Even Shorter Proofs Without New Variables

Adrián Rebola-Pardo
Technische Universität Wien, Austria
Johannes Kepler Universität Linz, Austria

Abstract

Proof formats for SAT solvers have diversified over the last decade, enabling new features such as extended resolution-like capabilities, very general extension-free rules, inclusion of proof hints, and pseudo-boolean reasoning. Interference-based methods have been proven effective, and some theoretical work has been undertaken to better explain their limits and semantics. In this work, we combine the subsumption redundancy notion from [9] and the overwrite logic framework from [42]. Natural generalizations then become apparent, enabling even shorter proofs of the pigeonhole principle (compared to those from [27]) and smaller unsatisfiable core generation.

2012 ACM Subject Classification Hardware → Theorem proving and SAT solving

Keywords and phrases Interference, SAT solving, Unsatisfiability proofs, Unsatisfiable cores

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.22

Funding This work has been supported by the LIT AI Lab funded by the State of Upper Austria, the LogiCS doctoral program W1255-N23 of the Austrian Science Fund (FWF), the Vienna Science and Technology Fund (WWTF) through grants VRG11-005 and ICT15-103, and Microsoft Research through its PhD Scholarship Programme.

Acknowledgements I would like to thank Georg Weissenbacher for the discussions on WSR, as well as the anonymous reviewers who provided very useful comments.

1 Introduction

The impressive recent improvements in SAT solving have come coupled with the need to ascertain their results. While satisfiability results are straightforward to check, unsatisfiability results require massive proofs, sometimes petabytes in size [28, 24]. The search for proof systems that enable both easy proof generation and smaller proofs has yield many achievements [17, 15, 53, 27, 41, 3, 9, 16, 4].

Modern proof systems rely on redundancy properties presenting a phenomenon known as *interference* [31, 23, 42]. Whereas traditional proof systems derive clauses that are implied by the premises, interference-based proof systems merely require introduced clauses to be consistent with them. Interference proofs preserve the existence of a model throughout the proof, rather than models themselves. A somewhat counterintuitive semantics thus arises: introducing a clause in an interference-based proof system does not only depend on the presence of some clauses, but also on the absence of some other clauses [39, 42].

The most general interference-based proof system in the literature is known as DSR [9]. While its predecesor DPR had success in generating short proofs of the pigeonhole formula without introducing new variables [27], DSR did not seem to succeeded in improving this result, despite being intuitively well-suited for it.

In this work, we analyze the semantics of DSR proofs extending previous work on DPR proofs [42]. We find similar results to that article; in particular, satisfiability-preserving DSR proofs can be reinterpreted as more traditional, DAG-shaped, model-preserving proofs over an extension of propositional logic with a *mutation* operator. Crucially, these DAG-shaped proofs remove the whole-formula dependence interference is characterized by, enabling an easier analysis of the necessary conditions for satisfiability-preservation.

This analysis hints at a generalization we call *weak substition redundancy* (WSR ['wɪzəʳ]), which allows shorter, more understandable, easier to generate, faster to check proofs. We demonstrate this by giving an even shorter proof of the pigeonhole formula. We also provide a couple of examples where smaller unsatisfiable cores can be generated during proof checking, and fewer lemmas are required during proof generation.

Interference-based proofs

Much of proof generation and checking is still done in the same way as a couple decades ago, by logging the sequence of *learnt clauses* in CDCL checkers, sometimes together with antecedents, and checking those clauses for simple entailment criteria such as *reverse unit propagation* (RUP) [17, 54]. Other parts of the proof are generated using more advanced deduction techniques; even their infrequent use can dramatically decrease the size of generated proofs [20, 53, 32, 26], overcoming not only technical limitations in proof generation, but also theoretical bounds [18, 51, 52]. Clause deletion information is also recorded in the proof, which is needed to reduce memory footprint in checking [21].

Much research has been invested on finding ever more powerful proof rules [31, 27, 9] that allow to succintly express inprocessing techniques such as Gaussian elimination [48, 47, 38, 10, 16] or symmetry breaking [1, 2, 22]. These proof rules are collectively called *interference-based rules*, since their derivation depends on the whole formula rather than just on the presence of some specific clauses [31, 23, 39, 42]. One of the most general interference techniques is *substitution redundancy* (SR), which allows a version of reasoning without loss of generality [9]; this technique has been recently lifted to pseudo-Boolean reasoning with impressive results [16].

Substitution redundancy and the pigeonhole problem

A previous version of SR, called *propagation redundancy* (PR) [27], was successful in achieving short proofs of the pigeonhole problem, known for having exponential proofs in resolution [18] and polynomial yet cumbersome proofs in extended resolution [11]. The proof from [27] can be understood in terms of reasoning without loss of generality [42]: it assumes that a given pigeon is in a given pigeonhole, for otherwise we could swap pigeons around.

PR does not have a method to swap the values of variables; rather, it can only conditionally set them to true or false. Hence, linearly many reasoning steps are needed to just to achieve the swap. SR, on the other hand, allows variable swaps, so one could expect that the clause expressing the result of this swap would satisfy the SR property. Surprisingly, it does not; in fact, the clause fails to satisfy a requirement that in its PR version was almost trivial.

Interference and logical dependency

Interference-based proofs do not have a "dependence" or "procedence" structure: since the ability to introduce a clause is contingent on the whole formula, no notion of "antecedents" exists for SR and its predecessors. This becomes a problem when computing unsatisfiable cores and trimmed proofs [37]; it also has the potential to harm the performance of proof checkers, since some techniques that allow skipping unnecessary steps during proof checking are based on logical dependence [19].

This also relates to an issue arising when generating proof fragments for inprocessing techniques. Sometimes, a clause C cannot be introduced as SR because some lemmas are needed; the proof generator might know these lemmas and how to derive them. However, because interference depends on the whole formula, introducing the lemmas before C can further constrain the requirements for C to be introduced, demanding yet more lemmas.

Contributions

Previous work showed that the semantics of PR can be expressed in terms of *overwrite logic* [42]. Overwrite logic extends propositional logic with an *overwrite operator*. Within overwrite logic, DPR proofs can be regarded as DAG-shaped, model-preserving proofs; PR introduction can then be shown to behave as reasoning without loss of generality.

In Section 3 we provide an extension to the overwrite logic framework, called *mutation logic*, which elucidates the semantics of DSR proofs. In particular, model-preserving proofs within mutation logic mimicking satisfiability-preserving DSR proofs can be extracted, as shown in Section 3.1. This allows a clearer understanding of the SR redundancy rule, which in turn makes some improvements over SR apparent.

By introducing minor modifications to the definition of SR, in Section 4 we obtain a new, more powerful redundancy rule called *weak substitution redundancy* (WSR). WSR proofs are more succint than DSR proofs, which we demonstrate by providing a shorter proof of the pigeonhole problem using only $O(n^2)$ clause introductions in Section 5.1.

Furthermore, WSR enables finer-grained ways to reason about dependency in interference-based proofs. This can yield shorter proof checking runtimes and smaller trimmed proofs and unsatisfiability cores when SR clauses are used (Section 5.2), as well as easier proof generation techniques by providing clearer separation for interference lemmas (Section 5.3).

2 Preliminaries

Given a *literal* l, we denote its *complement* as $\bar{l}$. We denote *clauses* by juxtaposing its literals within square brackets, i.e. we denote the clause $l_1 \vee l_2 \vee l_3$ as $[l_1 l_2 l_3]$. We similarly denote conjunctions of literals, called *cubes*, as juxtaposed literals within angle brackets, e.g. $\langle l_1 l_2 l_3 \rangle$. Crucially, we only consider clauses and cubes that do not contain complementary literals, as most SAT solvers and proof checkers already make that assumption. Equivalently, we disallow tautological clauses and unsatisfiable cubes. We also define complementation for clauses and cubes, i.e. $\overline{[l_1 \dots l_n]} = \langle \overline{l_1} \dots \overline{l_n} \rangle$ and $\overline{\langle l_1 \dots l_n \rangle} = [\overline{l_1} \dots \overline{l_n}]$. SAT solving typically operates over formulas in *conjunctive normal form* (CNF), which are conjunctions of clauses. Here we regard CNF formulas as finite sets of clauses.

An *atom* is either a literal, or one of the symbols $\top$ or $\bot$ representing the propositional constants true and false. Complementation is extended to atoms with $\overline{\top} = \bot$ and $\overline{\bot} = \top$. We can then define the usual propositional semantics as follows. A *model* I is a total map from atoms to $\{\top, \bot\}$ such that $I(\top) = \top$ and $\overline{I(l)} = I(\bar{l})$ for all atoms l.

