Faster Certified Symmetry Breaking Using Orders With Auxiliary Variables

Markus Anders[1], Bart Bogaerts[2,3], Benjamin Bogø[4,5], Arthur Gontier[6], Wietze Koops[5,4], Ciaran McCreesh[6], Magnus O. Myreen[7,8], Jakob Nordström[4,5], Andy Oertel[5,4], Adrian Rebola-Pardo[9,10], Yong Kiam Tan[11,12]

[1]RPTU Kaiserslautern-Landau, Kaiserslautern, Germany
[2]KU Leuven, Dept. of Computer Science, Leuven, Belgium
[3]Vrije Universiteit Brussel, Dept. of Computer Science, Brussels, Belgium
[4]University of Copenhagen, Copenhagen, Denmark
[5]Lund University, Lund, Sweden
[6]University of Glasgow, Glasgow, Scotland
[7]Chalmers University of Technology, Gothenburg, Sweden
[8]University of Gothenburg, Gothenburg, Sweden
[9]Vienna University of Technology, Vienna, Austria
[10]Johannes Kepler University Linz, Linz, Austria
[11]Nanyang Technological University, Singapore
[12]Institute for Infocomm Research (I^2R), A*STAR, Singapore
anders@cs.uni-kl.de, bart.bogaerts@kuleuven.be, bebo@di.ku.dk, Arthur.Gontier@glasgow.ac.uk, wietze.koops@cs.lth.se, Ciaran.McCreesh@glasgow.ac.uk, myreen@chalmers.se, jn@di.ku.dk, andy.oertel@cs.lth.se, adrian.rebola@tuwien.ac.at, yongkiam.tan@ntu.edu.sg

Abstract

Symmetry breaking is a crucial technique in modern combinatorial solving, but it is difficult to be sure it is implemented correctly. The most successful approach to deal with bugs is to make solvers *certifying*, so that they output not just a solution, but also a mathematical proof of correctness in a standard format, which can then be checked by a formally verified checker. This requires justifying symmetry reasoning within the proof, but developing efficient methods for this has remained a long-standing open challenge. A fully general approach was recently proposed by Bogaerts et al. (2023), but it relies on encoding lexicographic orders with big integers, which quickly becomes infeasible for large symmetries. In this work, we develop a method for instead encoding orders with auxiliary variables. We show that this leads to orders-of-magnitude speed-ups in both theory and practice by running experiments on proof logging and checking for SAT symmetry breaking using the state-of-the-art SATSUMA symmetry breaker and the VERIPB proof checking toolchain.

1 Introduction

An important challenge in combinatorial solving is to avoid repeatedly exploring different parts of the search space that are equivalent under symmetries. In a wide range of combinatorial solving paradigms, *symmetry breaking* is deployed as a default technique, including mixed integer programming (Achterberg and Wunderling 2013; Bolusani et al. 2024) and constraint programming (Walsh 2006). The importance of symmetry breaking is supported by both theoretical considerations (Urquhart 1999) and experimental results (Pfetsch and Rehn 2019). A detailed discussion of symmetry breaking is given, e.g., by Sakallah (2021).

Symmetry breaking has not been adopted as mainstream in Boolean satisfiability (SAT) solving, however, despite a body of work (Aloul, Markov, and Sakallah 2003; Devriendt et al. 2016; Anders, Brenner, and Rattan 2024) showing the potential for speed-ups also in this setting. One reason for this could perhaps be the higher cost, relatively speaking, of symmetry breaking compared to low-level SAT reasoning, but state-of-the art symmetry detection is efficient enough to use by default without degrading performance (Anders, Brenner, and Rattan 2024). A more important concern is that the SAT community places a strong emphasis on provable correctness. For over a decade, SAT solvers taking part in the annual SAT competitions have had to generate machine-verifiable proofs for their results. Such proofs are especially important for sophisticated techniques such as symmetry breaking, which is notoriously difficult to implement correctly. However, except for some special cases (Heule, Hunt Jr., and Wetzler 2015), it has not been known how to generate proofs for symmetry breaking in the DRAT proof format (Wetzler, Heule, and Hunt Jr. 2014) used in the competitions, or whether this is even possible.

The way symmetry breaking is typically done in SAT solving is by introducing *lex-leader* constraints, which are encoded as the clauses

$$\begin{array}{lll} s_1 \vee \overline{x}_1 & s_1 \vee y_1 & y_1 \vee \overline{x}_1 \\ s_{i+1} \vee \overline{s}_i \vee \overline{x}_{i+1} & s_{i+1} \vee \overline{s}_i \vee y_{i+1} & \overline{s}_i \vee y_{i+1} \vee \overline{x}_{i+1} \end{array} \tag{1}$$

that can be thought of as encoding a circuit enforcing $(x_1,\ldots,x_n) \preceq_{\text{lex}} (y_1,\ldots,y_n)$—here, s_i are fresh auxiliary variables encoding that the x- and y-variables are equal up to position i; using these, we enforce that x_i is false and y_i true the first time this does not hold. Such clauses are clearly not implied by the original formula, and the problem is how to prove that they can be added without changing the satisfiability of the input. Although the RAT rule (Järvisalo, Heule, and Biere 2012) in DRAT can handle a single symmetry (Kołodziejczyk and Thapen 2024), once the first symmetry is broken it is not known how or even if the other symmetries found by the symmetry breaker could be proven correct using DRAT.

Bogaerts et al. (2023) finally resolved this long-standing open problem by introducing a stronger proof format, which operates with *pseudo-Boolean* (i.e., 0–1 linear) inequalities rather than clauses, and reasons in terms of *dominance* (Chu and Stuckey 2015) to support fully general symmetry breaking without any limitations on the number of symmetries that can be handled. One benefit of this richer format is that a single inequality

$$2^{n-1}x_1 + \cdots + 2x_{n-1} + x_n \leq 2^{n-1}y_1 + \cdots + 2y_{n-1} + y_n \,, \quad (2)$$

can be used in the proof to encode lexicographic order, and from this constraint it is straightforward to derive the clauses (1) used by the solver. However, at least n^2 bits are needed to represent the coefficients in (2), while the representation of (1) scales linearly with n. This means that proof generation incurs a linear overhead compared to solving. Also, the algorithm by Anders, Brenner, and Rattan (2024) can break a symmetry in quasi-linear time measured in the number of variables k remapped by the symmetry, which introduces yet another asymptotic slowdown in proof generation if $k \ll n$. Furthermore, the exponentially growing integer coefficients in (2) require expensive arbitrary-precision arithmetic, which slows down proof checking. All of these problems combine to make the proof logging approach proposed by Bogaerts et al. (2023) infeasible for large-scale problems requiring non-trivial symmetry breaking.

In this work, we present an asymptotically faster method for generating and checking proofs of correctness for symmetry breaking. The main new technical idea is to use *auxiliary variables* to encode the lexicographic order used for the dominance reasoning, similar to the clausal encoding in (1). Unfortunately, this breaks the fundamental invariant of Bogaerts et al. (2023) that all low-level proofs should be implicational. When one needs to prove that a symmetry-breaking constraint respects lexicographical order, the encoding of this order will contain auxiliary variables that are not mentioned in the premises, and so this property cannot possibly be implied. We therefore need to make a substantial redesign of the proof system of Bogaerts et al. (2023) to work with auxiliary variables. Very briefly, our key technical twist is to split the encoding of the order into two parts, putting one part into the premises, so that the property of implicational low-level proofs can be maintained. Our redesigned proof system supports fully general symmetry breaking in a similar fashion to Bogaerts et al. (2023), but is significantly more efficient. Specifically, we prove that our approach leads to asymptotic gains for proof logging and checking for symmetry breaking by at least a linear factor in the size n of the lexicographic order used.

