Certified Symmetry and Dominance Breaking for Combinatorial Optimisation (Including Appendices)

Bart Bogaerts,[1] **Stephan Gocht,**[2,3] **Ciaran McCreesh,**[4] **Jakob Nordström**[3,2]

[1] Vrije Universiteit Brussel
[2] Lund University, Lund, Sweden
[3] University of Copenhagen, Copenhagen, Denmark
[4] University of Glasgow, Glasgow, UK
bart.bogaerts@vub.be, stephan.gocht@cs.lth.se, Ciaran.McCreesh@glasgow.ac.uk, jn@di.ku.dk

Abstract

Symmetry and dominance breaking can be crucial for solving hard combinatorial search and optimisation problems, but the correctness of these techniques sometimes relies on subtle arguments. For this reason, it is desirable to produce efficient, machine-verifiable certificates that solutions have been computed correctly. Building on the cutting planes proof system, we develop a certification method for optimisation problems in which symmetry and dominance breaking are easily expressible. Our experimental evaluation demonstrates that we can efficiently verify fully general symmetry breaking in Boolean satisfiability (SAT) solving, thus providing, for the first time, a unified method to certify a range of advanced SAT techniques that also includes XOR and cardinality reasoning. In addition, we apply our method to maximum clique solving and constraint programming as a proof of concept that the approach applies to a wider range of combinatorial problems.

1 Introduction

Symmetries pose a challenge when solving hard combinatorial problems. For example, consider the Crystal Maze puzzle[1] shown in Figure 1, which is often used in introductory constraint modelling courses. A human modeller might notice that the puzzle is the same under a vertical mirror symmetry, and could introduce the constraint $A < G$ to eliminate this. Or, they may notice a horizontal mirror symmetry, which could be broken with $A < B$. Alternatively, they might spot that the *values* are symmetrical, and that we can interchange 1 and 8, 2 and 7, and so on; this can be eliminated by saying that $A \leq 4$. In each case a constraint is being added that preserves satisfiability overall, but that restricts a solver to finding (ideally) just one witness from each equivalence class of solutions—the hope is that this will improve solver performance. However, although we may be reasonably sure that any of these three constraints is correct individually, are combinations of these constraints valid simultaneously? What if we had said $F < C$ instead of $A < B$? And what if we could use numbers more than once? Getting symmetry elimination constraints right can be error-prone even for experienced modellers, and when dealing with larger problems with many constraints and interacting symmetries it can be hard to tell whether an instance is genuinely unsatisfiable, or was made so by an incorrect symmetry constraint.

Despite these difficulties, symmetry elimination using both manual and automatic techniques has been key to many successes across modern combinatorial optimisation paradigms such as constraint programming (CP) (Garcia de la Banda et al. 2014), Boolean satisfiability (SAT) (Biere et al. 2021), and mixed-integer programming (MIP) (Achterberg and Wunderling 2013). As these optimisation technologies are increasingly being used for high-value and life-affecting decision-making processes, it becomes vital that we can trust their outputs—and unfortunately, current solvers do not always produce the correct answer (Brummayer, Lonsing, and Biere 2010; Cook et al. 2013; Akgün et al. 2018; Gillard, Schaus, and Deville 2019). The most promising way to address this problem appears to be to use *certification*, or *proof logging*, where a solver must produce an efficiently machine-verifiable certificate that the solution given is correct (Alkassar et al. 2011; McConnell et al. 2011). This approach has been successfully used in the SAT community, with numerous proof logging formats such as *RUP* (Goldberg and Novikov 2003), *TraceCheck* (Biere 2006), *DRAT* (Heule, Hunt Jr., and Wetzler 2013a,b; Wetzler, Heule, and Hunt Jr. 2014), *GRIT* (Cruz-Filipe, Marques-Silva, and Schneider-Kamp 2017), and *LRAT* (Cruz-Filipe et al. 2017). However, currently used methods work only for decision problems, and do not support the full range of SAT solving techniques, let alone CP and MIP solving. As a case in point, there is no efficient proof logging for symmetry breaking, except

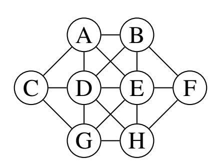

Figure 1: The Crystal Maze puzzle. Place numbers 1 to 8 in the circles, with every circle getting a different number, so that adjacent circles do not have consecutive numbers.

[1]https://theconversation.com/what-problems-will-ai-solve-in-future-an-old-british-gameshow-can-help-explain-49080

for limited cases with small symmetries which can interact only in simple ways (Heule, Hunt Jr., and Wetzler 2015). Tchinda and Djamégni (2020) recently proposed a proof logging method *DSRUP* for symmetric learning of variants of derived clauses, but this format does not support symmetry breaking (in the sense just discussed) and is also inherently unable to support pre- and inprocessing techniques, which are crucial in state-of-the-art SAT solvers.

In this work, we develop a proof logging method for *optimisation* problems, where we are given a formula F and an objective function f, that can deal with *dominance*, a generalization of symmetry. Dominance breaking starts from the observation that we can strengthen F by imposing a constraint C if every solution of F that does not satisfy C is dominated by another solution of F. This technique is used in many fields of combinatorial optimisation (**?**Walsh 2012; Gent, Petrie, and Puget 2006; McCreesh and Prosser 2016; Jouglet and Carlier 2011; Gebser, Kaminski, and Schaub 2011; Bulhões, Sadykov, and Uchoa 2018; Hoogeboom et al. 2020; Baptiste and Pape 1997; Demeulemeester and Herroelen 2002). The core idea of our method is to present an *explicit construction* of the dominating solution, so that a verifier can check that this construction strictly improves the objective value and preserves satisfaction of F. This constructed solution might itself be dominated, and hence not satisfy C, but since the objective value decreases with every application, the process must eventually terminate. Importantly, verification does not require construction of an assignment satisfying C, and can be performed efficiently even when multiple constraints are to be added; this resolves a practical issue with earlier approaches like (Heule, Hunt Jr., and Wetzler 2015), which struggle with large or overlapping symmetries. Following preliminaries in Section 2, we describe this method in full detail in Section 3.

We have developed a proof format and verifier on top of *VeriPB* (Elffers et al. 2020; Gocht and Nordström 2021; Gocht, McCreesh, and Nordström 2020; Gocht et al. 2020). The pseudo-Boolean constraints and cutting planes proof system (Cook, Coullard, and Turán 1987) used by *VeriPB* are convenient to express and reason with dominance inequalities, and moreover also make it possible to certify XOR and cardinality reasoning (Gocht and Nordström 2021), two other advanced techniques which previous SAT proof logging methods have not been able to support efficiently. In Section 4, we demonstrate that our new verifier can efficiently check *automated static symmetry breaking* in SAT, *manual static symmetry breaking* in CP, and *automated dynamic dominance handling* in maximum clique solving. While the latter two applications are proofs of concept, for static symmetry breaking we show in full generality, and for the first time, that proof logging is practical by running experiments on SAT competition benchmarks. We conclude the paper with some brief remarks in Section 5.

2 Preliminaries

Let us briefly review some standard material, referring the reader to, e.g., Buss and Nordström (2021) for more details. A *literal* ℓ over a Boolean variable x is x itself or its negation $\overline{x} = 1 - x$, where variables take values 0 (false) or 1 (true). A *pseudo-Boolean (PB) constraint* is a 0–1 linear inequality

$$C \doteq \textstyle\sum_i a_i \ell_i \geq A \ , \tag{1}$$

where a_i and A are integers (and $\doteq$ denotes syntactic equality). We can assume without loss of generality that PB constraints are *normalized*; i.e., that all literals ℓ_i are over distinct variables and that the *coefficients* a_i and the *degree (of falsity)* A are non-negative, but most of the time we will not need this. Instead, we will write PB constraints in more relaxed form as $\sum_i a_i \ell_i \geq A + \sum_j b_j \ell_j$ or $\sum_i a_i \ell_i \leq A + \sum_j b_j \ell_j$ when convenient, or even use equality $\sum_i a_i \ell_i = A$ as syntactic sugar for the pair of inequalities $\sum_i a_i \ell_i \geq A$ and $\sum_i -a_i \ell_i \geq -A$, assuming that all constraints are implicitly normalized if needed. The *negation* $\neg C$ of the constraint C in (1) is

$$\neg C \doteq \textstyle\sum_i -a_i \ell_i \geq -A + 1 \ . \tag{2}$$

A *pseudo-Boolean formula* is a conjunction $F \doteq \bigwedge_j C_j$ of PB constraints, which we can also think of as the set $\bigcup_j \{C_j\}$ of constraints in the formula, choosing whichever viewpoint seems most convenient. Note that a *(disjunctive) clause* $\ell_1 \vee \cdots \vee \ell_k$ is equivalent to the PB constraint $\ell_1 + \cdots + \ell_k \geq 1$, so formulas in *conjunctive normal form (CNF)* are special cases of PB formulas.