We say that I *satisfies* a literal l (written $I \models l$) whenever $I(l) = \top$. This definition is recursively extended in the usual way to clauses (disjunctively), cubes and CNF formulas (conjunctively). Similarly, we use the typical notions of *entailment* (denoted $\models$), logical *equivalence* ($\equiv$) and *satisfiability*. We also say that a logical expression φ *satisfiability-entails* another expression ψ (denoted $\varphi \models_{\text{sat}} \psi$) whenever, if φ is satisfiable, then ψ is satisfiable too. Similarly, φ is *satisfiability-equivalent* to ψ (denoted $\varphi \equiv_{\text{sat}} \psi$) whenever φ and ψ are either both satisfiable or both unsatisfiable (i.e. $\varphi \models_{\text{sat}} \psi$ and $\psi \models_{\text{sat}} \varphi$).

An *atomic substitution* σ is a total map from atoms to atoms satisfying the following constraints:

- **(i)** $\sigma(\top) = \top$.
- **(ii)** $\overline{\sigma(l)} = \sigma(\bar{l})$ for all atoms l.
- **(iii)** $\sigma(l) \neq l$ only for finitely many atoms l.

This definition is essentially equivalent to the substitutions from [9]. The form presented here makes it easier to compose atomic substitutions with other atomic substitutions, i.e. $(\sigma \circ \tau)(l) = \sigma(\tau(l))$, and with models, i.e. $(I \circ \sigma)(l) = I(\sigma(l))$; the latter is a model that satisfies a given logical expression φ iff I satisfies the expression resulting from applying the substitution σ to φ.

Note that atomic substitutions have a finite representation: only finitely many literals are mapped to atoms other than themselves, and giving the mapping for one polarity fixes the mapping for the other polarity. Hence, one can represent a substitution as a set of mappings $\{x_1 \mapsto l_1, \ldots, x_n \mapsto l_n\}$ where the x_i are pairwise distinct variables, the l_i are atoms, and any variable other than the x_i is mapped to itself.

Our restriction that clauses must be non-tautological is somewhat at odds with the concept of substitutions. An atomic substitution σ *trivializes* a clause C if either:

(a) there is a literal $l \in C$ with $\sigma(l) = \top$.

(b) there are two literals $l, k \in C$ with $\sigma(l) = \overline{\sigma(k)}$.

Applying σ to C yields a tautology whenever σ trivializes C, and a (non-tautological) clause otherwise. Then we can define the *reduct* of a clause C or a CNF formula F by an atomic substitution σ as:

$$C\big|_\sigma = [\sigma(l) \mid l \in C \text{ and } \sigma(l) \neq \bot], \quad \text{if } \sigma \text{ does not trivialize } C$$
$$F\big|_\sigma = \left\{C\big|_\sigma \mid C \in F \text{ and } \sigma \text{ does not trivialize C}\right\}$$

▶ **Lemma 1.** *Let C be a clause, F be a CNF formula, and σ be an atomic substitution. The following then hold:*

(i) *σ trivializes C if and only if $I \circ \sigma \models C$ for all models I.*

(ii) *If σ does not trivialize C, then $I \circ \sigma \models C$ if and only if $I \models C\big|_\sigma$ for all models I.*

(iii) *$I \circ \sigma \models F$ if and only if $I \models F\big|_\sigma$ for all models I.*

Proof. Let us first show (ii). First, observe that I satisfies $C\big|_\sigma$ if and only if I satisfies $\sigma(l)$ for some literal $l \in C$. But this is equivalent to $(I \circ \sigma)(l) = \top$ for some $l \in C$, which is precisely $I \circ \sigma \models C$.

We now show (i). The "only if" implication is straightforward from the definition of a trivializing substitution. For the "if" implication, we show that if σ does not trivialize C, then $I \circ \sigma$ falsifies C for some model I. Claim (ii) gives out that any model I falsifying $C\big|_\sigma$, which exists because it is a (non-tautological) clause, has this property.

Claim (iii) then follows easily from claims (i) and (ii). ◀

Note that, for atomic substitutions that only map variables to the constants $\top$ or $\bot$, there exists a correspondence with cubes. In particular, given variables $x_1, \ldots, x_n, y_1, \ldots, y_m$, the cube Q is bijectively associated to the atomic substitution $Q^\star$ where:

$$Q = \langle x_1 \ldots x_n \, \overline{y_1} \ldots \overline{y_m} \rangle \qquad Q^\star = \{x_1 \mapsto \top, \ldots, x_n \mapsto \top, y_1 \mapsto \bot, \ldots, y_m \mapsto \bot\}$$

2.1 Interference-based redundancy notions

Throughout the last decade, several redundancy notions collectively called *interference-based rules* have appeared in the literature [31, 27, 23, 9]. Originating from clause elimination techniques [29, 30, 33], interference can be also used to introduce clauses in the formula; unlike more classical techniques, though, these clauses do not need to be implied by the formula, but rather *consistent* with it. Specifically, given a CNF formula F, introducing a clause C through interference requires that $F \equiv_{\text{sat}} F \cup \{C\}$.

$$\text{SUB}\ \dfrac{E_0}{A_0}$$

$$\text{RES}\ \dfrac{A_0 \qquad E_1}{A_1}$$

$$\text{RES}\ \dfrac{A_1 \qquad E_2}{A_2}$$

$$\ddots$$

$$\text{RES}\ \dfrac{A_{n-1} \qquad E_n}{A_n}$$

Figure 1 General form of a subsumption-merge chain [39, 43] deriving the clause A_n from premises $E_0, \dots, E_n$. SUB represents the subsumption rule, so it requires $E_0 \subseteq A_0$. RES represents the resolution rule, which can be applied if there is a literal $l_i \in A_{i-1}$ with $\bar{l}_i \in E_i$; in this case, $A_i = A_{i-1} \setminus \{l_i\} \vee E_i \setminus \{\bar{l}_i\}$. Subsumption-merge chains additionally require that the RES inferences are actually self-subsuming [14], i.e. $E_i \setminus \{\bar{l}_i\} \subseteq A_{i-1}$. Under these conditions, the clause A_n is a RUP clause over any CNF formula containing $E_0, \dots, E_n$. Conversely, any RUP clause over F can be derived as A_n through a subsumption-merge chain from some clauses $E_0, \dots, E_n \in F$. In fact, the E_i are the reason clauses used during unit propagation in a RUP check in reverse ordering (up to a topologically-compatible reordering) [15, 39].

Many interference-based rules are based on a criterion for entailment called *reverse unit propagation* (RUP) [17]. A clause C is called a *RUP clause* over a CNF formula F whenever unit propagation applied to F using the assumption literals $\overline{C}$ yields a conflict; under these circumstances, it can be shown that $F \models C$.

RUP clauses can be characterized in terms of resolution proofs. In particular, a clause C is a RUP clause over F if and only if C can be derived from F through a derivation of a particular form, called a *subsumption-merge chain* [39]. These are derivations as shown in Figure 1, starting with a subsumption inference and followed by a number of *resolution merges*, also known as *self-subsuming resolutions* [14]. The specifics of subsumption-merge chains in relation to RUPs are not quite relevant for our discussion; we direct the interested reader to [39, 43]. For us, it suffices to know that checking whether C is a RUP clause over F is essentially the same as finding the subsumption-merge chain that derives C from F [54].

Building on RUP clauses, many redundancy notions can be defined. The most relevant for our discussion are, in increasing generality order, *resolution-asymmetric tautologies* (RATs), *propagation redundancies* (PRs) and *substitution-redundancies* (SRs):

▶ **Definition 2.** *Let C be a clause and F be a CNF formula.*

- **(i)** *We say C is a RAT clause [31] over F upon a literal l whenever $l \in C$ and, for every clause $D \in F$ with $\bar{l} \in D$, the expression $C \vee D \setminus \{\bar{l}\}$ is either a tautology or a RUP clause over F.*
- **(ii)** *We say C is a PR clause [27] over F upon a cube Q whenever $Q \models C$ (i.e. $Q \cap C \neq \emptyset$) and each clause in $F|_{Q^\star}$ is a RUP clause over $F|_{\overline{C}^\star}$.*
- **(iii)** *We say C is a SR clause [9] over F upon an atomic substitution σ whenever σ trivializes C and each clause in $F|_\sigma$ is a RUP clause over $F|_{\overline{C}^\star}$.*

For a given *witness* (i.e. the literal l, the cube Q or the substitution σ), checking whether a clause C is a RAT/PR/SR clause over F upon the corresponding witness is polynomial over the size of F. In particular, this check takes at most one RUP check for each clause [9]; and RUP checking is quadratic on the size of F [15]. Finding the right witness is nevertheless NP-complete [27]. These redundancy notions satisfy the general condition for interference:

▶ **Theorem 3.** *Let C be a clause and F be a CNF formula, and assume either of the following:*

(a) *C is a RAT clause over F upon a literal l [31].*

(b) *C is a PR clause over F upon some cube Q [27].*

(c) *C is an SR clause over F upon some atomic substitution σ [9].*

Then, $F \equiv_{sat} F \cup \{C\}$.