We have implemented support for our new proof system in the proof checker VERIPB (Bogaerts et al. 2023; Gocht and Nordström 2021; Gocht 2022) with its formally verified backend CAKEPB (Gocht et al. 2024). Together, these yield an efficient, end-to-end verified proof checking toolchain for symmetry breaking proofs. We have also enhanced the state-of-the-art SAT symmetry breaker SATSUMA (Anders, Brenner, and Rattan 2024) to generate proofs of correctness in our new format as well as that of Bogaerts et al. (2023) for a comparative evaluation of performance. Our experimental findings match our theoretical results and show that only a constant overhead in running time is required for proof logging with our new method. Proof checking performance is also vastly better compared to Bogaerts et al. (2023), although here there might be room for further improvements.

Our paper is organized as follows. After reviewing preliminaries in Section 2, we present our new proof logging system in Section 3. Sections 4 and 5 discuss how proof logging and checking can be improved asymptotically using our new method, which is confirmed by our experiments in Section 6. We conclude with a brief discussion of future work in Section 7. Further details, proofs, and a worked-out example can be found in the full-length version.

2 Preliminaries

We start with a brief review of pseudo-Boolean reasoning. For more details, we refer the reader to, e.g., Buss and Nordström (2021) or Bogaerts et al. (2023). A *Boolean variable* takes values 0 or 1. A *literal* over a Boolean variable x is x itself or its negation $\overline{x} = 1 - x$. A *pseudo-Boolean (PB) constraint* C is an integer linear inequality over literals

$$C \doteq \textstyle\sum_i a_i \ell_i \geq A \,, \quad (3)$$

where we use $\doteq$ to denote syntactic equivalence. Without loss of generality the coefficients a_i and the right-hand side A are non-negative and the literals ℓ_i are over distinct variables. The *trivially false constraint* is $\perp \doteq 0 \geq 1$. The *negation* $\neg C$ of the pseudo-Boolean constraint C in (3) is the pseudo-Boolean constraint $\neg C \doteq \sum_i a_i \overline{\ell}_i \geq \sum_i a_i - A + 1$. A *pseudo-Boolean formula* F is a conjunction $F \doteq \bigwedge_i C_i$ or equivalently a set $F \doteq \bigcup_i \{C_i\}$ of pseudo-Boolean constraints C_i, whichever view is more convenient. A *(disjunctive) clause* $\bigvee_i \ell_i$ is equivalent to the pseudo-Boolean constraint $\sum_i \ell_i \geq 1$. Hence, formulas in *conjunctive normal form (CNF)* are special cases of pseudo-Boolean formulas.

An *assignment* is a function mapping from Boolean variables to $\{0,1\}$. *Substitutions* (or *witnesses*) generalize assignments by allowing variables to be mapped to literals, too. A substitution ω is extended to literals by $\omega(\overline{x}) = \overline{\omega(x)}$, and to preserve truth values, i.e., $\omega(0) = 0$ and $\omega(1) = 1$. For a substitution ω, the support $\mathrm{supp}(\omega)$ is the set of variables x where $\omega(x) \neq x$. A substitution α can be composed with another substitution ω by applying ω first and then α, i.e., $(\alpha \circ \omega)(x) = \alpha(\omega(x))$. Applying a substitution ω to the pseudo-Boolean constraint C in (3) yields the pseudo-Boolean constraint $C{\upharpoonright_\omega} \doteq \sum_i a_i \omega(\ell_i) \geq A$. This is extended to formulas by defining $F{\upharpoonright_\omega} \doteq \bigwedge_i C_i{\upharpoonright_\omega}$. The pseudo-Boolean constraint C is satisfied by an assignment ω if $\sum_{i:\omega(\ell_i)=1} a_i \geq A$. A pseudo-Boolean formula F is satisfied by ω if ω satisfies every constraint in F. If there is no assignment that satisfies F, then F is *unsatisfiable*.

We use the notation $F(\vec{x})$ to stress that the formula is defined over the list of variables $\vec{x} = x_1, \ldots, x_n$, where we syntactically highlight a partitioning of the list of variables by writing $F(\vec{y},\vec{z})$ or $F(\vec{a},\vec{b},\vec{c})$ meaning $\vec{x}=\vec{y},\vec{z}$ or $\vec{x}=\vec{a},\vec{b},\vec{c}$, respectively (denoting concatenation of the lists of variables). To apply a substitution ω element-wise to a list of literals we write $\vec{\ell}\upharpoonright_{\omega}= \omega(\ell_1),\dots,\omega(\ell_n)$. For a formula $F(\vec{x})$ and a list of literals and truth values $\vec{y}=y_1,\dots,y_n$, the notation $F(\vec{y})$ is syntactic sugar for $F\upharpoonright_{\omega}$ with the implicitly defined substitution $\omega(x_i)=y_i$ for $i=1,\dots,n$. Finally, we write $\mathrm{var}(F)$ for the set of variables in a formula F.

2.1 The VeriPB Proof System

The proof system introduced by Bogaerts et al. (2023) (which we will refer to as the *original system*) can prove optimal values for *optimization problems* (F,f), where F is a pseudo-Boolean formula, and f is an integer linear *objective* function over literals to be minimized subject to satisfying F. The *satisfiability (SAT) problem* is a special case by having $f=0$ and F being a CNF formula. Proving the unsatisfiability of F then corresponds to proving that ∞ is a lower bound for (F,f). For clarity of exposition, we focus on decision problems, i.e., problems with objective function $f=0$, but the results can easily be extended to optimization problems as in Bogaerts et al. (2023).

A proof in this proof system consists of a sequence of rule applications, each deriving a new constraint. For implicational reasoning, the *cutting planes* proof system (Cook, Coullard, and Turán 1987) is used, which provides sound reasoning rules to derive pseudo-Boolean constraints implied by a pseudo-Boolean formula F, e.g., taking positive integer linear combinations or dividing by an integer and rounding up. We write $F\vdash C$ if there is a cutting planes proof deriving C from F. A set of constraints F' is derivable from another set F, denoted by $F\vdash F'$, if $F\vdash C$ for all $C\in F'$.

The proof system also has rules for deriving constraints which are not implied. To do this, the original system keeps track of two pseudo-Boolean formulas (i.e., sets of constraints), called *core* $\mathscr{C}$ and *derived* $\mathscr{D}$, which Järvisalo, Heule, and Biere (2012) call irredundant and redundant clauses, respectively. In addition, we need a pseudo-Boolean formula $\mathscr{O}_{\preceq}(\vec{u},\vec{v})$, encoding a preorder, i.e., a reflexive and transitive relation. This preorder is used to compare assignments α,β over the literals in a list $\vec{z}$ and we write $\alpha\preceq\beta$ if $\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\alpha},\vec{z}\upharpoonright_{\beta})$ evaluates to true. For a preorder $\preceq$, we define the strict order $\prec$ such that $\alpha\prec\beta$ holds if $\alpha\preceq\beta$ and $\beta\not\preceq\alpha$.

We call the tuple $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\vec{z})$ a *configuration*. Formally, proof rules incrementally modify the configuration. To handle optimization problems, the configuration of Bogaerts et al. (2023) also contains the current upper bound on f, which we can omit for decision problems.

The proof system maintains two invariants: (1) $\mathscr{C}$ is satisfiable if F is satisfiable, and (2) for any assignments α satisfying $\mathscr{C}$ there exists an assignment α' satisfying $\mathscr{C}$, $\mathscr{D}$, and $\alpha'\preceq\alpha$. Starting with the configuration $(F,\emptyset,\emptyset,\emptyset)$, any valid derivation of a configuration $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\vec{z})$ with $\perp\in\mathscr{C}\cup\mathscr{D}$ proves that F is unsatisfiable.

Proof Rules. We list the satisfiability version of the proof rules from Bogaerts et al. (2023) our work modifies; all other rules in the original proof system remain unchanged.