A *(partial) assignment* is a (partial) function from variables to $\{0, 1\}$; a *substitution* can also map variables to literals. We extend an assignment or substitution ρ from variables to literals in the natural way by respecting the meaning of negation, and for literals ℓ over variables x not in the domain of ρ, denoted $x \notin \mathit{dom}(\rho)$, we use the convention $\rho(\ell) = \ell$. (That is, we can consider all assignments and substitution to be total, but to be the identity outside of their specified domains. Strictly speaking, we also require that all substitutions be defined on the truth constants $\{0, 1\}$ and be the identity on these constants.) We sometimes write $x \mapsto b$ when $\rho(x) = b$, for b a literal or truth value.

We write $\rho \circ \omega$ to denote the composed substitution resulting from applying first ω and then ρ, i.e., $\rho \circ \omega(x) = \rho(\omega(x))$. As an example, for $\omega = \{x_1 \mapsto 0, x_3 \mapsto \overline{x}_4, x_4 \mapsto x_3\}$ and $\rho = \{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 0, x_4 \mapsto 0\}$ we have $\rho \circ \omega = \{x_1 \mapsto 0, x_2 \mapsto 1, x_3 \mapsto 1, x_4 \mapsto 0\}$. Applying ω to a constraint C as in (1) yields

$$C\!\upharpoonright_\omega \doteq \textstyle\sum_i a_i \omega(\ell_i) \geq A \ , \tag{3}$$

substituting literals or values as specified by ω. For a formula F we define $F\!\upharpoonright_\omega \doteq \bigwedge_j C_j\!\upharpoonright_\omega$.

Since we will sometimes have to make fairly elaborate use of substitutions, let us discuss some further notational conventions. If F is a formula over variables $\vec{x} = \{x_1, \ldots, x_m\}$, we can write $F(\vec{x})$ when we want to stress the set of variables over which F is defined. For a substitution ω with domain (contained in) $\vec{x}$, the notation $F\big(\vec{x}\!\upharpoonright_\omega\big)$ is understood to be a synonym of $F\!\upharpoonright_\omega$. For the same formula F and $\vec{y} = \{y_1, \ldots, y_m\}$, the notation $F(\vec{y})$ is syntactic sugar for $F\!\upharpoonright_\omega$ with ω denoting the substitution (implicitly) defined by $\omega(x_i) = y_i$ for $i = 1, \ldots, n$. Finally, for a formula $G = G(\vec{x}, \vec{y})$ over $\vec{x} \cup \vec{y}$ and substitutions α and β defined on $\vec{z} = \{z_1, \ldots, z_n\}$ (either of which could be the identity), the notation $G(\vec{z}{\restriction}_\alpha, \vec{z}{\restriction}_\beta)$ should be understood as $G{\restriction}_\omega$ for ω defined by $\omega(x_i) = \alpha(z_i)$ and $\omega(y_i) = \beta(z_i)$ for $i = 1, \ldots, n$.

The (normalized) constraint C in (1) is *satisfied* by ρ if $\sum_{\rho(\ell_i)=1} a_i \geq A$. A PB formula F is satisfied by ρ if all constraints in it are, in which case it is *satisfiable*. If there is no satisfying assignment, F is *unsatisfiable*. Two formulas are *equisatisfiable* if they are both satisfiable or both unsatisfiable. We also consider optimisation problems, where in addition to F we are given an integer linear objective function $f \doteq \sum_i w_i \ell_i$ and the task is to find an assignment that satisfies F and minimizes f. (To deal with maximization problems we can just negate the objective function.)

Cutting planes (Cook, Coullard, and Turán 1987) is a method for iteratively deriving constraints C from a pseudo-Boolean formula F. We write $F \vdash C$ for any constraint C derivable as follows. Any *axiom constraint* $C \in F$ is trivially derivable, as is any *literal axiom* $\ell \geq 0$. If $F \vdash C$ and $F \vdash D$, then any positive integer *linear combination* of these constraints is derivable. Finally, from a constraint in normalized form $\sum_i a_i \ell_i \geq A$ we can use *division* by a positive integer d to derive $\sum_i \lceil a_i/d \rceil \ell_i \geq \lceil A/d \rceil$, dividing and rounding up the degree and coefficients. For a set of PB constraints F' we write $F \vdash F'$ if $F \vdash C$ for all $C \in F'$.

For PB formulas F, F' and constraints C, C', we say that F *implies* or *models* C, denoted $F \models C$, if any assignment satisfying F also satisfies C, and write $F \models F'$ if $F \models C'$ for all $C' \in F'$. It is easy to see that if $F \vdash F'$ then $F \models F'$, and so F and $F \wedge F'$ are equisatisfiable. A constraint C is said to *literal-axiom-imply* another constraint C' if C' can be derived from C by addition of literal axioms $\ell \geq 0$.

A constraint C *unit propagates* the literal ℓ under ρ if $C{\restriction}_\rho$ cannot be satisfied unless $\ell \mapsto 1$. During *unit propagation* on F under ρ, ρ is extended iteratively by any propagated literals until an assignment ρ' is reached under which no constraint $C \in F$ is propagating, or under which some constraint C would propagate a literal had it not already been assigned to the opposite value. The latter scenario is referred to as a *conflict*, since ρ' *violates* the constraint C in this case, and ρ' is called a *conflicting* assignment. Using the generalization of (Goldberg and Novikov 2003) in (Elffers et al. 2020), we say that F implies C by *reverse unit propagation (RUP)*, and that C is a *RUP constraint* with respect to F, if $F \wedge \neg C$ unit propagates to conflict under the empty assignment. If C is a RUP constraint with respect to F, then it can be proven that there is also a derivation $F \vdash C$. More generally, it can be shown that $F \vdash C$ if and only if $F \wedge \neg C \vdash \bot$, where $\bot$ is a shorthand for the *trivially false constraint* $0 \geq 1$. Therefore, we will extend the notation and write $F \vdash C$ also when C is derivable from F by RUP or by contradiction. It is worth noting here again that, as shown in (2), the negation of any PB constraint can also be expressed syntactically as a PB constraint—this fact will be convenient in what follows.

3 A Proof System for Dominance Breaking

We proceed to develop our formal proof system for verifying dominance breaking, which we have implemented on top of the version of *VeriPB* in (Gocht and Nordström 2021). We remark that for applications it is absolutely crucial not only that the proof system be sound, but that all proofs be efficiently machine-verifiable. There are significant challenges involved in making proof logging and verification efficient, but in this section we mostly ignore these aspects of our work and focus on the theoretical underpinnings.

Our foundation is the cutting planes proof system described in Section 2. However, in a proof in our system for (F, f), where f is a linear objective function to be minimized under the pseudo-Boolean formula F (or where $f \doteq 0$ for decision problems), we also allow strengthening F by adding constraints C that are not implied by the formula. Pragmatically, adding C should be in order as long as we keep some optimal solution, i.e., a satisfying assignment to F that minimizes f, which we will refer to as an *f-minimal solution* of F. We will formalize this idea by allowing the use of an additional pseudo-Boolean formula $\mathscr{O}_\preceq(\vec{u}, \vec{v})$ that, together with a sequence of variables $\vec{z}$, defines a relation $\alpha \preceq \beta$ to hold between assignments α and β if $\mathscr{O}_\preceq(\vec{z}{\restriction}_\alpha, \vec{z}{\restriction}_\beta)$ evaluates to true. We require (a cutting planes proof) that $\mathscr{O}_\preceq$ is such that this defines a preorder, i.e., a reflexive and transitive relation. Adding new constraints C will be valid as long as we guarantee to preserve some f-minimal solution that is also minimal with respect to $\preceq$. In other words, $\preceq$ can be combined with f to define a preorder $\preceq_f$ on assignments by

$$\alpha \preceq_f \beta \quad \text{if} \quad \alpha \preceq \beta \text{ and } f{\restriction}_\alpha \leq f{\restriction}_\beta \ , \tag{4}$$

and we require that all derivation steps in the proof should preserve some solution that is minimal with respect to $\preceq_f$. The preorder defined by $\mathscr{O}_\preceq(\vec{u}, \vec{v})$ will only become important once we introduce our new *dominance-based strengthening rule* later in this section. For simplicity, up until that point the reader can assume that the pseudo-Boolean formula is $\mathscr{O}_\top \doteq \emptyset$ inducing the trivial preorder relating all assignments, though all proofs presented below work in full generality for the orders that will be introduced later.

