the *satisfied* ones.
Finally, symmetry breaking \cite{6} can be expressed in the DRAT proof system \cite{12} 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 \cite{7}, the symmetry breaking using PR without new variables appears similar to the method we presented for the pigeon hole formulas.

9 Open Problems and Future Work

In a more recent paper, we showed that there exists a polynomial-time procedure that translates PR proofs to DRAT proofs by introducing one new variable \cite{11}. Moreover, we proved that extended resolution polynomially simulates the DRAT proof system \cite{20}. The combination of these two results demonstrates that extended resolution polynomially simulates the PR proof system and therefore also its restricted variants. An open question is how the PR proof system without new variables relates to other strong proof systems for propositional logic that do not introduce new variables, such as Frege systems. 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 \cite{26, 27} or pebbling games \cite{25}.

On the practical side, we want to pursue some ideas to improve SAT solving by learning short PR clauses. Our first approach, called satisfaction-driven clause learning, generalizes the well-known conflict-driven clause learning paradigm by checking whether certain assignments—encountered during solving—can be pruned from the search space by adding PR clauses \cite{15}. Our current implementation can find short proofs of pigeon hole formulas, although the solver can only find a subset of all possible PR clauses. Moreover, we are still searching for efficient heuristics that help solvers with finding short PR clauses in general formulas. Another problem we are currently exploring is the minimization of conflict clauses by checking if a subset of a conflict clause is propagation redundant with respect to the formula under consideration. Finally, we want to implement a formally-verified proof checker for PR proofs.

10 Conclusion

We presented a clean and simple characterization of clause redundancy that is based on an implication relation between a formula and itself under different partial assignments. Replacing the implication relation by efficiently decidable notions of implication, e.g., the superset relation or implication via unit propagation, gives then rise to various polynomially-checkable redundancy criteria. One variant yields a proof system that turns out to coincide with RAT, which together with deletion is the de-facto standard in SAT solving. We conjecture the proof systems based on the other two variants to be stronger if the introduction of new variables is not allowed. We showed that these more general
proof systems admit short clausal proofs without new variables for the famous pigeon hole formulas. Experiments show that our proofs are much smaller than existing clausal proofs and that they are also much faster to check. Our new proof systems concisely simulate many other concepts from the literature such as autarkies, variable instantiation, safe assignments, and certain kinds of symmetry reasoning.