- The *redundance-based strengthening rule* (or *redundance rule* for short) allows transitioning from $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\vec{z})$ to $(\mathscr{C},\mathscr{D}\cup\{C\},\mathscr{O}_{\preceq},\vec{z})$ if a substitution ω and cutting planes proofs are provided showing that

$$\mathscr{C}\cup\mathscr{D}\cup\{\neg C\}\vdash(\mathscr{C}\cup\mathscr{D}\cup\{C\})\upharpoonright_{\omega}\cup\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\omega},\vec{z})\,. \quad (4)$$

- The *dominance-based strengthening rule* (or *dominance rule* for short) allows transitioning from $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\vec{z})$ to $(\mathscr{C},\mathscr{D}\cup\{C\},\mathscr{O}_{\preceq},\vec{z})$ if a substitution ω and cutting planes proofs are provided showing that

$$\mathscr{C}\cup\mathscr{D}\cup\{\neg C\}\vdash\mathscr{C}\upharpoonright_{\omega}\cup\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\omega},\vec{z}) \quad (5)$$

$$\mathscr{C}\cup\mathscr{D}\cup\{\neg C\}\cup\mathscr{O}_{\preceq}(\vec{z},\vec{z}\upharpoonright_{\omega})\vdash\perp\,. \quad (6)$$

We now briefly explain why the redundance rule preserves the second invariant. Let α be an assignment satisfying $\mathscr{C}$. Since the invariant holds for $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\vec{z})$, there exists an assignment α' satisfying $\mathscr{C}\cup\mathscr{D}$ and $\alpha'\preceq\alpha$. If α' happens to satisfy C, we are done. Otherwise, the derivation (4) guarantees that $\alpha'\circ\omega$ satisfies $\mathscr{C}\cup\mathscr{D}\cup\{C\}$ and $\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\alpha'\circ\omega},\vec{z}\upharpoonright_{\alpha'})$, i.e., $\alpha'\circ\omega\preceq\alpha'$. By transitivity we get $\alpha'\circ\omega\preceq\alpha$. For the dominance rule, α' might have to be composed with ω repeatedly, but the process is guaranteed to eventually satisfy C, since the composed assignment strictly decreases with respect to the order, which is encoded by (6).

Preorders. Before using a preorder $\mathscr{O}_{\preceq}$, it needs to be proven within the proof system that $\mathscr{O}_{\preceq}$ is indeed reflexive and transitive. For this, the original system requires cutting planes proofs for $\emptyset\vdash\mathscr{O}_{\preceq}(\vec{u},\vec{u})$ and $\mathscr{O}_{\preceq}(\vec{u},\vec{v})\cup\mathscr{O}_{\preceq}(\vec{v},\vec{w})\vdash\mathscr{O}_{\preceq}(\vec{u},\vec{w})$, where $\vec{w}$ is of the same size as $\vec{u}$ and $\vec{v}$.

2.2 Symmetry Breaking

We briefly review symmetry breaking as it is used in practice, which we want to certify. Typically, symmetry breaking considers permutations σ between literals with $\sigma(\overline{\ell})=\overline{\sigma(\ell)}$ for all literals ℓ, and finite support $\mathrm{supp}(\sigma)$. Practical symmetry breaking algorithms only detect *syntactic* symmetries of a formula F, i.e., permutations σ with $F\upharpoonright_{\sigma}\doteq F$.

To encode an ordering of assignments, typically the *lex-leader* constraint is used. Let S be a set of detected symmetries in a formula F, and $z_1,\dots,z_n$ be variables with $\mathrm{supp}(\sigma)\subseteq\{z_1,\dots,z_n\}$ for all $\sigma\in S$. Then we define the *lexicographic order* $\preceq_{\mathrm{lex}}$ over assignments α,β and the *lex-leader constraint* B_σ for a symmetry $\sigma\in S$ as

$$\alpha\preceq_{\mathrm{lex}}\beta \text{ iff } \textstyle\sum_{i=1}^{n}2^{n-i}\alpha(z_i)\leq\sum_{i=1}^{n}2^{n-i}\beta(z_i) \quad (7)$$

$$B_\sigma\doteq\textstyle\sum_{i=1}^{n}2^{n-i}(\sigma(x_i)-x_i)\geq 0\,. \quad (8)$$

Intuitively, (8) constrains assignments to be smaller w.r.t. the preorder in (7) than their symmetric counterpart. Symmetry breaking introduces the constraints B_σ for $\sigma\in S$ such that if F is satisfiable, then $F\cup\bigcup_{\sigma\in S}B_\sigma$ is satisfiable.

When doing proof logging for symmetry breaking, the dominance rule in the original system can derive the lex-leader constraints B_σ as follows. Suppose that the symmetry breaker detect symmetries $\sigma_1,\dots,\sigma_m$ of $\mathscr{C}$. To log these symmetries, we use the order defined by

$$\mathscr{O}_{\preceq_{\mathrm{lex}}}(\vec{x},\vec{y})=\left\{\textstyle\sum_{i=1}^{n}2^{n-i}(y_i-x_i)\geq 0\right\}\,. \quad (9)$$

To add a constraint B_{σ_i} to $\mathscr{D}$, we use the dominance rule with witness σ_i. The application of this rule is justified by $\mathscr{C} \vdash \mathscr{C}\upharpoonright_{\sigma_i}$ (trivial, because σ_i is a symmetry of $\mathscr{C}$), and the fact that $\neg B_{\sigma_i}$ implies both $\mathscr{O}_{\preceq_{\text{lex}}}(\vec{z}\upharpoonright_{\sigma_i}, \vec{z})$ and $\neg\mathscr{O}_{\preceq_{\text{lex}}}(\vec{z}, \vec{z}\upharpoonright_{\sigma_i})$.

When using symmetry breaking for the SAT problem, the symmetry breaker instead encodes $\vec{x} \preceq_{\text{lex}} \sigma(\vec{x})$ as the clauses

$$s_1 + \overline{x_1} \geq 1\,, \quad s_{i+1} + \overline{s_i} + \overline{x_{i+1}} \geq 1\,, \tag{10a}$$

$$s_1 + \sigma(x_1) \geq 1\,, \quad s_{i+1} + \overline{s_i} + \sigma(x_{i+1}) \geq 1\,, \tag{10b}$$

$$\sigma(x_1) + \overline{x_1} \geq 1\,, \quad \overline{s_i} + \sigma(x_{i+1}) + \overline{x_{i+1}} \geq 1\,, \tag{10c}$$

where s_i encodes that $(x_1, \ldots, x_i) = (\sigma(x_1), \ldots, \sigma(x_i))$. These clauses can be derived from (8) using redundance.

3 Strengthening with Auxiliary Variables

While the method presented in Section 2.2 enables proof logging for symmetry breaking, encoding the coefficients in (8) and (9) grows quadratically in the size of $\vec{z}$, which often includes all variables in the formula, making proof logging for large symmetries infeasible in practice. For proof checking, the situation is even more dire, as the proof checker has to reason internally with arbitrary-precision integer arithmetic to handle the coefficients in (8) and (9).

One way to avoid these big integers would be to represent the order as a set of clauses as in Equation (10a)–(10c), using a list of extension variables $\vec{s}$. However, this leads to challenges when defining the actual preorder $\preceq$. For an order without extension variables, we define $\alpha \preceq \beta$ to hold if $\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\alpha}, \vec{z}\upharpoonright_{\beta})$ is true. However, for a formula $\mathscr{O}_{\preceq}(\vec{x}, \vec{y}, \vec{s})$ containing extension variables $\vec{s}$ this does not work, since the variables $\vec{s}$ in $\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\alpha}, \vec{z}\upharpoonright_{\beta}, \vec{s})$ are unassigned and in general $\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\alpha}, \vec{z}\upharpoonright_{\beta}, \vec{s})$ will not hold for all assignments to $\vec{s}$.