A proof for (F, f) in our proof system consists of a sequence of *proof configurations* $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$, where

- $\mathscr{C}$ is a set of pseudo-Boolean *core constraints*;
- $\mathscr{D}$ is another set of pseudo-Boolean *derived constraints*;
- $\mathscr{O}_\preceq$ is a PB formula encoding a preorder and $\vec{z}$ a set of literals on which this preorder will be applied; and
- v is the best value found so far for f.

The initial configuration is $(F, \emptyset, \mathscr{O}_\top, \emptyset, \infty)$. The distinction between $\mathscr{C}$ and $\mathscr{D}$ is only relevant when a nontrivial preorder is used; we will elaborate on this when discussing dominance. The intended semantics of f and v is that if $v < \infty$, then there exists a solution α satisfying F such that $f{\restriction}_\alpha \leq v$, and in this case the proof can make use of the constraint $f \leq v - 1$ in the search for better solutions. As long as the optimal solution has not been found, it should hold that f-minimal solutions of $\mathscr{C} \cup \mathscr{D}$ have the same objective value as f-minimal solutions of F. The precise relation is formalized in the notion of *valid configurations* as defined next.

Definition 1. *A configuration* $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ *is* (F, f)-valid *if the following conditions hold:*

1. *If $v < \infty$, then there is a total assignment ρ satisfying F such that $f{\restriction}_\rho \leq v$.*
2. *For every $v' < v$, it holds that the sets $F \cup \{f \leq v'\}$ and $\mathscr{C} \cup \{f \leq v'\}$ are equisatisfiable.*
3. *For every total assignment ρ satisfying the constraints $\mathscr{C} \cup \{f \leq v - 1\}$, there exists a total assignment $\rho' \preceq_f \rho$ satisfying $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v - 1\}$.*

We will show that (F, f)-validity is an invariant of our proof system, i.e., that it is preserved by all derivation rules. Note that the two last items together imply that if the configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ is such that v is not yet the value of an optimal solution, then f-minimal solutions of F and of $\mathscr{C} \cup \mathscr{D}$ have the same objective value, just as desired.

A proof in our proof system ends when the configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v^*)$ is such that $\mathscr{C} \cup \mathscr{D}$ contains contradiction $\perp \doteq 0 \geq 1$. In that case, either $v^* = \infty$ and F is unsatisfiable, or v^* is the optimal value (or $v^* = 0$ for a satisfiable decision problem). We state this as a formal theorem (but due to space constraints, proofs of all statements in this section can be found in Appendix B).

Theorem 2. *Let F be a pseudo-Boolean formula and f an objective function. If $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v^*)$ is an (F, f)-valid configuration with $\{0 \geq 1\} \subseteq \mathscr{C} \cup \mathscr{D}$, then*

- *F is unsatisfiable if and only if $v^* = \infty$; and*
- *if F is satisfiable, then there is an f-minimal solution α of F with objective value $f{\restriction}_\alpha = v^*$.*

We are now ready to give a formal description of the rules in our proof system.

Implicational Derivation Rule

If we can exhibit a derivation of the pseudo-Boolean constraint C from $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v - 1\}$ in our (slightly extended) version of cutting planes as described in Section 2 (i.e., in formal notation, if $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v - 1\} \vdash C$), then we can go from the configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ to the configuration $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathscr{O}_\preceq, \vec{z}, v)$ by the *implicational derivation rule*. By the soundness of the cutting planes proof system, this means that $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v - 1\} \models C$, and so (F, f)-validity is preserved, but, more importantly, the cutting planes derivation provides a simple and efficient way for an algorithm to *verify* that this implication holds. This is a key feature of all rules in our proof system—not only are they sound, but the soundness of every rule application can be efficiently verified by checking a simple, syntactic object.

When doing proof logging, the solver would need to specify by which sequence of cutting planes derivation rules C was obtained. For practical purposes, though, it greatly simplifies matters that in many cases the verifier can figure out the required proof details automatically, meaning that the proof logger can just state the desired constraint without any further information. One important example of this is when C is a reverse unit propagation (RUP) constraint with respect to $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v - 1\}$. Another case is when C is literal-axiom-implied by some other constraint.

Objective Bound Update Rule

The *objective bound update rule* allows improving the estimate of what value can be achieved for the objective function f. We can go from $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ to $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v')$ if we know an assignment α satisfying $\mathscr{C}$ such that $f{\restriction}_\alpha = v' < v$. When actually doing proof logging, the solver would specify such an assignment α, which would then be checked by the proof verifier (in our case *VeriPB*).

To argue that this rule preserves (F, f)-validity, we note that the last two items are trivially satisfied (they are weaker after applying the rule than before). The first item is satisfied since item 2 guarantees the existence of an α' satisfying F with an objective value that is at least as good as v'. Note that we have no guarantee that α' will be a solution to F. However, although we will not emphasize this point here, it follows from our formal treatment below that the proof system guarantees that such an f-minimal solution α' to the original formula F can be efficiently reconstructed from the proof (where efficiency is measured in the size of the proof).

Redundance-Based Strengthening Rule

The redundance-based strengthening rule allows deriving a constraint C from $\mathscr{C} \cup \mathscr{D}$ even if C is not implied, provided that it can be shown that any assignment α that satisfies $\mathscr{C} \cup \mathscr{D}$ can be transformed into another assignment $\alpha' \preceq_f \alpha$ that satisfies both $\mathscr{C} \cup \mathscr{D}$ and C (in case $\mathscr{O}_\preceq = \mathscr{O}_\top$, the condition $\alpha' \preceq_f \alpha$ just means that $f{\restriction}_{\alpha'} \leq f{\restriction}_\alpha$). This rule is borrowed from (Gocht and Nordström 2021), which in turn relies heavily on (Heule, Kiesl, and Biere 2017; Buss and Thapen 2019). We extend this rule here from decision problems to optimization problems in the natural way.

Formally, we say that C can be derived from $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ by *redundance-based strengthening*, or just *redundance* for brevity, if there is a substitution ω (which we will refer to as the *witness*) such that

$$\begin{aligned} &\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \\ &\quad (\mathscr{C} \cup \mathscr{D} \cup C){\restriction}_\omega \cup \{f{\restriction}_\omega \leq f\} \cup \mathscr{O}_\preceq(\vec{z}{\restriction}_\omega, \vec{z}) \ . \end{aligned} \tag{5}$$

Intuitively, (5) says that if some assignment α satisfies $\mathscr{C} \cup \mathscr{D}$ but falsifies C, then $\alpha' = \alpha \circ \omega$ still satisfies $\mathscr{C} \cup \mathscr{D}$ and also satisfies C. In addition, the condition $f{\restriction}_\omega \leq f$ ensures that $\alpha \circ \omega$ achieves an objective function value that is at least as good as that for α. This together with the constraints $\mathscr{O}_\preceq(\vec{z}{\restriction}_\omega, \vec{z})$ guarantees that $\alpha' \preceq_f \alpha$. For proof logging purposes, the witness ω as well as any non-immediate cutting planes derivations of constraints on the right-hand side of (5) would have to be specified, but, e.g., all RUP constraints or literal-axiom-implied constraints can be left to the verifier to check.

Proposition 3. *If C is derivable from an (F, f)-valid configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ by redundance-based strengthening, then $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathscr{O}_\preceq, \vec{z}, v)$ is (F, f)-valid as well.*

Deletion Rule

We also need to be able to delete previously derived constraints. From a configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ we can transition to $(\mathscr{C}', \mathscr{D}', \mathscr{O}_\preceq, \vec{z}, v)$ using the *deletion rule* if

1. $\mathscr{D}' \subseteq \mathscr{D}$ and

2. $\mathscr{C}' = \mathscr{C}$ or $\mathscr{C}' = \mathscr{C} \setminus \{C\}$ for some constraint C derivable via the redundance rule from $(\mathscr{C}', \emptyset, \mathcal{O}_{\preceq}, \vec{z}, v)$.