In this paper we will mostly focus on substitution redundancy, which is the most general of them. However, we will use an equivalent definition, as per [9, Lemma 5]: instead of the condition that each clause in $F\big|_\sigma$ is a RUP clause over $F\big|_{\overline{C}^\star}$, we require that, for each clause $D \in F$, either σ trivializes D, or $\overline{C} \models D\big|_\sigma$, or the clause $C \vee D\big|_\sigma$ is a RUP clause over F.

2.2 Proof systems for SAT solving

RUP clauses provided the first effective solution to the problem of certifying an unsatisfiability result from a SAT solver. In particular, learnt clauses in a CDCL SAT solver [45] are RUP clauses [17, 15], so checking that each clause in the list of learnt clauses is a RUP clause over the previously derived formula amounts to certifying that the last clause in the list is entailed by the solved formula. If that clause is the empty clause, the list constitutes a refutation.

However, the proof complexity of RUP proofs is rather poor: there exist many simple problems whose refutations in resolution-based proof systems, such as RUP, are exponential on the size of the refuted formula [18, 51, 52]. In fact, this problem extends to (purely) CDCL SAT solvers, on which these results impose a performance upper bound [40, 5].

To alleviate the impact of these results, some inprocessing techniques were developed, including reencoding of cardinality constraints [6, 35], Gaussian elimination over $\mathbb{Z}_2$ [48, 47] and symmetry breaking [1, 2]. Unfortunately, the aforementioned limitations still apply to the generated refutation, so emitting a RUP proof would still take exponential time.

Allowing interference-based reasoning in the proof led to a vast number of proof formats [20, 53, 27, 13, 12, 34, 49, 9, 4] and proof generation techniques [46, 36, 22, 38, 10, 8, 16, 7]. The proof complexity of these systems is equivalent to that of extended resolution [50, 44, 32, 26], for which no exponential lower bounds are known.

Unlike more traditional, DAG-shaped proofs, interference-based proofs take the form of a list of *clause introductions* and *deletions*. Starting with the input CNF formula F, clause introductions of the form **i:** C add a clause C to F, whereas clause deletions of the form **d:** C remove C from F. At each point in the proof there is an *accumulated formula* where all the previous instructions in the proof have been applied.

Just as DAG-shaped proofs like resolution maintain a soundness invariant (i.e. each model satisfying the premises of the proof also satisfies the conclusion), interference-based proofs are *satisfiability-preserving* [39]: at any point in an interference-based proof of F, the accumulated formula G satisfies $F \models_{\text{sat}} G$. This is guaranteed by imposing some conditions on clause introductions; clause deletions do not have any requirements, because deleting a clause is always satisfiability-preserving.

Different proof systems then arise from different conditions on clause introductions. *Delete Resolution Asymmetric Tautology* (DRAT) requires them to be either RUP clauses or RAT clauses over the accumulated formula [20, 53], and similarly for *Delete Propagation Redundancy* (DPR) [27] and *Delete Substitution Redundancy* (DSR) [9]. Note that, in the case of introducing a RAT/PR/SR clause (as opposed to a RUP clause), the witness ω must be specified; in this case we denote it as **i:** C, ω.

2.3 Overwrite logic

Interference-based proofs represent a structural and semantic departure from traditional proof systems. This is due to the *non-monotonic* properties of SR: an SR clause over F upon σ is not necessarily an SR clause over a formula containing F. [31, 39].

The consequences of non-monotonicity are far-reaching. Interference-based proofs cannot be freely composed as, for example, resolution proofs can [25]: the correctness of a clause introduction depends, in principle, on the whole formula, which motivated the name "interference" as opposed to "inference" [23].

DPR proofs can be seen as model-preserving, tree-shaped, monotonic proofs over a more general logic, known as overwrite logic [42]. There, a model I can be *conditionally overwritten* with an *overwrite rule* of the form $(Q \coloneqq T)$, where Q and T are cubes. Then, the model $I \circ (Q \coloneqq T)$ is defined as $I \circ Q^\star$ if $I \models T$, or as I otherwise. That is, if T is satisfied, then the minimal assignment satisfying Q is overwritten on I. Instead of clauses, overwrite logic deals with *overwrite clauses*, represented as $\nabla\varepsilon_1 \dots \varepsilon_n . C$, where C is a clause and the $\varepsilon_i = (Q_i \coloneqq T_i)$ are overwrite rules. Such an overwrite clause is satisfied by a model I whenever $I \circ \varepsilon_1 \circ \cdots \circ \varepsilon_n \models C$.

This framework accurately expresses the reasoning performed by PR introduction [42]:

▶ **Theorem 4.** *Let C be a PR clause over a CNF formula F upon a cube Q. Then, the implication $F \models \nabla(Q \coloneqq \overline{C}).(F \cup \{C\})$ holds.*

This result means that non-monotonic, satisfiability-preserving reasoning using PR clauses can be turned into monotonic, model-preserving reasoning in overwrite logic. [42] further introduces a traditional, DAG-shaped proof system over overwrite clauses that mimics PR proofs, hence suggesting that the whole-formula dependence featured by interference-based proof systems can, to some extent, be curbed.

3 Mutation semantics for DSR proofs

The overwrite logic presented in Section 2.3 was designed to formalize the semantics of DPR proofs. In particular, models are overwritten with cubes, which act as witnesses for PR clause introductions. In order to extend this framework to DSR proofs, the role of cubes must now be fulfilled by atomic substitutions. Here we introduce *mutation logic*, which is a straightforward extension of overwrite logic.

In its most general form, a *mutation rule* is an expression $(\sigma \coloneqq \tau)$, where σ is an atomic substitution and τ is any logical expression that can be evaluated under a model. We call τ the *trigger* of the rule, and σ its *effect*. Mutation rules themselves are not logical expressions and they cannot be satisfied or falsified. They are instead intended to codify the idea "if the trigger τ is satisfied, then apply the effect substitution σ". We thus define the *application* of a mutation rule $(\sigma \coloneqq \tau)$ to a model I as:

$$I \circ (\sigma \coloneqq \tau) = \begin{cases} I \circ \sigma & \text{if } I \models \tau \\ I & \text{if } I \not\models \tau \end{cases}$$

As with overwrite logic, the main difference with propositional logic is the inclusion of a mutation operator ∇. As in [42], one can recursively define mutation formulas as either propositional formulas, or expressions of the form $\nabla(\sigma \coloneqq \tau).\varphi$ where σ is an atomic substitution and φ, τ are mutation formulas. The semantics of the mutation operator are given by $I \models \nabla(\sigma \coloneqq \tau).\varphi$ whenever $I \circ (\sigma \coloneqq \tau) \models \varphi$. In other words: evaluating $\nabla(\sigma \coloneqq \tau).\varphi$ corresponds to evaluating a formula φ' obtained from φ by applying the effect σ to φ only if the trigger τ is satisfied.

This framework is very general, but just as discussed in [42], nothing meaningful is lost by introducing some strong restrictions. For the purpose of this paper, we will only consider *cubic* mutation rules of the form $(\sigma \coloneqq Q)$ where Q is a propositional cube. The logical expressions we will use are of three kinds, where we use $\nabla\vec{\varepsilon}.\,\varphi$ to denote a nested mutation $\nabla\varepsilon_1.\,\ldots\nabla\varepsilon_n.\,\varphi$ with cubic mutations ε_i:

- *Mutation clauses* of the form $\nabla\vec{\varepsilon}.\,C$ where C is a propositional clause.
- *Mutation CNF formulas* (MCNF), which are finite sets of mutation clauses. The semantics of MCNF formulas are conjunctive, i.e. they are satisfied if every mutation clause in them is satisfied.
- *Uniformly mutation CNF formulas* (UMCNF) of the form $\nabla\vec{\varepsilon}.\,F$ where F is a propositional CNF formula. ∇ distributes over the propositional connectives, e.g. $\nabla\vec{\varepsilon}.\,(\varphi_1 \wedge \varphi_2) \equiv (\nabla\vec{\varepsilon}.\,\varphi_1) \wedge (\nabla\vec{\varepsilon}.\,\varphi_2)$. Hence, UMCNF can be embedded in the fragment of the MCNF formulas that contain clauses with the same mutation prefix.