Instead, what we are trying to capture is that $\alpha \preceq \beta$ holds precisely when $\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\alpha}, \vec{z}\upharpoonright_{\beta}, \vec{s})$ holds, *provided that* the extension variables $\vec{s}$ are set in the right way. Equivalently, we want to say that $\alpha \preceq \beta$ holds precisely when there exists an assignment ρ to $\vec{s}$ such that $\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\alpha}, \vec{z}\upharpoonright_{\beta}, \vec{s}\upharpoonright_{\rho})$ holds.

However, just adding extension variables to the proof obligations $\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\omega}, \vec{z})$ in the redundance and dominance rules would not work. What we would need to show is that *some* assignment to $\vec{s}$ exists such that $\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\omega}, \vec{z}, \vec{s})$ holds, but the proof system cannot express existential quantification. While the proof rules could specify the value of all extension variables $\vec{s}$, this would be very cumbersome. However, in all applications we have in mind, the preorder with extension variables already contains the information how to set the extension variables $\vec{s}$, since the extension variables are *defined* (functionally) in terms of the other variables.

To make this precise, let $\mathscr{S}_{\preceq}(\vec{x}, \vec{y}, \vec{s})$ be a *definition* of $\vec{s}$ in terms of the other variables (i.e., each assignment to the $\vec{x}$ and $\vec{y}$ can uniquely be extended to an assignment to $\vec{s}$ that satisfies $\mathscr{S}_{\preceq}(\vec{x}, \vec{y}, \vec{s})$). We now redefine $\preceq$ such that $\alpha \preceq \beta$ holds precisely when there exists an assignment ρ to $\vec{s}$ such that $\mathscr{S}_{\preceq}(\vec{z}\upharpoonright_{\alpha}, \vec{z}\upharpoonright_{\beta}, \vec{s}\upharpoonright_{\rho}) \wedge \mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\alpha}, \vec{z}\upharpoonright_{\beta}, \vec{s}\upharpoonright_{\rho})$ holds.

In this case, whenever we need to show that $\alpha \preceq \beta$, because of the definitional nature of $\mathscr{S}_{\preceq}$ we can *assume* that $\mathscr{S}_{\preceq}(\vec{z}\upharpoonright_{\alpha}, \vec{z}\upharpoonright_{\beta}, \vec{s}\upharpoonright_{\rho})$ holds and derive $\mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\alpha}, \vec{z}\upharpoonright_{\beta}, \vec{s}\upharpoonright_{\rho})$ from this, thereby completely eliminating the need for providing an assignment to $\vec{s}$ in every rule application. In our actual proof system, we relax the condition that $\mathscr{S}_{\preceq}$ is definitional slightly, but intuitively, $\mathscr{S}_{\preceq}$ is best thought of as a circuit defining the value of $\vec{s}$ in terms of the other variables.

An important restriction for this to be sound is that the extension variables $\vec{s}$ in the preorder, which we call *auxiliary variables*, do not appear outside the preorder.

We now formalize this. As mentioned in Section 2.1, we focus on decision problems.

3.1 Specifications

Let $\vec{a}$ be a list of variables. A pseudo-Boolean formula $\mathscr{S}(\vec{x}, \vec{a})$ is a *specification over the variables* $\vec{a}$, if it is derivable from the empty formula $\emptyset$ by the redundance rule, where each application only witnesses over variables in $\vec{a}$.

Definition 1. A formula $\mathscr{S}(\vec{x}, \vec{a}) = \{C_1, C_2, \ldots, C_n\}$ is a *specification over the variables* $\vec{a}$, if there is a list

$$(C_1, \omega_1), (C_2, \omega_2), \ldots, (C_n, \omega_n)$$

which satisfies the following:

1. The constraint C_1 can be obtained from the empty formula $\emptyset$ using the redundance rule with witness ω_1.
2. For each $i \in \{2, \ldots, n\}$ we have that C_i can be added by the redundance rule to $\bigcup_{j=1}^{i-1}\{C_j\}$ with the witness ω_i.
3. For every witness ω_i, $\operatorname{supp}(\omega_i) \subseteq \vec{a}$ holds.

A crucial property of specifications is that we can recover an assignment of the auxiliary variables from the assignment of the non-auxiliary variables. We state this property below.

Lemma 1. *Let $\mathscr{S}(\vec{x}, \vec{a})$ be a specification over $\vec{a}$. Let α be any assignment of the variables $\vec{x}$. Then, α can be extended to an assignment α', such that*

1. α' satisfies $\mathscr{S}$, and
2. $\alpha(x) = \alpha'(x)$ holds for every $x \in \vec{x}$.

To explain why Lemma 1 holds, recall from Section 2.1 that the redundance rule satisfies the following: if a constraint C is added by redundance with witness ω, then given an assignment α satisfying all other constraints, either α or $\alpha \circ \omega$ also satisfies C. Hence, defining α' by composing α with the witnesses ω_i corresponding to the constraints that do not already hold ensures that α' satisfies $\mathscr{S}$, and the fact that each witness ω_i is the identity on $\vec{x}$ (since $\operatorname{supp}(\omega_i) \subseteq \vec{a}$) ensures that $\alpha(x) = \alpha'(x)$ for $x \in \vec{x}$.

3.2 Orders with Auxiliary Variables

Next, we explain how two pseudo-Boolean formulas $\mathscr{O}_{\preceq}(\vec{u}, \vec{v}, \vec{a})$ and $\mathscr{S}_{\preceq}(\vec{u}, \vec{v}, \vec{a})$, together with the two disjoint lists of variables $\vec{z}$ and $\vec{a}$, define a preorder.

Definition 2. Let $\vec{u}$ and $\vec{v}$ be disjoint lists of variables of size n and let $\vec{a}$ be a list of auxiliary variables. Let $\mathscr{O}_{\preceq}(\vec{u}, \vec{v}, \vec{a})$ and $\mathscr{S}_{\preceq}(\vec{u}, \vec{v}, \vec{a})$ be two pseudo-Boolean formulas such that $\mathscr{S}_{\preceq}$ is a specification over $\vec{a}$.

Then we define the relation $\preceq$ over the domain of total assignments to a list of variables $\vec{z}$ of size n as follows: For assignments α, β we let $\alpha \preceq \beta$ hold, if and only if there exists an assignment ρ to the variables $\vec{a}$, such that

$$\mathscr{S}_{\preceq}(\vec{z}\upharpoonright_{\alpha}, \vec{z}\upharpoonright_{\beta}, \vec{a}\upharpoonright_{\rho}) \wedge \mathscr{O}_{\preceq}(\vec{z}\upharpoonright_{\alpha}, \vec{z}\upharpoonright_{\beta}, \vec{a}\upharpoonright_{\rho})$$

evaluates to true.