This last condition above perhaps seems slightly odd, but it is there since deleting arbitrary constraints could violate (F, f)-validity in two different ways. Firstly, it would allow finding better-than-optimal solutions. Secondly, and perhaps surprisingly, in combination with the dominance-based strengthening rule, which we will discuss below, arbitrary deletion is unsound, as it can turn satisfiable instances into unsatisfiable ones. This is illustrated in Example 5 further below.

To see that deletion preserves (F, f)-validity, it is clear that item 1 remains satisfied by deletion, as does the direction of item 2 that claims satisfiability of $\mathscr{C} \cup \{f \leq v'\}$. For the other direction of item 2 and for item 3, intuitively the redundance rule guarantees that solutions of the configuration after deletion can be mapped to solutions of the configuration before deletion that are at least as good.

An alternative to condition 2 would be to enforce the more restrictive demand $\mathscr{C}' \vdash \mathscr{C}$. However, this would prevent the use of some SAT preprocessing techniques such as bounded variable elimination (Eén and Biere 2005).

Transfer Rule

Constraints can always be moved from the derived set $\mathscr{D}$ to the core set $\mathscr{C}$ using the *transfer rule*, which allows a transition from $(\mathscr{C}, \mathscr{D}, \mathcal{O}_{\preceq}, \vec{z}, v)$ to $(\mathscr{C}', \mathscr{D}, \mathcal{O}_{\preceq}, \vec{z}, v)$ if $\mathscr{C} \subseteq \mathscr{C}' \subseteq \mathscr{C} \cup \mathscr{D}$. This clearly preserves (F, f)-validity.

The transfer rule together with deletion allows replacing constraints in the original formula with stronger constraints. For example, assume that $x + y \geq 1$ is in $\mathscr{C}$ and that we derive $x \geq 1$. Then we can move $x \geq 1$ from $\mathscr{D}$ to $\mathscr{C}$ and then delete $x + y \geq 1$. The required redundance check $\{x \geq 1, \neg(x + y \geq 1)\} \vdash \perp$ is immediate.

The rules discussed so far do not change $\mathcal{O}_{\preceq}$, and so any derivation using these rules only will operate with the trivial preorder $\mathcal{O}_{\top}$ imposing no conditions. The proof system defined in terms of these rules is a straightforward extension of *VeriPB* as developed in (Elffers et al. 2020; Gocht, McCreesh, and Nordström 2020; Gocht et al. 2020; Gocht and Nordström 2021) to an optimization setting. We next discuss the main contribution of this paper, namely the new dominance rule making use of the preorder $\mathcal{O}_{\preceq}$.

Dominance-Based Strengthening Rule

Any preorder $\preceq$ induces a strict order $\prec$ defined by $\alpha \prec \beta$ if $\alpha \preceq \beta$ and $\beta \not\preceq \alpha$. The relation $\prec_f$ obtained in this way from the preorder (4) coincides with what Chu and Stuckey (2015) call a *dominance relation* in the context of constraint optimisation. Our dominance rule allows deriving a constraint C from $\mathscr{C} \cup \mathscr{D}$ even if C is not implied, similar to the redundance rule. However, for the dominance rule an assignment α satisfying $\mathscr{C} \cup \mathscr{D}$ but falsifying C need only to be mapped to an assignment α' that satisfies $\mathscr{C}$, but not necessarily $\mathscr{D}$ or C. On the other hand, the new assignment α' should satisfy the strict inequality $\alpha' \prec_f \alpha$ and not just $\alpha' \preceq_f \alpha$ as in the redundance rule. To show that this new dominance rule preserves (F, f)-validity, we will prove that it is possible to construct an assignment that satisfies $\mathscr{C} \cup \mathscr{D} \cup \{C\}$ by iteratively applying the witness of the dominance rule, in combination with (F, f)-validity of the configuration before application of the dominance rule. As our base case, if α' satisfies $\mathscr{C} \cup \mathscr{D} \cup \{C\}$, we are done. Otherwise, since α' satisfies $\mathscr{C}$, by (F, f)-validity we are guaranteed the existence of an assignment α'' satisfying $\mathscr{C} \cup \mathscr{D}$ for which $\alpha'' \prec_f \alpha' \prec_f \alpha$ holds. If α'' still does not satisfy C, we can repeat the argument. In this way, we get a strictly decreasing sequence (with respect to $\prec_f$) of assignments. Since the set of possible assignments is finite, this sequence will eventually terminate.

Formally, we can derive C by dominance-based strengthening provided that there exists a substitution ω such that

$$\begin{aligned} &\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \\ &\quad \mathscr{C}\restriction_{\omega} \cup \mathcal{O}_{\preceq}(\vec{z}\restriction_{\omega}, \vec{z}) \cup \neg\mathcal{O}_{\preceq}(\vec{z}, \vec{z}\restriction_{\omega}) \cup \{f\restriction_{\omega} \leq f\} \ , \end{aligned} \tag{6}$$

where $\mathcal{O}_{\preceq}(\vec{z}\restriction_{\omega}, \vec{z})$ and $\neg\mathcal{O}_{\preceq}(\vec{z}, \vec{z}\restriction_{\omega})$ together state that $\alpha \circ \omega \prec \alpha$ for any assignment α. A minor technical problem is that the pseudo-Boolean formula $\mathcal{O}_{\preceq}(\vec{z}, \vec{z}\restriction_{\omega})$ may contain multiple constraints, so that the negation of it is no longer a PB formula. To get around this, we split (6) into two separate conditions and shift $\neg\mathcal{O}_{\preceq}(\vec{z}, \vec{z}\restriction_{\omega})$ to the premise of the implication, which eliminates the negation. Thus, the formal version of our *dominance-based strengthening rule*, or just *dominance rule* for brevity, says that we can go from $(\mathscr{C}, \mathscr{D}, \mathcal{O}_{\preceq}, \vec{z}, v)$ to $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathcal{O}_{\preceq}, \vec{z}, v)$ if there is a substitution ω such that the conditions

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \mathscr{C}\restriction_{\omega} \cup \mathcal{O}_{\preceq}(\vec{z}\restriction_{\omega}, \vec{z}) \cup \{f\restriction_{\omega} \leq f\} \tag{7a}$$

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \cup \mathcal{O}_{\preceq}(\vec{z}, \vec{z}\restriction_{\omega}) \vdash \perp \tag{7b}$$

are satisfied. Just as for the redundance rule, the witness ω as well as any non-immediate derivations would have to be specified in the proof log.

Proposition 4. *If C is derivable from an (F, f)-valid configuration $(\mathscr{C}, \mathscr{D}, \mathcal{O}_{\preceq}, \vec{z}, v)$ by dominance-based strengthening, then $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathcal{O}_{\preceq}, \vec{z}, v)$ is also (F, f)-valid.*

When introducing the deletion rule, we already mentioned that deleting arbitrary constraints can be unsound in combination with dominance-based strengthening. We now illustrate this phenomenon.

Example 5. *Consider the formula $F = \{p \geq 1\}$ with objective $f \doteq 0$ and the configuration*

$$(\mathscr{C}_1 = \{p \geq 1\}, \mathscr{D}_1 = \{p \geq 1\}, \mathcal{O}_{\preceq}, \{p\}, \infty) \ , \tag{8}$$

where $\mathcal{O}_{\preceq}(u, v)$ is defined as $\{v + \overline{u} \geq 1\}$. This configuration is (F, f)-valid and $\mathscr{C} \cup \mathscr{D}$ is satisfiable. If *we were allowed to delete constraints arbitrarily from $\mathscr{C}$, we could derive a configuration with $\mathscr{C}_2 = \emptyset$ and $\mathscr{D}_2 = \{p \geq 1\}$. However, now the dominance rule can derive $C \doteq \overline{p} \geq 1$, using the witness $\omega = \{p \mapsto 0\}$. To see that all conditions for applying dominance-based strengthening are indeed satisfied, we notice that conditions* (7a)–(7b) *simplify to*

$$\emptyset \cup \{p \geq 1\} \cup \{p \geq 1\} \vdash \emptyset \cup \{p + 1 \geq 1\} \cup \emptyset \tag{9a}$$

$$\emptyset \cup \{p \geq 1\} \cup \{p \geq 1\} \cup \{0 + \overline{p} \geq 1\} \vdash \perp \tag{9b}$$

respectively. Both claims clearly hold, meaning that we arrive at a configuration that contains both $p \geq 1$ and $\overline{p} \geq 1$.