References

1. Andersson G, B Jesse P, Cook B, Hanna Z (2002) A proof engine approach to solving combinational design automation problems. In: Proc. of the 39th Annual Design Automation Conference (DAC 2002), ACM, pp 725–730
2. Audemard G, Katsirelos G, Simon L (2010) A restriction of extended resolution for clause learning sat solvers. In: Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010), AAAI Press
3. Balyo T, Heule MJH, Järvisalo M (2017) SAT competition 2016: Recent developments. To appear in: Proc. of the 31st AAAI Conference on Artificial Intelligence (AAAI 2017), AAAI Press
4. Clarke EM, Biere A, Raimi R, Zhu Y (2001) Bounded model checking using satisfiability solving. Formal Methods in System Design 19(1):7–34
5. Cook SA (1976) A short proof of the pigeon hole principle using extended resolution. SIGACT News 8(4):28–32
6. Crawford J, Ginsberg M, Luks E, Roy A (1996) Symmetry-breaking predicates for search problems. In: Proc. of the 5th Int. Conference on Principles of Knowledge Representation and Reasoning (KR 1996), Morgan Kaufmann, pp 148–159
7. Devriendt J, Bogaerts B, Bruynooghe M, Denecker M (2016) Improved static symmetry breaking for SAT. In: Proc. of the 19th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2016), Springer, Cham, LNCS, vol 9710, pp 104–122
8. Goldberg EI, Novikov Y (2003) 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, pp 10,886–10,891
9. Haken A (1985) The intractability of resolution. Theoretical Computer Science 39:297–308
10. Heule MJH (2018) Schur number five. In: Proc. of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018), AAAI Press, pp 6598–6606
11. Heule MJH, Biere A (2018) What a difference a variable makes. In: Proc. of the 24th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2018), Springer, Cham, LNCS, vol 10806, pp 75–92
12. Heule MJH, Hunt Jr WA, Wetzler ND (2015) Expressing symmetry breaking in DRAT proofs. In: Proc. of the 25th Int. Conference on Automated Deduction (CADE-25), Springer, Cham, LNCS, vol 9195, pp 591–606
13. Heule MJH, Kullmann O, Marek VW (2016) 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), Springer, Cham, LNCS, vol 9710, pp 228–245
14. Heule MJH, Kiesl B, Biere A (2017) Short proofs without new variables.
In: Proc. of the 26th Int. Conference on Automated Deduction (CADE-
26), Springer, Cham, LNCS, vol 10395, pp 130–147
15. Heule MJH, Kiesl B, Seidl M, Biere A (2017) PRuning through satis-
faction. In: Proc. of the 13th Haifa Verification Conference (HVC 2017),
Springer, Cham, LNCS, vol 10629, pp 179–194
16. Ivančič F, Yang Z, Ganai MK, Gupta A, Ashar P (2008) Efficient SAT-
based bounded model checking for software verification. Theoretical Com-
puter Science 404(3):256–274
17. Järvisalo M, Biere A, Heule MJH (2012) Simulating circuit-level simplifi-
cations on CNF. Journal on Automated Reasoning 49(4):583–619
18. Järvisalo M, Heule MJH, Biere A (2012) Inprocessing rules. In: Proc. of
the 6th Int. Joint Conference on Automated Reasoning (IJCAR 2012),
Springer, Heidelberg, LNCS, vol 7364, pp 355–370
19. Kiesl B, Seidl M, Tompits H, Biere A (2016) Super-blocked clauses.
In: Proc. of the 8th Int. Joint Conference on Automated Reasoning
(IJCAR 2016), Springer, Cham, LNCS, vol 9706, pp 45–61
20. Kiesl B, Rebola-Pardo A, Heule MJH (2018) Extended resolution simu-
lates DRAT. In: Proc. of the 9th Int. Joint Conference on Automated
Reasoning (IJCAR 2018), Springer, Cham, LNCS, vol 10900, pp 516–531
21. Konev B, Lisitsa A (2015) Computer-aided proof of Erdős discrepancy
properties. Artificial Intelligence 224(C):103–118
22. Kullmann O (1999) On a generalization of extended resolution. Discrete
Applied Mathematics 96-97:149–176
23. Manthey N, Heule MJH, Biere A (2013) Automated reencoding of
boolean formulas. In: Proc. of the 8th Int. Haifa Verification Conference
(HVC 2012), Springer, Heidelberg, LNCS, vol 7857
24. Monien B, Speckenmeyer E (1985) Solving satisfiability in less than 2
n
steps. Discrete Applied Mathematics 10(3):287 – 295
25. Nordström J (2009) A simplified way of proving trade-off results for reso-
lution. Information Processing Letters 109(18):1030–1035
26. Tseitin GS (1968) On the complexity of derivation in propositional calcu-
lus. Studies in Mathematics and Mathematical Logic 2:115–125
27. Urquhart A (1995) The complexity of propositional proofs. The Bulletin
of Symbolic Logic 1(4):425–467
28. Van Gelder A (2012) Producing and verifying extremely large proposi-
tional refutations. Annals of Mathematics and Artificial Intelligence
65(4):329–372
29. Weaver S, Franco JV, Schlipf JS (2006) Extending existential quantifica-
tion in conjunctions of BDDs. JSAT 1(2):89–110
30. Wetzler ND, Heule MJH, Hunt Jr WA (2014) 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), Springer, Cham, LNCS, vol 8561, pp 422–429