Similarly to how overwrite logic allows the expression of PR clauses as model-preserving inferences under an overwrite [42], SR clauses become consequences under a mutation.

▶ **Theorem 5.** *Let F be a CNF formula and C be an SR clause over F upon an atomic mutation σ. Then, $F \models \nabla(\sigma \coloneqq \overline{C}).\,(F \cup \{C\})$.*

Proof. Let I be any model with $I \models F$. Our goal is to show that the model $I' = I \circ (\sigma \coloneqq \overline{C})$ satisfies $F \cup \{C\}$. If $I \models C$ holds, then $I' = I$, which satisfies both F and C.

Let us now show the case with $I \not\models C$, where we have $I' = I \circ \sigma$. First observe that, since C is an SR clause upon σ, the clause C is trivialized by σ. Lemma 1 then shows $I' \models C$. Now, consider any clause $D \in F$. By the definition of SR clauses, either σ trivializes D, or $\overline{C} \models D\big|_\sigma$, or the clause $C \vee D\big|_\sigma$ is a RUP clause over F.

As above, the first case implies $I' \models D$. For the second and third cases, it suffices to show $I \models D\big|_\sigma$, since Lemma 1 then proves $I' \models D$. For the second case, this follows from $I \models \overline{C}$. For the third case, it follows from $I \models F$ and $I \not\models C$. We have thus shown that $I' \models F \cup \{C\}$ as we wanted. ◀

As for PR clauses in [42], one can read Theorem 5 as claiming that SR clause introduction (and in general, interference-based reasoning) performs reasoning *without loss of generality*. In particular: C can be assumed in F because, were it not to hold in a given model of F, a transformation, namely the one given by σ, could be applied to the variables such that F is still satisfied after the transformation, and C becomes satisfied too.

3.1 DSR proofs as model-preserving proofs

The entailment in Theorem 5 raises the question whether SR proofs can be equivalently expressed as model-preserving, DAG-shaped proofs over the corresponding mutated clauses. Following [42], we can define a proof system as shown in Figure 2.

▶ **Theorem 6.** *The inference rules in Figure 2 are sound, i.e. any model satisfying the premises of each rule satisfies its conclusion as well.*

Proof. The proofs for RES and SUB are straightforward, since the ∇ operator preserves implications.

For ∇TAUT, consider any model I, and let $I' = I \circ \vec{\varepsilon} \circ (\sigma \coloneqq \overline{C})$. If $I \circ \vec{\varepsilon} \models C$, then $I' = I \circ \vec{\varepsilon}$, which satisfies C. Otherwise, $I' = I \circ \vec{\varepsilon} \circ \sigma$, and since σ trivializes C we have $I' \models C$.

$$\text{RES}\ \frac{\nabla\vec{\varepsilon}.\, C \qquad \nabla\vec{\varepsilon}.\, D}{\nabla\vec{\varepsilon}.\, C \setminus \{l\} \vee D \setminus \{\overline{l}\}}$$

$$\text{SUB}\ \frac{\nabla\vec{\varepsilon}.\, C}{\nabla\vec{\varepsilon}.\, D}\ \text{where } C \subseteq D$$

$$\nabla\text{TAUT}\ \frac{}{\nabla\vec{\varepsilon}.\, \nabla(\sigma := \overline{C}).\, C}\ \text{where } \sigma \text{ trivializes } C$$

$$\nabla\text{INTRO}\ \frac{\nabla\vec{\varepsilon}.\, C \qquad {}^{\star}\nabla\vec{\varepsilon}.\, \overline{Q} \vee C\big|_{\sigma}}{\nabla\vec{\varepsilon}.\, \nabla(\sigma := Q).\, C}\ \text{where } \star \text{ is only needed if } Q \not\models C\big|_{\sigma}$$

$$\nabla\text{ELIM}\ \frac{\nabla\vec{\varepsilon}.\, \nabla(\sigma := Q).\, C \qquad \nabla\vec{\varepsilon}.\, \nabla(\sigma := Q).\, C\big|_{\sigma}}{\nabla\vec{\varepsilon}.\, C\big|_{\sigma}}\ \text{where } \sigma \text{ does not trivialize } C$$

Figure 2 A proof system over mutation clauses.

Let us now show ∇ELIM correct. Consider any model I satisfying the premises, and call $I' = I \circ \vec{\varepsilon} \circ (\sigma := \overline{C})$, so that $I' \models C$ and $I' \models C\big|_{\sigma}$. If $I \circ \vec{\varepsilon} \models Q$, then $I' = I \circ \vec{\varepsilon} \circ \sigma$ satisfies C; then $I \circ \vec{\varepsilon}$ satisfies $C\big|_{\sigma}$ by Lemma 1. Otherwise, $I' = I \circ \vec{\varepsilon}$ satisfies $C\big|_{\sigma}$.

Finally, for ∇INTRO, let I be any model satisfying the premises, and $I' = I \circ \vec{\varepsilon} \circ (\sigma := Q)$. If $I \circ \vec{\varepsilon}$ satisfies Q then either it also satisfies $\overline{Q} \vee C\big|_{\sigma}$ or $Q \models C\big|_{\sigma}$. Either way, we can conclude $I \circ \vec{\varepsilon} \models C\big|_{\sigma}$, and since in this case we have $I' = I \circ \vec{\varepsilon} \circ \sigma$, Lemma 1 implies that $I' \models C$. The other case is $I \circ \vec{\varepsilon} \not\models Q$, and in this case $I' = I \circ \vec{\varepsilon}$, which satisfies C. ◀

Upon closer inspection of the proof of Theorem 5, the relation between the SR property and satisfiability-preservation becomes clearer. When each clause $D \in F$ is required that either σ trivializes D, or $\overline{C} \models D\big|_{\sigma}$, or the clause $C \vee D\big|_{\sigma}$ is a RUP clause over F, these conditions enable deriving $\nabla(\sigma := \overline{C}).\, D$ through rules ∇TAUT or ∇INTRO hold: the left-hand premise in ∇INTRO just means that C has been derived earlier in the SR proof, while the right-hand premise ensures that $C \vee D\big|_{\sigma}$ can be derived (e.g. as a RUP clause).

On the other hand, ∇TAUT guarantees that $\nabla(\sigma := \overline{C}).\, C$ can be derived, since the definition of SR clauses *forces* C to be trivialized by σ. Given that ∇ distributes over $\wedge$, these conditions are proving $F \models \nabla(\sigma := \overline{C}).\, F$.

Similar to [42], a translation of a DSR proof into a mutation logic proof then works as follows. At each step in the DSR proof, we consider the list of rules $(\sigma_i := \overline{C}_i)$ corresponding to each SR clause C_i introduced upon σ_i earlier in the proof; this list is potentially empty, e.g. at the start of the proof. Let us denote this list by $\vec{\varepsilon}$. Then, at that point, all clauses D in the accumulated CNF formula have been derived as mutation clauses $\nabla\vec{\varepsilon}.\, D$ in the translation. The translation then proceeds as follows:

1. Deletions in the DSR proof are not translated.
2. A RUP clause C can be derived through a subsumption-merge chain [39]; rules RES and SUB can express a similar derivation of the mutation version of C.
3. For an SR clause C over a CNF formula F upon an atomic substitution σ, we must derive mutation clauses $D_{\nabla} = \nabla\vec{\varepsilon}.\, \nabla(\sigma := \overline{C}).\, D$ for each clause $D \in F \cup \{C\}$.
   a. When σ trivializes D, the mutation clause D_{∇} can be derived as an axiom through ∇TAUT. Note that this case includes the case $D = C$ as well; this detail will become relevant in Section 4.1.
   b. When $\overline{C} \models D\big|_{\sigma}$, the mutation clause D_{∇} can be infered from $\nabla\vec{\varepsilon}.\, D$ through ∇INTRO. We know this premise has been previously derived because $D \in F$.

c. When $C \vee D\big|_\sigma$ is a RUP clause over F, a subsumption-merge chain deriving that clause with premises in F exists [15, 39]. Replacing clauses D' with mutation clauses $\nabla\vec{\varepsilon}.\, D'$, resolution inferences with ∇RES and subsumption inferences with ∇SUB in that proof then yields a derivation of $\nabla\vec{\varepsilon}.\, C \vee D\big|_\sigma$ from previously derived mutation clauses. Finally, the rule ∇INTRO derives D_∇.