Preorder Encodings

As mentioned before, $\mathscr{O}_{\preceq}$ is shorthand for a pseudo-Boolean formula $\mathscr{O}_{\preceq}(\vec{u}, \vec{v})$ over two sets of formal placeholder variables $\vec{u} = \{u_1, \ldots, u_n\}$ and $\vec{v} = \{v_1, \ldots, v_n\}$ of equal size, which should also match the size of $\vec{z}$ in the configuration. To use $\mathscr{O}_{\preceq}$ in a proof, it is required to show that this formula encodes a preorder. This is done by providing (in a proof preamble) cutting planes derivations establishing

$$\emptyset \vdash \mathscr{O}_{\preceq}(\vec{u}, \vec{u}) \tag{10a}$$

$$\mathscr{O}_{\preceq}(\vec{u}, \vec{v}) \cup \mathscr{O}_{\preceq}(\vec{v}, \vec{w}) \vdash \mathscr{O}_{\preceq}(\vec{u}, \vec{w}) \tag{10b}$$

where (10a) formalizes reflexivity and (10b) transitivity (and where notation like $\mathscr{O}_{\preceq}(\vec{v}, \vec{w})$ is shorthand for applying to $\mathscr{O}_{\preceq}(\vec{u}, \vec{v})$ the substitution ω that maps u_i to v_i and v_i to w_i, as discussed in Section 2). These two conditions guarantee that the relation $\preceq$ defined by $\alpha \preceq \beta$ if $\mathscr{O}_{\preceq}(\vec{z}{\upharpoonright}_{\alpha}, \vec{z}{\upharpoonright}_{\beta})$ forms a preorder on the set of assignments.

By way of example, to encode the lexicographic order $u_1 u_2 \ldots u_n \preceq_{\text{lex}} v_1 v_2 \ldots v_n$, we can use a single constraint

$$\mathscr{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{v}) \doteq \textstyle\sum_{i=1}^{n} 2^{n-i} \cdot (v_i - u_i) \geq 0 \ . \tag{11}$$

Reflexivity is vacuously true since $\mathscr{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{u}) \doteq 0 \geq 0$, and transitivity also follows easily since adding $\mathscr{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{v})$ and $\mathscr{O}_{\preceq_{\text{lex}}}(\vec{v}, \vec{w})$ yields $\mathscr{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{w})$.

A potential concern with encodings such as (11) is that coefficients can become very large as the number of variables in the order grows. It is perfectly possible to address this by allowing order encodings using auxiliary variables in addition to $\vec{u}$ and $\vec{v}$. We have chosen not to develop the theory for this in the current paper, however, since we feel that it makes the exposition unnecessarily complicated without adding anything of real significance to the scientific contribution.

Order Change Rule

The final proof rule that we need is a rule for introducing a nontrivial order, and it turns out that it can also be convenient to be able to use different orders at different points in the proof. Switching orders is possible, but to maintain soundness it is important to first clear the set $\mathscr{D}$ (after transferring the constraints we want to keep to $\mathscr{C}$). The reason for this is simple: if we allow arbitrary order changes, then the third item of (F, f)-validity would no longer hold, but when $\mathscr{D} = \emptyset$, it is trivially true.

Formally, provided that $\mathscr{O}_{\preceq^2}$ has been established to be a preorder (via cutting planes proofs for (10a) and (10b)), and provided that $\vec{z}_2$ is a list of variables of the size required by this order, it is allowed to go from the configuration $(\mathscr{C}, \emptyset, \mathscr{O}_{\preceq^1}, \vec{z}_1, v)$ to the configuration $(\mathscr{C}, \emptyset, \mathscr{O}_{\preceq^2}, \vec{z}_2, v)$ using the *order change* rule. As explained above, it is clear that this rule preserves (F, f)-validity.

This concludes the presentation of our proof system. Each rule has been shown to preserve (F, f)-validity, and the initial configuration is clearly (F, f)-valid. Therefore, by Theorem 2 our proof system is sound: whenever we can derive a configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ such that $\mathscr{C} \cup \mathscr{D}$ contains $0 \geq 1$, it holds that v is the value of f in any f-minimal solution of F (or, for a decision problem, we have $v < \infty$ precisely when F is satisfiable). As mentioned above, in this case the full sequence of configurations $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ together with annotations about the derivation steps—including, in particular, any witnesses ω—contains all information needed to efficiently reconstruct such an f-minimal solution of F. It is also straightforward to show that our proof system is complete: after using the bound update rule to log an optimal solution v^*, it follows from the implicational completeness of cutting planes that contradiction can be derived from $F \cup \{f \leq v^* - 1\}$.

4 Applications

We now exhibit three applications that have not previously admitted efficient certification, and demonstrate that our new method can support simple, practical proof logging in each case. We first show that, by enhancing the *BreakID* tool for SAT solving (Devriendt et al. 2016) with *VeriPB* proof logging, we can cover the entire solving toolchain when symmetries are involved. We then revisit the Crystal Maze example from the introduction. Finally, we discuss how dominance-based strengthening can be used to support vertex domination reasoning in a maximum clique solver. All code for our implementations and experiments, as well as data and scripts for all plots, can be found at https://doi.org/10.5281/zenodo.6373986.

Symmetry Breaking in SAT Solvers

Symmetry handling has a long and successful history in SAT solving, with a wide variety of techniques considered by, e.g., Aloul, Sakallah, and Markov (2006); Benhamou and Saïs (1994); Benhamou et al. (2010); Devriendt et al. (2012); Devriendt, Bogaerts, and Bruynooghe (2017); Metin, Baarir, and Kordon (2019); Sabharwal (2009). These techniques were used to great effect in, e.g., the 2013 and 2016 editions of the *SAT competition*,[2] where the *SAT+UNSAT hard combinatorial track* and the *no-limit* track, respectively, were won by solvers employing symmetry breaking. However, the victory in 2013 can partly be explained by a small parser bug. For reasons such as this, proof logging is now obligatory in the main track of the SAT competition. While it is hard to overemphasize the importance of this development, it unfortunately means that symmetry breaking can no longer be used, since there is no way of efficiently certifying the correctness of such reasoning in *DRAT*. We will now explain how pseudo-Boolean reasoning with the dominance rule can provide proof logging for the *static symmetry breaking* techniques of Devriendt et al. (2016).

Let π be a permutation of the set of literals in a given CNF formula F (i.e., , a bijection on the set of literals), extended to (sets of) clauses in the obvious way. We say that π is a *symmetry* of F if it commutes with negation, i.e., $\pi(\overline{\ell}) = \overline{\pi(\ell)}$, and preserves satisfaction of F, i.e., $\alpha \circ \pi$ satisfies F if and only if α does. A *syntactic symmetry* in addition satisfies that $\pi(F) \doteq F{\upharpoonright}_{\pi} \doteq F$. As is standard, we only consider syntactic symmetries.

The most common way of breaking symmetries is by adding *lex-leader constraints* (Crawford et al. 1996). We

[2] www.satcompetition.org

here use $\preceq_{\text{lex}}$ to denote the lexicographic order on assignments induced by the sequence of variables $x_1, \ldots, x_m$. Given a set G of symmetries of F, a lex-leader constraint is a formula ψ_{LL} such that α satisfies ψ_{LL} if and only if $\alpha \preceq_{\text{lex}} \alpha \circ \pi$ for each $\pi \in G$. Let $\{x_{i_1}, \ldots, x_{i_n}\}$ be the *support* of π (i.e., all variables x such that $\pi(x) \neq x$), ordered so that $i_j \leq i_k$ if and only if $j \leq k$. Then the constraints

$$y_0 \geq 1 \tag{12a}$$

$$\overline{y}_{j-1} + \overline{x}_{i_j} + \pi(x_{i_j}) \geq 1 \qquad 1 \leq j \leq n \tag{12b}$$

$$\overline{y}_j + y_{j-1} \geq 1 \qquad 1 \leq j < n \tag{12c}$$

$$\overline{y}_j + \overline{\pi(x_{i_j})} + x_{i_j} \geq 1 \qquad 1 \leq j < n \tag{12d}$$

$$y_j + \overline{y}_{j-1} + \overline{x}_{i_j} \geq 1 \qquad 1 \leq j < n \tag{12e}$$

$$y_j + \overline{y}_{j-1} + \pi(x_{i_j}) \geq 1 \qquad 1 \leq j < n \tag{12f}$$

form a lex-leader constraint for π, where each y_j is a fresh variable representing that α and $\alpha \circ \pi$ are equal up to x_{i_j}, and where (12b) does the actual breaking.