To ensure that $\mathscr{O}_{\preceq}$ and $\mathscr{S}_{\preceq}$ actually define a preorder, we require cutting planes proofs that show reflexivity, i.e., $\emptyset \vdash \alpha \preceq \alpha$, and transitivity, i.e., $\alpha \preceq \beta \wedge \beta \preceq \gamma \vdash \alpha \preceq \gamma$. To write these proof obligations using the cutting planes proof system, which cannot handle an existentially quantified conclusion, we can use the specification as a premise. The specification premise essentially tells us which auxiliary variables the existential quantifier should pick. In particular, for reflexivity, the proof obligation is

$$\mathscr{S}_{\preceq}(\vec{x},\vec{x},\vec{a}) \vdash \mathscr{O}_{\preceq}(\vec{x},\vec{x},\vec{a})\,. \tag{11}$$

For transitivity, the proof obligation is

$$\begin{aligned}&\mathscr{S}_{\preceq}(\vec{x},\vec{y},\vec{a}) \cup \mathscr{O}_{\preceq}(\vec{x},\vec{y},\vec{a}) \cup \mathscr{S}_{\preceq}(\vec{y},\vec{z},\vec{b})\\ &\qquad \cup \mathscr{O}_{\preceq}(\vec{y},\vec{z},\vec{b}) \cup \mathscr{S}_{\preceq}(\vec{x},\vec{z},\vec{c}) \vdash \mathscr{O}_{\preceq}(\vec{x},\vec{z},\vec{c})\,.\end{aligned} \tag{12}$$

Intuitively, (12) says that if the circuits defining the auxiliary variables are correctly evaluated, which is encoded by the premises $\mathscr{S}_{\preceq}(\vec{x},\vec{y},\vec{a}) \cup \mathscr{S}_{\preceq}(\vec{y},\vec{z},\vec{b}) \cup \mathscr{S}_{\preceq}(\vec{x},\vec{z},\vec{c})$, then transitivity should hold, i.e., $\mathscr{O}_{\preceq}(\vec{x},\vec{y},\vec{a}) \cup \mathscr{O}_{\preceq}(\vec{y},\vec{z},\vec{b}) \vdash \mathscr{O}_{\preceq}(\vec{x},\vec{z},\vec{c})$. However, if the auxiliary variables are not correctly set, then no claims are made.

These proof obligations ensure that $\preceq$ is a preorder:

Lemma 2. *If $\mathscr{O}_{\preceq}$ and $\mathscr{S}_{\preceq}$ satisfy Equations* (11) *and* (12)*, then $\preceq$ as defined by $\mathscr{O}_{\preceq}$ and $\mathscr{S}_{\preceq}$ is a preorder.*

3.3 Validity

We extend the configurations of the proof system to $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\mathscr{S}_{\preceq},\vec{z},\vec{a})$. In particular, we extend the notion of *weak-(F,f)-validity* from Bogaerts et al. (2023) to our new configurations, focusing on decision problems:

Definition 3. A configuration $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\mathscr{S}_{\preceq},\vec{z},\vec{a})$ is *weakly F-valid* if the following conditions hold:

1. If F is satisfiable, then $\mathscr{C}$ is satisfiable.
2. For every assignment α satisfying $\mathscr{C}$, there exists an assignment α' satisfying $\mathscr{C} \cup \mathscr{D}$ and $\alpha' \preceq \alpha$.

In the following, we furthermore assume that for any configuration $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\mathscr{S}_{\preceq},\vec{z},\vec{a})$, the following hold:

1. $\mathscr{O}_{\preceq}$ and $\mathscr{S}_{\preceq}$ refer to formulas for which Equations (11) and (12) have been successfully proven.
2. $\mathscr{S}_{\preceq}$ is a specification over $\vec{a}$.
3. The variables $\vec{a}$ only occur in $\mathscr{O}_{\preceq}$ and $\mathscr{S}_{\preceq}$, and these variables are disjoint from $\vec{z}$.

Observe that due to these invariants, satisfying assignments for $\mathscr{C} \cup \mathscr{D}$ do not need to assign the variables $\vec{a}$. In the following, we assume that such assignments are indeed defined only over the domain $\mathrm{var}(\mathscr{C} \cup \mathscr{D})$.

3.4 Dominance-Based Strengthening Rule

As in the original system, the dominance rule allows adding a constraint C to the derived set using witness ω if from the premises $\mathscr{C} \cup \mathscr{D} \cup \{\neg C\}$ we can derive $\mathscr{C}{\restriction}_{\omega}$ and show that $\alpha \circ \omega \prec \alpha$ holds for all assignments α satisfying $\mathscr{C} \cup \mathscr{D} \cup \{\neg C\}$. To show $\alpha \circ \omega \prec \alpha$, we separately show that $\alpha \circ \omega \preceq \alpha$ and $\alpha \not\preceq \alpha \circ \omega$. To show that $\alpha \circ \omega \preceq \alpha$, we have to show that $\mathscr{O}_{\preceq}(\vec{z}{\restriction}_{\omega},\vec{z},\vec{a})$, assuming that the circuit defining the auxiliary variables $\vec{a}$ has been evaluated correctly, which is encoded by the specification $\mathscr{S}_{\preceq}(\vec{z}{\restriction}_{\omega},\vec{z},\vec{a})$. This leads to the proof obligation

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \cup \mathscr{S}_{\preceq}(\vec{z}{\restriction}_{\omega},\vec{z},\vec{a}) \vdash \mathscr{O}_{\preceq}(\vec{z}{\restriction}_{\omega},\vec{z},\vec{a})\,. \tag{13}$$

To show that $\alpha \not\preceq \alpha \circ \omega$, we have to show that $\neg\mathscr{O}_{\preceq}(\vec{z},\vec{z}{\restriction}_{\omega},\vec{a})$ assuming $\mathscr{S}_{\preceq}(\vec{z},\vec{z}{\restriction}_{\omega},\vec{a})$. However, since $\neg\mathscr{O}_{\preceq}(\vec{z},\vec{z}{\restriction}_{\omega},\vec{a})$ is not necessarily a pseudo-Boolean formula (due to the negation), we instead show that we can derive contradiction from $\mathscr{O}_{\preceq}(\vec{z},\vec{z}{\restriction}_{\omega},\vec{a})$, leading to the proof obligation:

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \cup \mathscr{S}_{\preceq}(\vec{z},\vec{z}{\restriction}_{\omega},\vec{a}) \cup \mathscr{O}_{\preceq}(\vec{z},\vec{z}{\restriction}_{\omega},\vec{a}) \vdash \bot\,. \tag{14}$$

The following lemma shows that these proof obligations indeed imply $\alpha \circ \omega \preceq \alpha$ and $\alpha \not\preceq \alpha \circ \omega$, respectively:

Lemma 3. *Let G be a formula and ω a witness with* $\mathrm{supp}(\omega) \subseteq \mathrm{var}(G)$*. Furthermore, let $\vec{a} \cap \mathrm{var}(G) = \emptyset$ and $\mathscr{S}_{\preceq}$ be a specification over $\vec{a}$. Also, let $\mathscr{O}_{\preceq}$ and $\mathscr{S}_{\preceq}$ define a preorder $\preceq$. Then the following hold:*

1. *If $G \cup \mathscr{S}_{\preceq}(\vec{z}{\restriction}_{\omega},\vec{z},\vec{a}) \vdash \mathscr{O}_{\preceq}(\vec{z}{\restriction}_{\omega},\vec{z},\vec{a})$ holds, then for each assignment α satisfying G, $\alpha \circ \omega \preceq \alpha$ holds.*
2. *If $G \cup \mathscr{S}_{\preceq}(\vec{z},\vec{z}{\restriction}_{\omega},\vec{a}) \cup \mathscr{O}_{\preceq}(\vec{z},\vec{z}{\restriction}_{\omega},\vec{a}) \vdash \bot$ holds, then for each assignment α satisfying G, $\alpha \not\preceq \alpha \circ \omega$ holds.*

Hence, we define the dominance rule as follows:

Definition 4 (Dominance-based strengthening with specification)**.** We can transition from the configuration $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\mathscr{S}_{\preceq},\vec{z},\vec{a})$ to $(\mathscr{C},\mathscr{D} \cup \{C\},\mathscr{O}_{\preceq},\mathscr{S}_{\preceq},\vec{z},\vec{a})$ using the dominance rule if the following conditions are met:

1. The constraint C does not contain variables in $\vec{a}$.
2. There is a witness ω for which $\mathrm{image}(\omega) \cap \vec{a} = \emptyset$ holds.
3. We have cutting planes proofs for the following:

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \cup \mathscr{S}_{\preceq}(\vec{z}{\restriction}_{\omega},\vec{z},\vec{a}) \vdash \mathscr{C}{\restriction}_{\omega} \cup \mathscr{O}_{\preceq}(\vec{z}{\restriction}_{\omega},\vec{z},\vec{a}) \tag{15}$$

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \cup \mathscr{S}_{\preceq}(\vec{z},\vec{z}{\restriction}_{\omega},\vec{a}) \cup \mathscr{O}_{\preceq}(\vec{z},\vec{z}{\restriction}_{\omega},\vec{a}) \vdash \bot\,. \tag{16}$$

Using Lemma 3, we can show that the dominance rule preserves the invariants required by weak F-validity:

Lemma 4. *If we can transition from $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\mathscr{S}_{\preceq},\vec{z},\vec{a})$ to $(\mathscr{C},\mathscr{D} \cup \{C\},\mathscr{O}_{\preceq},\mathscr{S}_{\preceq},\vec{z},\vec{a})$ by the dominance rule, and $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\mathscr{S}_{\preceq},\vec{z},\vec{a})$ is weakly F-valid, then the configuration $(\mathscr{C},\mathscr{D} \cup \{C\},\mathscr{O}_{\preceq},\mathscr{S}_{\preceq},\vec{z},\vec{a})$ is also weakly F-valid.*

3.5 Redundance-Based Strengthening Rule

We also modify the redundance rule to work in our extended proof system. Similarly to the dominance rule, we can use $\mathscr{S}_{\preceq}(\vec{z}{\restriction}_{\omega},\vec{z},\vec{a})$ as an extra premise in our proof obligations.

Definition 5 (Redundance-based strengthening with specification)**.** We can transition from the configuration $(\mathscr{C},\mathscr{D},\mathscr{O}_{\preceq},\mathscr{S}_{\preceq},\vec{z},\vec{a})$ to $(\mathscr{C},\mathscr{D} \cup \{C\},\mathscr{O}_{\preceq},\mathscr{S}_{\preceq},\vec{z},\vec{a})$ using the redundance rule if the following conditions are met:

1. The constraint C does not contain variables in $\vec{a}$.
2. There is a witness ω for which $\mathrm{image}(\omega) \cap \vec{a} = \emptyset$ holds.
3. We have cutting planes proof that the following holds:

$$\begin{aligned}&\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \cup \mathscr{S}_{\preceq}(\vec{z}{\restriction}_{\omega},\vec{z},\vec{a})\\ &\quad \vdash (\mathscr{C} \cup \mathscr{D} \cup \{C\}){\restriction}_{\omega} \cup \mathscr{O}_{\preceq}(\vec{z}{\restriction}_{\omega},\vec{z},\vec{a})\,.\end{aligned} \tag{17}$$

The redundance rule preserves weak F-validity:

Lemma 5. *If we can transition from* $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \mathscr{S}_{\preceq}, \vec{z}, \vec{a})$ *to* $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathscr{O}_{\preceq}, \mathscr{S}_{\preceq}, \vec{z}, \vec{a})$ *by the redundance rule, and* $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \mathscr{S}_{\preceq}, \vec{z}, \vec{a})$ *is weakly F-valid, then the configuration* $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathscr{O}_{\preceq}, \mathscr{S}_{\preceq}, \vec{z}, \vec{a})$ *is also weakly F-valid.*

4 Efficient Proof Logging in SATSUMA

Using our extended proof system, we implement proof logging in the state-of-the-art symmetry breaker SATSUMA[1]. Like in the original system (as explained in Section 2.2), the (negation of the) symmetry breaking constraints can be used to show the proof obligations for the order. However, in the extended system this is more complicated, because we need to relate two different sets of extension variables (those in the symmetry breaking constraints and the auxiliary variables in the specification).

Our new method achieves an asymptotic speedup over the old method. Defining the lexicographical order over n variables can be done in time $\mathrm{O}(n)$ with our new method (for both checking and logging), while the old method requires time $\mathrm{O}(n^2)$. Breaking a symmetry σ over $k = |\operatorname{supp}(\sigma)|$ variables ($k \leq n$) takes time $\mathrm{O}(k)$ for logging and $\mathrm{O}(n)$ for checking with our new method, while the old method requires time $\mathrm{O}(nk)$ for logging and time $\mathrm{O}(n^2 + nk^2)$ for checking. Therefore, the new method is in each case asymptotically at least a factor n faster for both logging and checking than the old method used by Bogaerts et al. (2023).

5 Proof Checker Implementation

We implemented checking for our extended proof system in the proof checker VERIPB[2] and the formally verified proof checker CAKEPB[3]. Several optimizations are necessary to handle orders with many specification constraints efficiently.

Lazy Constraint Loading and Evaluation. When checking cutting planes derivations for the dominance or redundance rule (15)–(17), the proof checkers load the specification constraints from $\mathscr{S}_{\preceq}$ only when they are used in the proof. More specifically, the constraints in $\mathscr{S}_{\preceq}$ are not even computed until loaded explicitly, which improves the checking performance by a linear factor if the specification $\mathscr{S}_{\preceq}$ is not required for a cutting planes derivation.

Implicit Reflexivity Proof. Since the loaded order is always proven to be reflexive (11), the cutting planes derivation for $\mathscr{O}_{\preceq}(\vec{z}\restriction_{\omega}, \vec{z}, \vec{a})$ can be skipped for the redundance rule (17) if the domain of the witness ω does not contain a variable in $\vec{z}$. Requiring an explicit cutting planes derivation for $\mathscr{O}_{\preceq}(\vec{z}\restriction_{\omega}, \vec{z}, \vec{a})$ would again involve computation over all constraints in the specification $\mathscr{S}_{\preceq}$, which incurs a linear overhead for proof logging and checking.

Formal Verification. We updated CAKEPB in two phases, yielding the same end-to-end verification guarantees for proof checking as discussed in Gocht et al. (2024). First, we formally verified soundness of all updates to the proof system (including Lemmas 1–5). Second, we implemented and verified these changes in the CAKEPB codebase, including soundness for the optimizations described above.

6 Experimental Evaluation

From the analysis in Section 4, we know that our new proof system is asymptotically better in theory. We now show that it indeed enables much faster proof logging and checking in practice, on both crafted and real-world problem instances. The experiments in this section are performed on machines with dual AMD EPYC 7643 processors, 2TBytes of RAM, and local solid state hard drives, running Ubuntu 22.04.2. We limit each individual process to 32GBytes RAM, and run up to 16 processes in parallel (having checked that this does not make a measurable difference to runtimes). We give SATSUMA a time limit of 1,000s per instance, and VERIPB and CAKEPB 10,000s. We remark that runtimes involving writing proofs are often bound by disk I/O performance; nevertheless, our general experimental trends are valid. In each case, when we run VERIPB, we run it in elaboration mode. This means that, in addition to checking a proof, it also outputs a simplified proof that is suitable for giving to CAKEPB. This also means that any instance that fails for VERIPB due to limits cannot be run through CAKEPB.

The aim of our experiments is not to determine whether symmetry breaking is a good idea, or how it should be done. Indeed, SATSUMA produces the same CNF (modulo a potential sorting of the constraints) regardless of whether it is outputting proofs using the old method or the new method, or not outputting proofs at all. Thus, we limit our experiments to checking that the proofs produced by SATSUMA are in fact valid, rather than reporting times for checking the entire solving process. This allows us to precisely measure the effects of our changes.

We perform experiments across two sets of instances, with different purposes. Our first set consists of five families of crafted benchmarks which have well-understood symmetries, generated using CNFGEN (Lauria et al. 2017). We note that the number of variables grows quadratically or cubically in the instance size.