4. At the end of the proof, the empty clause $[\,]$ is derived in the SR clause, and the translation has derived the mutated clause $\nabla\varepsilon.\,[\,]$. The identity $[\,] = [\,]\big|_\sigma$ for all substitutions σ ensures that ∇ELIM can be iteratively applied to eliminate all mutation operators, so that $[\,]$ is derived in the translation as well.

4 Extending DSR proofs

Understanding DSR proofs as mutation logic proofs opens the door to finer-grained reasoning about interference-based proofs. Crucially, one of the main issues with interference-based proofs is that deriving a clause involves reasoning over the whole currently derived formula. In particular, interference-based proofs can be highly *non-monotonic*: deleting a clause in the current formula can enable new SR introductions; and conversely, introducing a clause can disable previously available SR introductions.

This is, at first sight, at odds with the translation described in Section 3.1: the proofs we obtain there are model-preserving, DAG-shaped proofs with clear dependencies with other derived clauses. What can be derived in a subproof is never affected by independent proof sub-DAGs, so clause introduction never disables SR introductions. Deletions are even more intriguing, since they do not even exist in the mutation logic framework (just as there is no notion of deletion in a resolution proof DAG).

Another noticeable feature is how differently an SR clause C over F is treated in the definition compared to the clauses $D \in F$. Even if at first sight it might look reasonable to consider different conditions on the premises and on the conclusion, the translation from Section 3.1 uses the same set of inference rules to derive both C_∇ and D_∇.

4.1 Weak substitution redundancy

In the translation, the conditions of the definition are used to guarantee that C_∇ can be derived through a ∇TAUT inference. However, we have *three* rules that can derive this mutated clause, and the three are involved in deriving D_∇ for each $D \in F$. We can thus relax the conditions over C by demanding just the same as for each D: either σ must trivialize C, or $\overline{C} \models C\big|_\sigma$, or the clause $C \vee C\big|_\sigma$ must be a RUP clause over F.

Furthermore, there is nothing in the translation forcing us to derive D_∇ for *each and all* clauses $D \in F$. Rather, we must only do so for those clauses that the proof uses later on. However, even if we do not need D after the SR introduction, we still can use D for the RUP checks of *other* clauses in F. Note that this is not quite the same as deleting D before the SR introduction: doing so could make the RUP checks of other clauses in F fail.

These two details suggest an extension of substitution redundancy, which we call *weak substitution redundancy* (WSR).

▶ **Definition 7.** *A clause C is a WSR clause over a CNF formula F upon an atomic substitution σ modulo a subformula $\Delta \subseteq F$ whenever, for each clause $D \in (F \setminus \Delta) \cup \{C\}$, either of the following holds:*

(a) *σ trivializes D.*

(b) *$\overline{C} \models D\big|_\sigma$.*

(c) *$C \vee D\big|_\sigma$ is a RUP clause over F.*

▶ **Theorem 8.** *Let C be a WSR clause over a CNF formula F upon an atomic substitution σ modulo a subformula $\Delta \subseteq F$. Then, $F \models \nabla(\sigma \coloneqq \overline{C}).((F \setminus \Delta) \cup \{C\})$ holds. In particular, if F is satisfiable, then so is $(F \setminus \Delta) \cup \{C\}$.*

Proof. Similar to the proof of Theorem 5. The main difference is that $F \models \nabla(\sigma \coloneqq \overline{C}).C$ must now be shown using the same reasoning as $F \models \nabla(\sigma \coloneqq \overline{C}).D$ for $D \in F$. ◀

The complexity of checking a WSR clause introduction is similar to that of a PR/SR check. On the one hand, one extra RUP check might be needed if $C \vee C\big|_\sigma$ is not a tautology; on the other hand, one RUP check is spared for each clause in Δ.

A minor benefit of WSR clauses is that, while not every RUP is a RAT, PR or SR clause, every RUP clause is a WSR clause upon the identity atomic substitution. The reason for this is that the condition that the atomic substitution σ must trivialize $[]$ always fails. This allows reasoning about WSR proofs without the need for case discussion.

▶ **Corollary 9.** *Let C be clause and F be a CNF formula. Then, C is a RUP clause over F if and only if C is a WSR clause over F upon the identity atomic substitution modulo $\emptyset$.*

This, together with the embedded notion of deletions as Δ, enables the definition of a proof system with only one rule **w:** $C, \sigma \setminus \Delta$. This rule introduces clause C and deletes clauses in Δ, and is correct whenever C is a WSR clause over F upon σ modulo Δ. We call this proof system the *WSR proof system*.

5 Applications of WSR proofs

So far, we have not yet shown any benefit of WSR over SR (or that they are not equivalent, for that matter). In this section, we demonstrate techniques using WSR proofs that are unavailable in previously existing interference-based proof systems.

5.1 A shorter proof of the pigeonhole problem

One of the first propositional problems that was found to only have exponential resolution proofs was the pigeonhole problem [18]. While polynomial proofs in the extended resolution system had already been known for a decade [11], these proofs needed to introduce fresh variables to support definitions. However, the seminal work on PR clauses presented a shorter DPR proof that did not use extra variables, using $O(n^3)$ instructions [27].

In [42] an analysis of this proof from the overwrite logic perspective was presented; let us briefly reproduce it here. The pigeonhole problem encodes the unsatisfiable problem "find an assignment of n pigeons to $n-1$ pigeonholes such that no two pigeons share the same hole". We consider variables p_{ij} encoding "pigeon i is in hole j". Let us define the following clauses:

$$
\begin{aligned}
H_{in} &= [p_{ij} \mid 1 \leq j < n] \quad \text{for } n > 0 \text{ and } 1 \leq i \leq n \\
P_{ijk} &= [\overline{p_{ik}}\,\overline{p_{jk}}] \quad \text{for } 1 \leq i < j \text{ and } 1 \leq k \\
L_{ijn} &= [\overline{p_{i(n-1)}}\,\overline{p_{nj}}] \quad \text{for } n > 1,\ 1 \leq i < n \text{ and } 1 \leq j < n-1 \\
R_{in} &= [\overline{p_{i(n-1)}}] \quad \text{for } n > 1 \text{ and } i < n
\end{aligned}
$$

Briefly, H_{in} says that pigeon i stays in some hole $1 \leq j < n$; P_{ijk} prevents that pigeons i and j both occupy hole k; L_{ijn} can be read as "if the pigeon i is in the last hole, then hole j does not contain the last pigeon"; and finally R_{in} prevents that pigeon i is in the last hole. The pigeonhole problem for n pigeons is then encoded by

$$\Pi_n = \{H_{in} \mid 1 \leq i \leq n\} \cup \{P_{ijk} \mid 1 \leq i < j \leq n \text{ and } 1 \leq k < n\}$$

Intuitively, a refutation of Π_n proceeds by noting that, without loss of generality, each pigeon $i < n$ is not in hole $n-1$; were this not the case, one can swap pigeon i with pigeon n (which is not in hole n because that would violate $P_{in(n-1)}$). Then, pigeons $1, \ldots, (n-1)$ and holes $1, \ldots, (n-2)$ are in the conditions of the pigeonhole problem Π_{n-1}. This process can be iterated until Π_1 is reached, which is trivially unsatisfiable.

The proof from [27] follows this reasoning, but a single PR clause is not expressive enough to encode swaps: the only mutations that it can handle are setting variables to true or false. Thus, the proof first derives clauses L_{ijn} for $1 \leq i < n$ and $1 \leq j < n-1$ as PR clauses with the cube $Q_{ijn} = \langle \overline{p_{i(n-1)}}\, \overline{p_{nj}} p_{ij} p_{n(n-1)} \rangle$. This encodes the following reasoning: without loss of generality, if the pigeon i is in the last hole, then hole j does not contain the last pigeon; were this not the case, ensure that pigeon i is not in the last hole but in the hole j instead, and that the last pigeon is not in hole j but in the last hole instead. Once the clauses L_{ijn} have been derived for each $1 \leq i < n$, the clause R_{in} ensuring that pigeon i is not in the last hole can be derived as a RUP clause.