To derive this in our proof system, assume that we have a configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{x}, v)$ where assignments are compared lexicographically on $\vec{x} = \{x_1, \ldots, x_m\}$ according to $\mathcal{O}_{\preceq}$ as in (11). Let π be a syntactic symmetry of $\mathscr{C}$ (i.e., such that $\mathscr{C}\!\upharpoonright_{\pi} \doteq \mathscr{C}$) with support contained in $\vec{x}$. In this case

$$C_{LL} \doteq \textstyle\sum_{i=1}^{m} 2^{m-i} \cdot (\pi(x_i) - x_i) \geq 0 \tag{13}$$

expresses that $\pi(\vec{x})$ is greater than or equal to $\vec{x}$. Noting that SAT problems lack an objective function, we can apply the dominance rule with $\omega = \pi$ to derive C_{LL}. To see that (7a) holds, we note that $\neg C_{LL}$ expresses that $\vec{x}$ is strictly larger than $\pi(\vec{x})$, and hence this implies $\mathscr{O}_{\preceq}(\vec{x}\!\upharpoonright_{\pi}, \vec{x})$. Clearly, (7b) is true as well, since its premise contains both C_{LL} and its negation. Since the y-variables are fresh, we can also derive the constraints (12a) and (12c)–(12f) as explained by Gocht and Nordström (2021). It remains to show how to deduce the constraints (12b) from C_{LL}.

As before, assume that the support of π is $\{x_{i_1}, \ldots, x_{i_n}\}$ with $i_j \leq i_k$ if and only if $j \leq k$. Note first that for all x_i that are not in the support of π, the term $\pi(x_i) - x_i$ disappears since $\pi(x_i) = x_i$ and thus C_{LL} simplifies to

$$\textstyle\sum_{j=1}^{n} 2^{m-i_j} \cdot (\pi(x_{i_j}) - x_{i_j}) \geq 0 \ , \tag{14}$$

which can only hold if the term with the largest coefficient is non-negative. It follows that C_{LL} implies $\pi(x_{i_1}) - x_{i_1} \geq 0$ by reverse unit propagation (RUP), and hence can be derived from our current configuration with the implicational rule, also yielding the weaker constraint (12b) with $j = 1$.

To deal with $j > 1$, we define

$$C_{LL}(0) \doteq C_{LL} \tag{15a}$$

$$C_{LL}(k) \doteq C_{LL}(k-1) + 2^{m-i_k} \cdot (12d\,[j = k]) \tag{15b}$$

where $(12d\,[j = k])$ denotes substitution of j by k in (12d). Simplifying $C_{LL}(k)$ yields

$$\textstyle\sum_{i=1}^{k} 2^{m-i}\overline{y_j} + \sum_{i=k+1}^{m} 2^{m-i} \cdot (\pi(x_i) - x_i) \geq 0 \ , \tag{16}$$

which, in combination with all constraints (12c), directly entails (12b) with $j = k$. To see this, note that if y_k is false, then (12b) is trivially true for $j = k + 1$. On the other hand, if y_k is true, then so are all the preceding y-variables, and the dominant term in $C_{LL}(k)$ becomes $\pi(x_{i_k}) - x_{i_k}$, which implies (12b) for $j = k$ analogously to the case for $j = 1$.

It is important to note here that the order is set once and is the same for all symmetries $\pi \in G$ to be broken. Since constraints are added only to $\mathscr{D}$, dominance rule applications for different symmetries will not interfere with each other. Furthermore, contrary to the symmetry logging approach of Heule, Hunt Jr., and Wetzler (2015), handling a symmetry once is enough to guarantee complete breaking. See Appendix C for a worked-out *VeriPB* example of symmetry breaking together with explanations of how the proof logging syntax matches rules in our proof system.

To validate our approach, we implemented *VeriPB* proof logging for the symmetry breaking method in *BreakID*, and modified *Kissat*[3] to output *VeriPB*-proofs (since the redundance rule is a generalization of the RAT rule, this required only minor changes). We used a simpler version of the deletion rule that only guarantees to prove a lower bound on the objective value—if this lower bound is infinity, this certifies that decision problems are unsatisfiable (see the discussion of *weak* (F, f)*-validity* in Appendix B).

Out of all the benchmark instances from all the SAT competitions since 2016, we selected all instances in which at least one symmetry was detected; there were 1089 such instances in total. We performed our experiments on machines with dual Intel Xeon E5-2697A v4 processors with 512GBytes RAM and solid-state drive (SSD), running Ubuntu 20.04. We ran twenty instances in parallel on each machine, limiting each instance to 16GBytes RAM, and with a timeout of 5,000s for solving and 100,000s for verification.

The left plot in Figure 2 displays the performance overhead for symmetry breaking, comparing for each instance the running time with and without proof logging. For most instances, the overhead is negligible (99% of instances are at most 32% slower). The other two plots in Figure 2 display the relationship between the time needed to generate a proof (both for SAT and UNSAT instances) and to verify the correctness of this proof. When only considering verification of the symmetry breaking (middle plot), 1058 instances out of 1089 could be verified, 2 timed out, and 29 terminated due to running out of memory. 75% of the instances could be verified within 3.2 times the solving time and 95% within a factor 20. The time needed for verification is thus considerably longer than solving time, but still practical in the majority of cases. After symmetry breaking, 721 instances could be solved with the SAT solver (right plot) and we could verify 671 instances, while for 33 instances verification timed out and for 17 instances the verifier ran out of memory. Notably, 84 instances could only be solved with symmetry breaking, out of which we could verify 81.

Symmetries in Constraint Programming

In the general setting considered in constraint programming, we must deal with variables with larger (non-Boolean) do-

[3]http://fmv.jku.at/kissat/

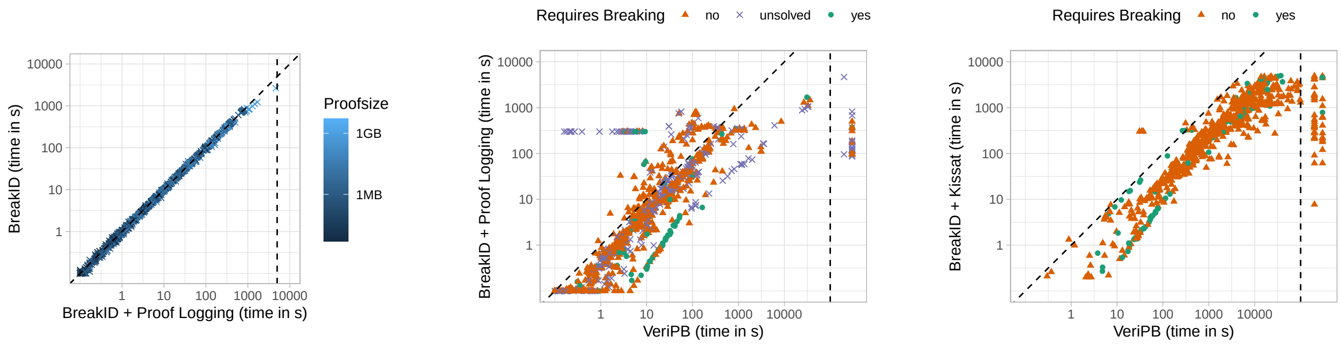

Figure 2: On the left, performance overhead due to proof logging symmetry breaking. In the center, performance of verifying symmetry breaking. On the right, performance of verifying symmetry breaking and SAT solving. Points behind the vertical dashed line indicate timeouts (left) and out of memory (right).

mains and with rich constraints supported by propagation algorithms. One might think that a proof system based upon Boolean variables and linear inequalities would not be suitable for this larger class of problem. However, Elffers et al. (2020) showed how to use *VeriPB* for constraint satisfaction problems by first encoding variables and constraints in pseudo-Boolean form, and then constructing cutting planes proofs to justify the behaviour of propagators such as *alldifferent*. Similarly, the work we present here can also be applied to constraint satisfaction and optimisation problems.