We show the results in Figure 1. For each of the five families, on the top row we plot the time needed to run SATSUMA to produce symmetry breaking constraints, without proof logging and with both kinds of proof logging enabled. In each case, the new method scales similarly to not doing proof logging, although there is a cost to be paid to output the proofs to disk. However, particularly for the PHP and RPHP families, it is clear that even writing the proofs is both asymptotically and practically much more expensive using the old method. On the bottom row of the figure, we plot the checking times. We see much better scaling from the new method in all five cases. Formally verified proof checking using CAKEPB is slightly slower than with VERIPB, which is not surprising—for the families where CAKEPB's curve stops on smaller instances, this is due to CAKEPB hitting memory limits when VERIPB did not. The new method is particularly helpful for CAKEPB as its formally verified ar-

[1] SATSUMA code: https://doi.org/10.5281/zenodo.17607863

[2] VERIPB code: https://doi.org/10.5281/zenodo.17608873

[3] CAKEPB code: https://doi.org/10.5281/zenodo.17609070

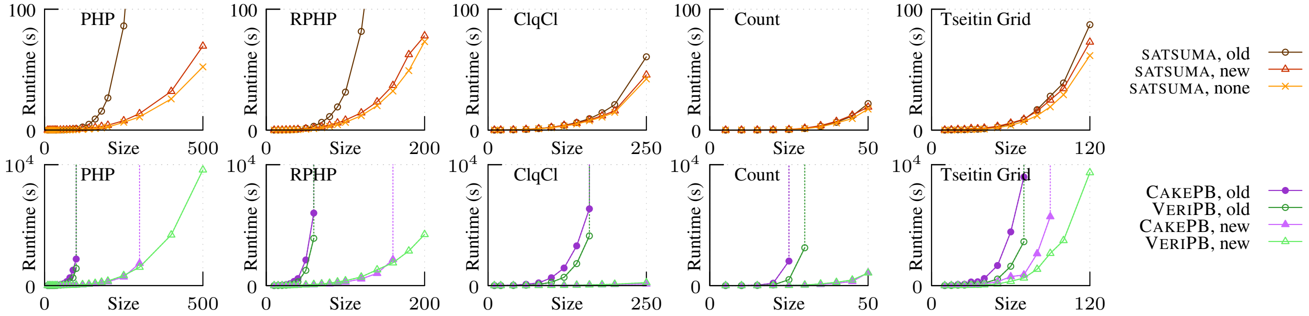

Figure 1: On top, the cost of running SATSUMA with or without proof logging, on crafted benchmark instances, as the instance size grows. In each case logging with the new method scales similarly to not doing logging, whilst the old method exhibits worse scaling for several families. On the bottom, the cost of checking these proofs: the new method exhibits better scaling.

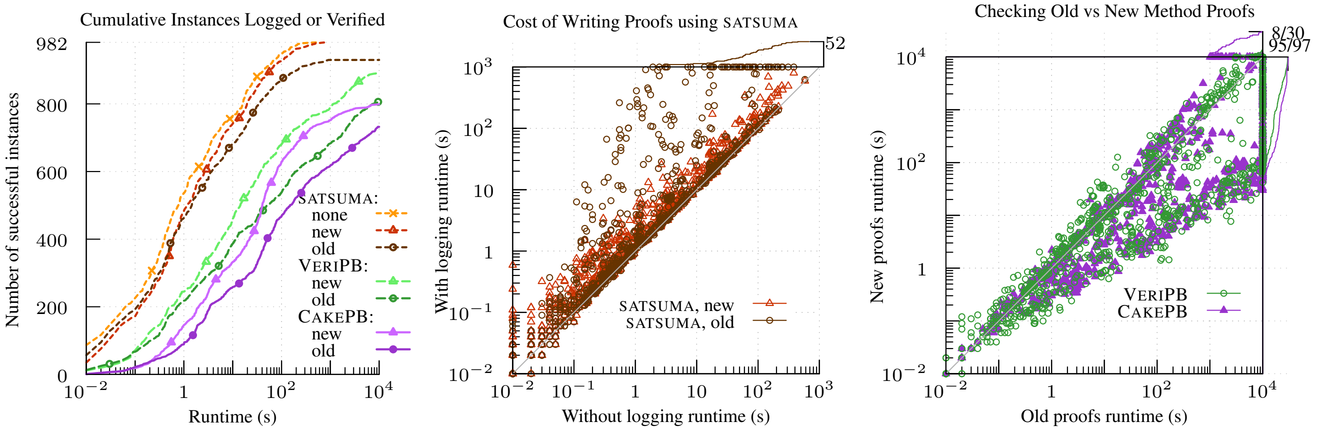

Figure 2: On the left, the cumulative number of "interesting" SAT competition instances broken, logged, and checked over time. The centre plot compares the added cost of writing proofs with the old and new methods, compared to not writing proofs; 52 instances reached time or memory limits with the old method. The right-hand plot compares the cost of checking proofs using the old and new methods, with points below the diagonal line showing where both methods succeeded but the new method was faster; additionally, 8 and 30 instances reach limits with VERIPB and CAKEPB respectively with the new method where the old method succeeded, compared to 95 and 97 respectively with the old method where the new method succeeded.

bitrary precision arithmetic library is known to be less efficient than other (unverified) libraries (Tan et al. 2019).

Our second set of instances are taken from the SAT competition (Iser and Jabs 2024). Because we are only interested in instances where we can measure something interesting about symmetry breaking, we selected the 982 instances from the main competition tracks from 2020 to 2024 where SATSUMA was able to run to completion and identify at least one symmetry. In the left-hand plot of Figure 2, we show the cumulative number of instances successfully logged or checked over time. The leftmost ("best") curve is to run SATSUMA with no proof logging, and this is closely followed by running SATSUMA with proof logging using the new method, where we could produce proofs for all 982 instances. When producing proofs using the old method, in contrast, we were only able to produce proofs for 930 instances before limits were reached. We were able to check the correctness of the symmetry breaking constraints for the new method for 893 instances (799 with CAKEPB), and with the old method for only 806 instances (732 with CAKEPB). The two scatter plots in the figure give a more detailed comparison of the added cost of running SATSUMA with proof-logging enabled, and comparing the checking costs of the old and new proof methods. In both cases it is clear that the new method is never more than a small constant factor worse than the old method, and that it is often many orders of magnitude faster.

7 Concluding Remarks

We have presented a substantial redesign of the VERIPB proof system (Bogaerts et al. 2023; Gocht and Nordström 2021; Gocht 2022) in order to support faster certified symmetry breaking. Central to our redesign is support for using auxiliary variables to encode ordering constraints over assignments. Theoretically, the use of orders with auxiliary variables allows us to avoid encoding lexicographical orders using big integers, that are prohibitive for problems with large symmetries; this improves on the previous state-of-the-art proof logging approach (Bogaerts et al. 2023) by at least a linear factor. To evaluate this in practice, we implemented proof logging using our new method in the state-of-the-art symmetry breaking tool SATSUMA (Anders, Brenner, and Rattan 2024), and proof checking for our extended system in VERIPB and the formally verified checker CAKEPB (Gocht et al. 2024); our experimental evaluation shows orders-of-magnitude improvement for proof logging and checking compared to the old approach.

Although proof logging is now asymptotically as fast as symmetry breaking, enabling proof logging can still incur a constant factor overhead. However, improving this would be mostly an engineering effort—we are already able to produce proofs for all of our benchmark instances within reasonable time. Checking the proof can still be asymptotically slower than symmetry breaking in theory, which leaves room to significantly improve the performance of proof checking for symmetry breaking. The key challenge here is that the proof checker currently has to reason about all variables in the order for each symmetry broken, while the symmetry breaker does this once at a higher level. Future work could investigate proof logging for conditional and dynamic symmetry breaking during search (Gent et al. 2005) or other dynamic methods of exploiting symmetries (Devriendt, Bogaerts, and Bruynooghe 2017), in contrast to the static symmetry breaking we presented here.