When considered together, the mutations $Q^{\star}_{ijn}$ for $1 \leq j < n-1$ express the atomic substitution that swaps pigeons i and n, that is:

$$\sigma_{in} = \{p_{ij} \mapsto p_{nj}, p_{nj} \mapsto p_{ij} \mid 1 \leq j < n\} \quad \text{for } 1 \leq i < n$$

DSR can handle this kind of mutation. Let us write a DSR derivation of Π_{n-1} from Π_n (where we are omitting some trailing deletions for simplicity):

$$(\textbf{i:}\ R_{1n}, \sigma_{1n}), \ldots, (\textbf{i:}\ R_{(n-1)n}, \sigma_{(n-1)n}), (\textbf{i:}\ H_{1(n-1)}), \ldots, (\textbf{i:}\ H_{(n-1)(n-1)})$$

Clauses $H_{i(n-1)}$ can be introduced as RUP clauses, since they result from resolution on H_{in} and R_{in}. Furthermore, one would hope for the R_{in} clauses to be SR clauses over the preceding formula upon σ_{in}. Let us check this. For each clause D in the preceding formula F, we need to check that either of the following holds:

(a) D is trivialized by σ_{in}
(b) $\langle p_{i(n-1)} \rangle \models D\big|_{\sigma_{in}}$
(c) the clause $D_\nabla = \left[\overline{p_{i(n-1)}}\right] \vee D\big|_{\sigma_{in}}$ is a RUP clause over F.

Checking case by case one can see that the reduct $D\big|_{\sigma_{in}}$ is always another clause in F, so D_∇ is either a tautology or can be derived by subsumption from F (which implies it is a RUP clause).

The clause R_{in}, nevertheless, is not an SR clause over F upon σ_{in}, because it is not trivialized by σ_{in}. Observe, however, that

$$C_\nabla = C \vee C\big|_{\sigma_{in}} = \left[\overline{p_{i(n-1)}}\, \overline{p_{n(n-1)}}\right] = P_{in(n-1)} \in F$$

In particular, C_∇ it is a RUP clause over F. Hence, R_{in} is in fact a WSR clause over F upon σ_{in} modulo $\emptyset$. Hence, we can define the WSR derivation π_n of Π_1 from Π_n given in Figure 3 for $n > 1$. The derivation π_n has $O(n^2)$ instructions, and is in fact a refutation, since $[] \in \Pi_1$.

5.2 Smaller cores and shorter checking runtime

SAT solvers generate proofs which often introduce clauses uninvolved in the derivation of a contradiction. This is practically unavoidable because of how solvers generate proofs: solvers mostly just log every learnt clause [17], and at that point the solver does not know what learnt clauses will be useful.

State-of-the-art proof checkers thus validate the proof backwards [19, 53]. Starting from the empty clause at the end of the proof, the checker finds out what clauses are needed to derive each clause as a RUP clause. Required clauses are then *marked*; as the checker

Figure 3 A WSR refutation π_n of the pigeonhole problem Π_n for $n \geq 1$ containing only $O(n^2)$ instructions. Here, π_1 is the empty list and id represents the identity atomic substitution.

$$
\begin{aligned}
&\textbf{w: } R_{1n},\, \sigma_{1n} \setminus \emptyset \\
&\textbf{w: } R_{2n},\, \sigma_{2n} \setminus \emptyset \\
&\quad\vdots \\
&\textbf{w: } R_{(n-1)n},\, \sigma_{(n-1)n} \setminus \emptyset \\
&\textbf{w: } H_{1(n-1)},\, \mathrm{id} \setminus \{R_{1n}\} \\
&\textbf{w: } H_{2(n-1)},\, \mathrm{id} \setminus \{R_{2n}\} \\
&\quad\vdots \\
&\textbf{w: } H_{(n-1)(n-1)},\, \mathrm{id} \setminus (\{R_{1n}\} \cup (\Pi_n \setminus \Pi_{n-1})) \\
&\pi_{n-1}
\end{aligned}
$$

proceeds backwards, unmarked clauses are skipped. If one were to visualize a RUP proof as a DAG, this amounts to only checking the connected component that actually derives the empty clause while disregarding all other connected components in the DAG.

Backwards checking has three interesting consequences. First, it vastly improves checking runtime: not only are checks for unmarked clauses skipped, but also their premises are skipped as well (unless they are used to derive another marked clause). Second, a shorter, *trimmed* proof can be extracted as a by-product of checking. Finally, by the time the checker reaches the start of the proof, the marked clauses in the input formula form a (not necessarily minimal) unsatisfiable core.

Backwards checking in interference-based proofs

Interference-based proofs do not have DAG-like dependencies as RUP proofs have. Let us formalize the problem of backwards checking in this situation. We assume that the checker keeps track of a CNF formula F and marked clauses $M \subseteq F$ as it proceeds backwards through the proof. When a RAT/PR/SR introduction **i:** C, ω is reached with $C \in M$, the checker removes C from both the formula F and the marked clauses M and validates the corresponding RAT/PR/SR introduction. The goal then is to find some (preferably small) subformula M' with $M \subseteq M' \subseteq F$ such that C is a RAT/PR/SR clause upon ω over M'; this will be the new set of marked clauses.

In the best case scenario, C satisfies the corresponding redundancy property over M, so the checker can move on with $M' = M$. There is only one way the redundancy property might not hold over M: when one of the RUP checks from Definition 2 fails over M (but still succeeds over F), the premises of the induced subsumption-merge chain must become marked; let us (conspicuously) call this set of *newly* marked clauses Δ. The problem we are tackling is whether clauses in Δ *really* need their own RUP check as mandated by Definition 2.

For RAT, it turns out, they do not: one can show that, for a witness literal l in a RAT check, the clauses in Δ never contain $\bar{l}$, so they never trigger further RUP checks. Such a convenient coincidence does not hold for PR or SR, though. In order to establish that C is a PR/SR clause upon some M', the clauses in Δ must undergo their own RUP check, which might add new clauses to Δ, and so on until fixpoint.

This is nevertheless wasteful. By the time the first Δ has been computed, introducing C can already be claimed to be satisfiability-preserving, *just not as a PR/SR*: the conditions above prove that C is a WSR clause upon ω over $M \cup \Delta$ modulo Δ. This means that a proof checker (even one that only checks PR/SR) can simply set $M' = M \cup \Delta$ and continue checking the rest of the proof.

To the best of our knowledge, existing checkers do not deal with this situation in an optimal way, e.g. the reference DPR checker `dpr-trim` resorts instead to the fixpoint method[1]. Note that the fixpoint method always produces a larger M' than the WSR-based method, with associated longer runtimes, larger unsatisfiability cores and longer trimmed proofs.

Even if for (uncertified) checking WSR only seems relevant at a theoretical level, state-of-the-art proof checkers emit trimmed, annotated proofs that can be further checked with a verified tool [12, 49]. The formats these annotated proofs use, such as LRAT or LPR, are based on RAT/PR, and so the fixpoint method is needed if an annotated proof must be emitted in one of these formats. Either way, the need for the fixpoint method could be removed by emitting WSR-based annotated proofs.

▶ **Example 10.** Let us define three CNF formulas. The formula M contains clauses:

$$[a\,\overline{c}\,x] \quad [\overline{a}\,\overline{u}\,\overline{v}\,\overline{x}] \quad [c\,\overline{u}\,\overline{v}\,x] \quad [a\,\overline{x}\,\overline{y}\,\overline{z}] \quad [a\,\overline{c}\,\overline{x}\,y] \quad [\overline{a}\,b\,u]$$
$$[c\,u] \quad [\overline{u}\,y\,z] \quad [\overline{a}\,\overline{b}\,\overline{c}] \quad [c\,\overline{x}\,\overline{z}]^{\bullet} \quad [\overline{c}\,\overline{x}z]^{\bullet} \quad [c\,\overline{x}\,\overline{y}]^{\bullet} \quad [\overline{a}\,b\,\overline{u}\,v\,\overline{x}]^{\bullet}$$

The formula Δ contains:

$$[b\,\overline{u}\,x] \quad [\overline{b}\,\overline{t}\,v\,x\,\overline{y}] \quad [\overline{b}\,t\,v\,x\,\overline{z}] \quad [t\,v\,\overline{y}\,z] \quad [\overline{t}\,v\,y\,\overline{z}]$$

Finally, $\Gamma = \{[\overline{b}\,x\,\overline{u}\,y\,\overline{z}]\}$. Let us assume that a proof checker is checking a DSR refutation of the unsafistiable formula $F = M \cup \Delta \cup \Gamma$ backwards. It eventually reaches the first instruction, an SR clause introduction for $C = [x\,\overline{u}]$ upon the atomic substitution $\sigma = \{x \mapsto \top, a \mapsto \top, v \mapsto \bot, t \mapsto \top\}$. At this point, the clauses in M (in addition to C) have been marked for checking; since this is the first instruction, the marked clauses after checking C for SR are an unsatisfiable core of F. One can check that σ trivializes C, and that all clauses in M except for the ones highlighted with $\bullet$ satisfy the conditions in Definition 2 using propagation clauses exclusively from M. For the highlighted clauses, propagating with clauses from $M \cup \Delta$ does suffice to satisfy Definition 2.