Recall the symmetry breaking constraints proposed for the Crystal Maze puzzle in the introduction. Given the difficulties in knowing which combinations of constraints are valid, it would be desirable if these constraints could be *introduced as part of a proof*, rather than taken as axioms. This would give a modeller immediate feedback as to whether the constraints have been chosen correctly. Our proof system is indeed powerful enough to express all three of the examples we presented, and we have implemented a small tool which can write out the appropriate proof fragments; this allows the entire Crystal Maze example to be verified with *VeriPB*. Interestingly, although symmetries can be broken in different ways in high-level CP models (including through lexicographic and value precedence constraints), when we encode the problem in pseudo-Boolean form these differences largely disappear, and after creating a suitable order we can re-use the SAT techniques just discussed. So, although a full proof-logging constraint solver does not yet exist, we can confidently claim that symmetries no longer block this goal.

Lazy Global Domination in Maximum Clique

Gocht et al. (2020) showed how *VeriPB* can be used to implement proof logging for a wide range of maximum clique algorithms, observing that the cutting planes proof system is rich enough to justify a wide range of bound and inference functions used by various solvers (despite cutting planes not knowing what a graph or clique is). However, there is one clique-solving technique in the literature that is *not* amenable to cutting planes reasoning. In order to solve problem instances that arise from a distance-relaxed clique-finding problem, McCreesh and Prosser (2016) enhanced their maximum clique algorithm with a *lazy global domination* rule that works as follows. Suppose that the solver has constructed a candidate clique C and is considering to extend C by two vertices v and w, where the neighbourhood of v excluding w is a (non-strict) superset of the neighbourhood of w excluding v. Then if the solver first tries v and rejects it, there is no need to branch on w as well.

In principle, it should be possible to introduce additional constraints justifying this kind of reasoning in advance using redundance-based strengthening, without the need for the full dominance breaking framework in Section 3 (with some technicalities involving consistent orderings for tiebreaking). However, due to the prohibitive cost of computing the full vertex dominance relation in advance, McCreesh and Prosser instead implement a form of *lazy* dominance detection, which only triggers following a backtrack.

To provide proof logging for this, we must instead be able to introduce vertex dominance constraints precisely when they are used. It is hard to see how to achieve this with the redundance rule, but it is possible using dominance-based strengthening: we have implemented this in the proof logging maximum clique solver in (Gocht et al. 2020), as discussed in more detail in Appendix E.

5 Conclusion

In this paper, we show that the pseudo-Boolean proof logging method in *VeriPB* (Gocht and Nordström 2021) can be extended with a rule for dominance breaking so as to efficiently certify unlimited symmetry breaking in SAT solving, even when combined with XOR and cardinality reasoning. A natural next question is whether our method is strong enough to capture other techniques such as those used for MaxSAT; several such techniques, such as the *dominating unit-clause rule* (Niedermeier and Rossmanith 2000) and *group subsumed label elimination* (Leivo, Berg, and Järvisalo 2020), appear to be special cases of dominance, making this a promising direction. Our work also contributes towards extending proof logging techniques from SAT to other combinatorial solving paradigms such as constraint programming and dedicated graph solving algorithms.

Acknowledgements

Bart Bogaerts was partially supported by Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G070521N). Ciaran McCreesh was supported by a Royal Academy of Engineering Research Fellowship. Stephan Gocht and Jakob Nordström were supported by the Swedish Research Council grant 2016-00782, and Jakob Nordström also received funding from the Independent Research Fund Denmark grant 9040-00389B. Part of this work was carried out while taking part in the semester program *Satisfiability: Theory, Practice, and Beyond* in the spring of 2021 at the Simons Institute for the Theory of Computing at UC Berkeley.

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.

Akgün, Ö.; Gent, I. P.; Jefferson, C.; Miguel, I.; and Nightingale, P. 2018. Metamorphic Testing of Constraint Solvers. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP '18)*, volume 11008 of *Lecture Notes in Computer Science*, 727–736. Springer.

Alkassar, E.; Böhme, S.; Mehlhorn, K.; Rizkallah, C.; and Schweitzer, P. 2011. An Introduction to Certifying Algorithms. *it - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 53(6): 287–293.

Aloul, F. A.; Sakallah, K. A.; and Markov, I. L. 2006. Efficient symmetry breaking for Boolean satisfiability. *IEEE Transactions on Computers*, 55(5): 549–558.

Baptiste, P.; and Pape, C. L. 1997. Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problems. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP '97)*, volume 1330 of *Lecture Notes in Computer Science*, 375–389. Springer.

Benhamou, B.; Nabhani, T.; Ostrowski, R.; and Saïdi, M. R. 2010. Enhancing Clause Learning by Symmetry in SAT Solvers. In *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI '10)*, volume 1, 329–335.

Benhamou, B.; and Saïs, L. 1994. Tractability through Symmetries in Propositional Calculus. *Journal of Automated Reasoning*, 12(1): 89–102.

Biere, A. 2006. TraceCheck. http://fmv.jku.at/tracecheck/.

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.

Brummayer, R.; Lonsing, F.; and Biere, A. 2010. Automated Testing and Debugging of SAT and QBF Solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, 44–57. Springer.

Bulhões, T.; Sadykov, R.; and Uchoa, E. 2018. A Branch-and-price Algorithm for the Minimum Latency Problem. *Computers & Operations Research*, 93: 66–78.

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

Buss, S. R.; and Thapen, N. 2019. DRAT Proofs, Propagation Redundancy, and Extended Resolution. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT '19)*, volume 11628 of *Lecture Notes in Computer Science*, 71–89. Springer.

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.

Cook, W.; Koch, T.; Steffy, D. E.; and Wolter, K. 2013. A Hybrid Branch-and-Bound Approach for Exact Rational Mixed-Integer Programming. *Mathematical Programming Computation*, 5(3): 305–344.

Crawford, J. M.; Ginsberg, M. L.; Luks, E. M.; and Roy, A. 1996. Symmetry-Breaking Predicates for Search Problems. In *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR '96)*, 148–159.

Cruz-Filipe, L.; Heule, M. J. H.; Hunt, W. A.; Kaufmann, M.; and Schneider-Kamp, P. 2017. Efficient Certified RAT Verification. In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, 220–236. Springer.

Cruz-Filipe, L.; Marques-Silva, J.; and Schneider-Kamp, P. 2017. Efficient Certified Resolution Proof Checking. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '17)*, volume 10205 of *Lecture Notes in Computer Science*, 118–135. Springer.

Demeulemeester, E. L.; and Herroelen, W. S. 2002. *Project Scheduling: A Research Handbook*, volume 49 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers.

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.

Devriendt, J.; Bogaerts, B.; De Cat, B.; Denecker, M.; and Mears, C. 2012. Symmetry Propagation: Improved Dynamic

Symmetry Breaking in SAT. In *Proceedings of the 24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '12)*, 49–56.

Eén, N.; and Biere, A. 2005. Effective Preprocessing in SAT Through Variable and Clause Elimination. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT '05)*, volume 3569 of *Lecture Notes in Computer Science*, 61–75. Springer.

Elffers, J.; Gocht, S.; McCreesh, C.; and Nordström, J. 2020. Justifying All Differences Using Pseudo-Boolean Reasoning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, 1486–1494.

Garcia de la Banda, M.; Stuckey, P. J.; Van Hentenryck, P.; and Wallace, M. 2014. The future of optimization technology. *Constraints*, 19(2): 126–138.

Gebser, M.; Kaminski, R.; and Schaub, T. 2011. Complex Optimization in Answer Set Programming. *Theory and Practice of Logic Programming*, 11(4–5): 821–839.

Gent, I. P.; Petrie, K. E.; and Puget, J. 2006. Symmetry in Constraint Programming. In Rossi, F.; van Beek, P.; and Walsh, T., eds., *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, 329–376. Elsevier.

Gillard, X.; Schaus, P.; and Deville, Y. 2019. SolverCheck: Declarative Testing of Constraints. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, volume 11802 of *Lecture Notes in Computer Science*, 565–582. Springer.