8 Acknowledgments

We would like to thank anonymous reviewers of *Pragmatics of SAT* and *AAAI* for their useful comments.

This work is partially funded by the European Union (ERC, CertiFOX, 101122653). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

This work is also partially funded by the Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G064925N).

Benjamin Bogø and Jakob Nordström are funded by the Independent Research Fund Denmark grant 9040-00389B. Jakob Nordström is also funded by the Swedish Research Council grant 2024-05801. Wietze Koops and Andy Oertel are funded by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Benjamin Bogø, Wietze Koops, Jakob Nordström and Andy Oertel also acknowledge to have benefited greatly from being part of the Basic Algorithms Research Copenhagen (BARC) environment financed by the Villum Investigator grant 54451.

Ciaran McCreesh and Arthur Gontier were supported by the Engineering and Physical Sciences Research Council grant number EP/X030032/1. Ciaran McCreesh was also supported by a Royal Academy of Engineering research fellowship.

Magnus Myreen is funded by the Swedish Research Council grant 2021-05165.

Adrian Rebola-Pardo is funded in whole or in part by the Austrian Science Fund (FWF) BILAI project 10.55776/COE12.

Yong Kiam Tan was supported by the Singapore NRF Fellowship Programme NRF-NRFF16-2024-0002.

Our computational experiments used resources provided by LUNARC at Lund University.

Different subsets of the authors wish to thank the participants of the Dagstuhl workshops 22411 *Theory and Practice of SAT and Combinatorial Solving* and 25231 *Certifying Algorithms for Automated Reasoning* and of the *1st International Workshop on Highlights in Organizing and Optimizing Proof-logging Systems (WHOOPS '24)* at the University of Copenhagen for several stimulating discussions.

References

Achterberg, T.; and Wunderling, R. 2013. Mixed Integer Programming: Analyzing 12 Years of Progress. In Jünger, M.; and Reinelt, G., eds., *Facets of Combinatorial Optimization*, 449–481. Springer.

Aloul, F. A.; Markov, I. L.; and Sakallah, K. A. 2003. Shatter: efficient symmetry-breaking for boolean satisfiability. In *Proceedings of the 40th Annual Design Automation Conference (DAC '03)*, 836–839. Association for Computing Machinery.

Anders, M.; Brenner, S.; and Rattan, G. 2024. Satsuma: Structure-Based Symmetry Breaking in SAT. In *Proceedings of the 27th International Conference on Theory and Applications of Satisfiability Testing (SAT '24)*, volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 4:1–4:23.

Biere, A.; Heule, M. J. H.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition.

Bogaerts, B.; Gocht, S.; McCreesh, C.; and Nordström, J. 2023. Certified Dominance and Symmetry Breaking for Combinatorial Optimisation. *Journal of Artificial Intelligence Research*, 77: 1539–1589. Preliminary version in *AAAI '22*.

Bolusani, S.; Besançon, M.; Bestuzheva, K.; Chmiela, A.; Dionísio, J.; Donkiewicz, T.; van Doornmalen, J.; Eifler, L.; Ghannam, M.; Gleixner, A.; Graczyk, C.; Halbig, K.; Hedtke, I.; Hoen, A.; Hojny, C.; van der Hulst, R.; Kamp, D.; Koch, T.; Kofler, K.; Lentz, J.; Manns, J.; Mexi, G.; Mühmer, E.; Pfetsch, M. E.; Schlösser, F.; Serrano, F.; Shinano, Y.; Turner, M.; Vigerske, S.; Weninger, D.; and Xu, L. 2024. The SCIP Optimization Suite 9.0. Technical report, Optimization Online. Available at https://optimization-online.org/2024/02/the-scip-optimization-suite-9-0/.

Buss, S. R.; and Nordström, J. 2021. Proof Complexity and SAT Solving. In (Biere et al. 2021), chapter 7, 233–350.

Chu, G.; and Stuckey, P. J. 2015. Dominance Breaking Constraints. *Constraints*, 20(2): 155–182. Preliminary version in *CP '12*.

Cook, W.; Coullard, C. R.; and Turán, G. 1987. On the Complexity of Cutting-Plane Proofs. *Discrete Applied Mathematics*, 18(1): 25–38.

Devriendt, J.; Bogaerts, B.; and Bruynooghe, M. 2017. Symmetric Explanation Learning: Effective Dynamic Symmetry Handling for SAT. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17)*, volume 10491 of *Lecture Notes in Computer Science*, 83–100. Springer.

Devriendt, J.; Bogaerts, B.; Bruynooghe, M.; and Denecker, M. 2016. Improved Static Symmetry Breaking for SAT. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT '16)*, volume 9710 of *Lecture Notes in Computer Science*, 104–122. Springer.

Gent, I. P.; Kelsey, T.; Linton, S. A.; McDonald, I.; Miguel, I.; and Smith, B. M. 2005. Conditional Symmetry Breaking. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP '05)*, volume 3709 of *Lecture Notes in Computer Science*, 256–270. Springer.

Gocht, S. 2022. *Certifying Correctness for Combinatorial Algorithms by Using Pseudo-Boolean Reasoning*. Ph.D. thesis, Lund University. Available at https://portal.research.lu.se/en/publications/certifying-correctness-for-combinatorial-algorithms-by-using-pseu.

Gocht, S.; McCreesh, C.; Myreen, M. O.; Nordström, J.; Oertel, A.; and Tan, Y. K. 2024. End-to-End Verification for Subgraph Solving. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI '24)*, 8038–8047.

Gocht, S.; and Nordström, J. 2021. Certifying Parity Reasoning Efficiently Using Pseudo-Boolean Proofs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, 3768–3777.

Heule, M. J. H.; Hunt Jr., W. A.; and Wetzler, N. 2015. Expressing Symmetry Breaking in DRAT Proofs. In *Proceedings of the 25th International Conference on Automated Deduction (CADE-25)*, volume 9195 of *Lecture Notes in Computer Science*, 591–606. Springer.

Iser, M.; and Jabs, C. 2024. Global Benchmark Database. In *Proceedings of the 27th International Conference on Theory and Applications of Satisfiability Testing (SAT '24)*, volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 18:1–18:10.

Järvisalo, M.; Heule, M. J. H.; and Biere, A. 2012. Inprocessing Rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR '12)*, volume 7364 of *Lecture Notes in Computer Science*, 355–370. Springer.

Kołodziejczyk, L.; and Thapen, N. 2024. The Strength of the Dominance Rule. In *Proceedings of the 27th International Conference on Theory and Applications of Satisfiability Testing (SAT '24)*, volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 20:1–20:22.

Lauria, M.; Elffers, J.; Nordström, J.; and Vinyals, M. 2017. CNFgen: A Generator of Crafted Benchmarks. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17)*, volume 10491 of *Lecture Notes in Computer Science*, 464–473. Springer.

Pfetsch, M. E.; and Rehn, T. 2019. A computational comparison of symmetry handling methods for mixed integer programs. *Math. Program. Comput.*, 11(1): 37–93.

Sakallah, K. A. 2021. Symmetry and Satisfiability. In (Biere et al. 2021), chapter 13, 509–570.

Tan, Y. K.; Myreen, M. O.; Kumar, R.; Fox, A. C. J.; Owens, S.; and Norrish, M. 2019. The Verified CakeML Compiler Backend. *Journal of Functional Programming*, 29: e2:1–e2:57.

Urquhart, A. 1999. The Symmetry Rule in Propositional Logic. *Discrete Applied Mathematics*, 96–97: 177–193.

Walsh, T. 2006. General Symmetry Breaking Constraints. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP '06)*, volume 4204 of *Lecture Notes in Computer Science*, 650–664. Springer.

Wetzler, N.; Heule, M. J. H.; and Hunt Jr., W. A. 2014. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, 422–429. Springer.