As we learnt in Section 4, we can now stop checking: C is a WSR clause over the formula $M \cup \Delta$ modulo Δ; the newly marked clauses (which form the generated unsatisfiable core) are thus $M \cup \Delta$. Current checkers will nevertheless not stop here, since SR is more restrictive than WSR. In particular, they check the newly marked clauses Δ for the conditions in Definition 2 as well. As it turns out, C is not even an SR clause over $M \cup \Delta$, but only over F: for the RUP check for $C \vee [\overline{t}\,v\,y\,\overline{z}]\,|_{\sigma}$ to succeed, the clause in Γ is needed too. That clause becomes subsequently marked, and a further check is performed for it. This check finally succeeds, reaching a fixpoint.

This example shows that SR marks strictly more clauses than WSR, which translates into larger generated unsatisfiable cores and trimmed proofs, as well as a longer checking runtime since the extra marked clauses will be themselves checked.

5.3 Interference-free interference lemmas

The differences between SR and WSR presented in Section 5.2 can too be exploited during proof generation. While the largest share of a proof generated by a state-of-the-art SAT solver consists of learnt clauses introduced as RUP clauses as well as clause deletions, inprocessing techniques also contribute to the proof. Typically, an inprocessing technique performs some

[1] See `https://github.com/marijnheule/dpr-trim/blob/83eb40b9028100aca63a419eb6d08b45acf517ad/dpr-trim.c`, line 660.

reasoning and then a (to some extent) hardcoded proof fragment of the results is generated. No proof search is performed; rather, the specialized reasoning performed by the inprocessing technique is translated into the target proof system by a method that has previously been proven correct (on paper, not *in silico*).

Interference-based proof systems are notable for their ability to generate succint proof fragments for many inprocessing techniques and non-CDCL methods, including parity reasoning [38, 16], symmetry breaking [22] and BDD-based reasoning [7]. Devising these proofs is complex for several reasons; among them is that, in an interference-based proof system, introduced lemmas may need further lemmas for satisfy Definition 2.

Let us assume we want to generate a proof fragment deriving a clause C as an SR clause from F upon some atomic substitution σ. The clause C has been obtained through some inprocessing technique, and we know that all the clauses $D_\nabla = C \vee D\big|_\sigma$ for $D \in F$ are implied by C because of some property of the inprocessing technique. However, we might find that some of the D_∇ are not RUPs over F; after all, RUP is just a criterion for entailment. We can derive some additional clauses (i.e. lemmas) $L^1, \dots, L^n$ from F such that D_∇ is a RUP over $F \cup L^1, \dots, L^n$, but now the definition of SR clauses demands that the L^i_∇ are RUP clauses as well, which might need additional clauses and so on.

This is, in essence, the proof generation version of the proof checking situation from Section 5.2. Just as we did there, with WSR we can completely bypass the need to prove that the L^i_∇ are RUP clauses: C is already a WSR clause over $F \cup \{L^1, \dots, L^n\}$ upon σ modulo $\{L^1, \dots, L^n\}$. In other words, WSR allows introducing interference lemmas that need not be taken into account for RUP checks.

▶ **Example 11.** Let us consider the CNF formula F containing clauses:

$$[a\,b\,\overline{x}\,y] \quad [a\,b\,x\,y\,\overline{z}] \quad [a\,b\,x\,z] \quad [\overline{a}\,u\,v] \quad [\overline{c}\,u\,v] \quad [a\,c\,\overline{b}\,y] \quad [a\,c\,b\,\overline{y}]$$
$$[c\,\overline{b}\,\overline{y}\,\overline{z}] \quad [c\,\overline{b}\,x\,\overline{y}\,z] \quad [c\,\overline{b}\,\overline{x}\,z] \quad [\overline{u}\,v] \quad [u\,\overline{v}] \quad [\overline{u}\,\overline{v}]$$

We want to derive the clause $C = [x]$. Unfortunately, C is not a RUP clause over F, so we try to introduce it as an SR clause upon the atomic substitution $\sigma = \{x \mapsto \top, y \mapsto z, z \mapsto y\}$. This *almost* works: all the conditions in Definition 2 hold, except for $C \vee [\overline{b}\,c\,\overline{x}\,z]\,\big|_\sigma = [\overline{b}\,c\,x\,y]$ not being a RUP clause over F. We can derive some RUP lemmas from F, for example $L_1 = [\overline{a}\,y\,v]$ and then $L_2 = [\overline{a}\,y]$; the clause $[\overline{b}\,c\,x\,y]$ is indeed a RUP clause over $F \cup \{L_1, L_2\}$.

Here is where WSR and SR show their differences again. Under WSR, we can already introduce C in F, because the paragraph above implies that C is a WSR clause over $F \cup \{L_1, L_2\}$ upon σ modulo $\{L_1, L_2\}$. This is not the case for SR, though: because the clause $C \vee L_2\big|_\sigma = [\overline{a}\,x\,z]$ is not a RUP clause over $F \cup \{L_1, L_2\}$, the clause C is not SR over F upon σ. We would need to find additional lemmas to make it so, which might then need further lemmas themselves.

6 Conclusion

We have presented a generalization of the SR redundancy notion, called weak substitution redundancy (WSR). This extension is straightforward once the semantics of interference have been understood, which we achieve by extending the overwrite logic framework from [42] into mutation logic, which is able to handle atomic substitutions.

The main differences between SR and WSR are the weakening of one unnecessarily strong condition in the definition, and the specification of a set of clauses that can be used for ensuring the interference conditions but will not participate in interference themselves.

These minor differences have an impact on the versatility of the proof system. Shorter proofs can be obtained, lemmas can be used in a less obstrusive way, the efficiency of the backwards checking algorithm is enhanced, and smaller unsatisfiable cores and trimmed proofs can be generated.

References

1 Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 22(9):1117–1137, 2003. doi:10.1109/TCAD.2003.816218.

2 Fadi A. Aloul, Karem A. Sakallah, and Igor L. Markov. Efficient symmetry breaking for boolean satisfiability. *IEEE Trans. Computers*, 55(5):549–558, 2006. doi:10.1109/TC.2006.75.

3 Johannes Altmanninger and Adrián Rebola-Pardo. Frying the egg, roasting the chicken: unit deletions in DRAT proofs. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 61–70. ACM, 2020. doi:10.1145/3372885.3373821.

4 Seulkee Baek, Mario M. Carneiro, and Marijn J. H. Heule. A flexible proof format for SAT solver-elaborator communication. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings, Part I*, volume 12651 of *Lecture Notes in Computer Science*, pages 59–75. Springer, 2021. doi:10.1007/978-3-030-72016-2_4.

5 Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res.*, 22:319–351, 2004. doi:10.1613/jair.1410.

6 Armin Biere, Daniel Le Berre, Emmanuel Lonca, and Norbert Manthey. Detecting cardinality constraints in CNF. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014 – 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 285–301. Springer, 2014. doi:10.1007/978-3-319-09284-3_22.

7 Randal E. Bryant, Armin Biere, and Marijn J. H. Heule. Clausal proofs for pseudo-boolean reasoning. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2022. doi:10.1007/978-3-030-99524-9_25.

8 Randal E. Bryant and Marijn J. H. Heule. Generating extended resolution proofs with a bdd-based SAT solver. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings, Part I*, volume 12651 of *Lecture Notes in Computer Science*, pages 76–93. Springer, 2021. doi:10.1007/978-3-030-72016-2_5.

9 Sam Buss and Neil Thapen. DRAT proofs, propagation redundancy, and extended resolution. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing – SAT 2019 – 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2019. doi:10.1007/978-3-030-24258-9_5.

10 Leroy Chew and Marijn J. H. Heule. Sorting parity encodings by reusing variables. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2020. doi:10.1007/978-3-030-51825-7_1.

11 Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8(4):28–32, 1976. doi:10.1145/1008335.1008338.

12 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction – CADE 26 – 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, 2017. doi:10.1007/978-3-319-63046-5_14.

13 Luís Cruz-Filipe, João Marques-Silva, and Peter Schneider-Kamp. Efficient certified resolution proof checking. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 118–135, 2017. doi:10.1007/978-3-662-54577-5_7.

14 Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005. doi:10.1007/11499107_5.

15 Allen Van Gelder. Producing and verifying extremely large propositional refutations – have your cake and eat it too. *Ann. Math. Artif. Intell.*, 65(4):329–372, 2012. doi:10.1007/s10472-012-9322-x.

16 Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-boolean proofs. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3768–3777. AAAI Press, 2021. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16494.

17 Evguenii I. Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10886–10891. IEEE Computer Society, 2003. doi:10.1109/DATE.2003.10008.

18 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

19 Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 181–188. IEEE, 2013. URL: http://ieeexplore.ieee.org/document/6679408/.

20 Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24 – 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2013. doi:10.1007/978-3-642-38574-2_24.

21 Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Bridging the gap between easy generation and efficient verification of unsatisfiability proofs. *Softw. Test. Verification Reliab.*, 24(8):593–607, 2014. doi:10.1002/stvr.1549.

22 Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Expressing symmetry breaking in DRAT proofs. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction – CADE-25 – 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 591–606. Springer, 2015. doi:10.1007/978-3-319-21401-6_40.

23 Marijn Heule and Benjamin Kiesl. The potential of interference-based proof systems. In Giles Reger and Dmitriy Traytel, editors, *ARCADE 2017, 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements, Gothenburg, Sweden, 6th August 2017*, EPiC Series in Computing, pages 51–54. EasyChair, 2017. URL: https://easychair.org/publications/paper/TWVW, doi:10.29007/vr7n.

24 Marijn J. H. Heule. Schur number five. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6598–6606. AAAI Press, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16952.

25 Marijn J. H. Heule and Armin Biere. Compositional propositional proofs. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning – 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 444–459. Springer, 2015. doi:10.1007/978-3-662-48899-7_31.

26 Marijn J. H. Heule and Armin Biere. What a difference a variable makes. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II*, volume 10806 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2018. doi:10.1007/978-3-319-89963-3_5.

27 Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Short proofs without new variables. In Leonardo de Moura, editor, *Automated Deduction – CADE 26 – 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 130–147. Springer, 2017. doi:10.1007/978-3-319-63046-5_9.

28 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2016. doi:10.1007/978-3-319-40970-2_15.

29 Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2010. doi:10.1007/978-3-642-12002-2_10.

30 Matti Järvisalo, Armin Biere, and Marijn Heule. Simulating circuit-level simplifications on CNF. *J. Autom. Reason.*, 49(4):583–619, 2012. doi:10.1007/s10817-011-9239-9.

31 Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning – 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012. doi:10.1007/978-3-642-31365-3_28.

32 Benjamin Kiesl, Adrián Rebola-Pardo, and Marijn J. H. Heule. Extended resolution simulates DRAT. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning – 9th International Joint Conference, IJCAR 2018, Held as Part*

of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, volume 10900 of *Lecture Notes in Computer Science*, pages 516–531. Springer, 2018. doi:10.1007/978-3-319-94205-6_34.

33 Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. Local redundancy in SAT: generalizations of blocked clauses. *Log. Methods Comput. Sci.*, 14(4), 2018. doi:10.23638/LMCS-14(4:3)2018.

34 Peter Lammich. Efficient verified (UN)SAT certificate checking. In Leonardo de Moura, editor, *Automated Deduction – CADE 26 – 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 237–254. Springer, 2017. doi:10.1007/978-3-319-63046-5_15.

35 Norbert Manthey, Marijn Heule, and Armin Biere. Automated reencoding of boolean formulas. In Armin Biere, Amir Nahir, and Tanja E. J. Vos, editors, *Hardware and Software: Verification and Testing – 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2012. doi:10.1007/978-3-642-39611-3_14.

36 Norbert Manthey and Tobias Philipp. Formula simplifications as DRAT derivations. In Carsten Lutz and Michael Thielscher, editors, *KI 2014: Advances in Artificial Intelligence – 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Proceedings*, volume 8736 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2014. doi:10.1007/978-3-319-11206-0_12.

37 Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Efficient MUS extraction with resolution. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 197–200. IEEE, 2013. URL: http://ieeexplore.ieee.org/document/6679410/.

38 Tobias Philipp and Adrián Rebola-Pardo. DRAT proofs for XOR reasoning. In Loizos Michael and Antonis C. Kakas, editors, *Logics in Artificial Intelligence – 15th European Conference, JELIA 2016, Larnaca, Cyprus, November 9-11, 2016, Proceedings*, volume 10021 of *Lecture Notes in Computer Science*, pages 415–429, 2016. doi:10.1007/978-3-319-48758-8_27.

39 Tobias Philipp and Adrián Rebola-Pardo. Towards a semantics of unsatisfiability proofs with inprocessing. In Thomas Eiter and David Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 65–84. EasyChair, 2017. URL: https://easychair.org/publications/paper/V8G, doi:10.29007/7jgq.

40 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011. doi:10.1016/j.artint.2010.10.002.

41 Adrián Rebola-Pardo and Luís Cruz-Filipe. Complete and efficient DRAT proof checking. In Nikolaj Bjørner and Arie Gurfinkel, editors, *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 – November 2, 2018*, pages 1–9. IEEE, 2018. doi:10.23919/FMCAD.2018.8602993.

42 Adrián Rebola-Pardo and Martin Suda. A theory of satisfiability-preserving proofs in SAT solving. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, volume 57 of *EPiC Series in Computing*, pages 583–603. EasyChair, 2018. URL: https://easychair.org/publications/paper/zr7z, doi:10.29007/tc7q.

43 Adrián Rebola-Pardo and Georg Weissenbacher. RAT elimination. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 423–448. EasyChair, 2020. URL: https://easychair.org/publications/paper/cMtF, doi:10.29007/fccb.

44 Robert A. Reckhow. *On the lengths of proofs in the propositional calculus*. PhD thesis, University of Toronto, 1975.

45 João P. Marques Silva and Karem A. Sakallah. GRASP – a new search algorithm for satisfiability. In Rob A. Rutenbar and Ralph H. J. M. Otten, editors, *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996*, pages 220–227. IEEE Computer Society / ACM, 1996. doi:10.1109/ICCAD.1996.569607.

46 Carsten Sinz and Armin Biere. Extended resolution proofs for conjoining bdds. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *Computer Science – Theory and Applications, First International Computer Science Symposium in Russia, CSR 2006, St. Petersburg, Russia, June 8-12, 2006, Proceedings*, volume 3967 of *Lecture Notes in Computer Science*, pages 600–611. Springer, 2006. doi:10.1007/11753728_60.

47 Mate Soos. Enhanced gaussian elimination in dpll-based SAT solvers. In Daniel Le Berre, editor, *POS-10. Pragmatics of SAT, Edinburgh, UK, July 10, 2010*, volume 8 of *EPiC Series in Computing*, pages 2–14. EasyChair, 2010. URL: https://easychair.org/publications/paper/j1D, doi:10.29007/g7ss.

48 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing – SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 – July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009. doi:10.1007/978-3-642-02777-2_24.

49 Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen. cake_lpr: Verified propagation redundancy checking in cakeml. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings, Part II*, volume 12652 of *Lecture Notes in Computer Science*, pages 223–241. Springer, 2021. doi:10.1007/978-3-030-72013-1_12.

50 G. S. Tseitin. On the complexity of derivation in propositional calculus. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 466–483. Springer Berlin Heidelberg, 1983. doi:10.1007/978-3-642-81955-1_28.

51 Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987. doi:10.1145/7531.8928.

52 Alasdair Urquhart. The symmetry rule in propositional logic. *Discret. Appl. Math.*, 96-97:177–193, 1999. doi:10.1016/S0166-218X(99)00039-6.

53 Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014 – 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014. doi:10.1007/978-3-319-09284-3_31.

54 Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10880–10885. IEEE Computer Society, 2003. doi:10.1109/DATE.2003.10014.