Gocht, S.; McBride, R.; McCreesh, C.; Nordström, J.; Prosser, P.; and Trimble, J. 2020. Certifying Solvers for Clique and Maximum Common (Connected) Subgraph Problems. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, 338–357. Springer.

Gocht, S.; McCreesh, C.; and Nordström, J. 2020. Subgraph Isomorphism Meets Cutting Planes: Solving With Certified Solutions. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, 1134–1140.

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.

Goldberg, E.; and Novikov, Y. 2003. Verification of Proofs of Unsatisfiability for CNF Formulas. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03)*, 886–891.

Heule, M. J. H.; Hunt Jr., W. A.; and Wetzler, N. 2013a. Trimming While Checking Clausal Proofs. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13)*, 181–188.

Heule, M. J. H.; Hunt Jr., W. A.; and Wetzler, N. 2013b. Verifying Refutations with Extended Resolution. In *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, 345–359. Springer.

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.

Heule, M. J. H.; Kiesl, B.; and Biere, A. 2017. Short Proofs Without New Variables. In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, 130–147. Springer.

Hoogeboom, M.; Dullaert, W.; Lai, D.; and Vigo, D. 2020. Efficient Neighborhood Evaluations for the Vehicle Routing Problem with Multiple Time Windows. *Transportation Science*, 54(2): 400–416.

Jouglet, A.; and Carlier, J. 2011. Dominance Rules In Combinatorial Optimization Problems. *European Journal of Operational Research*, 212(3): 433–444.

Leivo, M.; Berg, J.; and Järvisalo, M. 2020. Preprocessing in Incomplete MaxSAT Solving. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI '20)*, 347–354.

McConnell, R. M.; Mehlhorn, K.; Näher, S.; and Schweitzer, P. 2011. Certifying Algorithms. *Computer Science Review*, 5(2): 119–161.

McCreesh, C.; and Prosser, P. 2016. Finding Maximum k-Cliques Faster Using Lazy Global Domination. In *Proceedings of the 9th Annual Symposium on Combinatorial Search (SOCS '16)*, 72–80.

Metin, H.; Baarir, S.; and Kordon, F. 2019. Composing Symmetry Propagation and Effective Symmetry Breaking for SAT Solving. In *Proceedings of the 11th International NASA Formal Methods Symposium (NFM '19)*, volume 11460 of *Lecture Notes in Computer Science*, 316–332. Springer.

Niedermeier, R.; and Rossmanith, P. 2000. New Upper Bounds for Maximum Satisfiability. *Journal of Algorithms*, 36(1): 63–88.

Sabharwal, A. 2009. SymChaff: Exploiting Symmetry in a Structure-Aware Satisfiability Solver. *Constraints*, 14(4): 478–505. Preliminary version in *AAAI '05*.

Tchinda, R. K.; and Djamégni, C. T. 2020. On Certifying the UNSAT Result of Dynamic Symmetry-Handling-Based SAT Solvers. *Constraints*, 25(3–4): 251–279.

Walsh, T. 2012. Symmetry Breaking Constraints: Recent Results. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI '12)*, 2192–2198.

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 Internatjuional Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, 422–429. Springer.

A Introduction to the Technical Appendices

In this collection of technical appendices, we provide some material that had to be omitted in the main text due to space constraints.

In Appendix B, we provide the missing proofs in the formal exposition of our cutting planes proof system with dominance. In Appendix C, we provide details about how our proof logging technique for SAT symmetry breaking works, including a discussion of various optimizations of the basic symmetry breaking algorithm and how we deal with them. In Appendix D we discuss how the methods developed in our work can be used to certify the correctness of added symmetry breaking constraints in constraint programming, and Appendix E contains a fairly detailed discussion of how to certify vertex dominance breaking in a maximum clique solver.

B A Proof System for Dominance Breaking

In this appendix, we give a full, formal presentation of our proof system for verifying dominance breaking, which we have implemented on top of the tool *VeriPB* as developed in the sequence of papers (Elffers et al. 2020; Gocht, McCreesh, and Nordström 2020; Gocht et al. 2020; Gocht and Nordström 2021). In order to give a self-contained presentation, this section essentially copies the material from Section 3, inserting all formal proofs where they belong.

We remark that for applications it is absolutely crucial not only that the proof system be sound, but that all proofs be efficiently machine-verifiable. There are significant challenges involved in making proof logging and verification efficient, but in this section we mostly ignore these aspects of our work and focus on the theoretical underpinnings.

Our foundation is the cutting planes proof system described in Section 2. However, in a proof in our system for (F, f), where f is a linear objective function to be minimized under the pseudo-Boolean formula F (or where $f \doteq 0$ for decision problems), we also allow strengthening F by adding constraints C that are not implied by the formula. Pragmatically, adding C should be in order as long as we keep some optimal solution, i.e., a satisfying assignment to F that minimizes f, which we will refer to as an *f-minimal solution* of F. We will formalize this idea by allowing the use of an additional pseudo-Boolean formula $\mathscr{O}_{\preceq}(\vec{u}, \vec{v})$ that, together with a sequence of variables $\vec{z}$, defines a relation $\alpha \preceq \beta$ to hold between assignments α and β if $\mathscr{O}_{\preceq}(\vec{z}{\restriction_\alpha}, \vec{z}{\restriction_\beta})$ evaluates to true. We require (a cutting planes proof) that $\mathscr{O}_{\preceq}$ is such that this defines a preorder, i.e., a reflexive and transitive relation. Adding new constraints C will be valid as long as we guarantee to preserve some f-minimal solution that is also minimal with respect to $\preceq$. In other words, $\preceq$ can be combined with f to define a preorder $\preceq_f$ on assignments by

$$\alpha \preceq_f \beta \quad \text{if} \quad \alpha \preceq \beta \text{ and } f{\restriction_\alpha} \leq f{\restriction_\beta} \ , \tag{17}$$

and we require that all derivation steps in the proof should preserve some solution that is minimal with respect to $\preceq_f$. The preorder defined by $\mathscr{O}_{\preceq}(\vec{u}, \vec{v})$ will only become important once we introduce our new *dominance-based strengthening rule* later in this section. For simplicity, up until that point the reader can assume that the pseudo-Boolean formula is $\mathscr{O}_{\top} \doteq \emptyset$ inducing the trivial preorder relating all assignments, though all proofs presented below work in full generality for the orders that will be introduced later.

A proof for (F, f) in our proof system consists of a sequence of *proof configurations* $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$, where

- $\mathscr{C}$ is a set of pseudo-Boolean *core constraints*;
- $\mathscr{D}$ is another set of pseudo-Boolean *derived constraints*;
- $\mathscr{O}_{\preceq}$ is a PB formula encoding a preorder and $\vec{z}$ a set of literals on which this preorder will be applied; and
- v is the best value found so far for f.

The initial configuration is $(F, \emptyset, \mathscr{O}_{\top}, \emptyset, \infty)$. The distinction between $\mathscr{C}$ and $\mathscr{D}$ is only relevant when a nontrivial preorder is used; we will elaborate on this when discussing dominance. The intended semantics of f and v is that if $v < \infty$, then there exists a solution α satisfying F such that $f{\restriction_\alpha} \leq v$, and in this case the proof can make use of the constraint $f \leq v - 1$ in the search for better solutions. As long as the optimal solution has not been found, it should hold that f-minimal solutions of $\mathscr{C} \cup \mathscr{D}$ have the same objective value as f-minimal solutions of F. The precise relation is formalized in the notion of *valid configurations* as defined next.

Definition 6. *A configuration* $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ *is* (F, f)-valid *if the following conditions hold:*

1. *If $v < \infty$, then there is a total assignment ρ satisfying F such that $f{\restriction_\rho} \leq v$.*
2. *For every $v' < v$, it holds that the sets $F \cup \{f \leq v'\}$ and $\mathscr{C} \cup \{f \leq v'\}$ are equisatisfiable.*
3. *For every total assignment ρ satisfying the constraints $\mathscr{C} \cup \{f \leq v - 1\}$, there exists a total assignment $\rho' \preceq_f \rho$ satisfying $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v - 1\}$.*

We will show that (F, f)-validity is an invariant of our proof system, i.e., that it is preserved by all derivation rules. Note that the two last items together imply that if the configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ is such that v is not yet the value of an optimal solution, then f-minimal solutions of F and of $\mathscr{C} \cup \mathscr{D}$ have the same objective value, just as desired.

A proof in our proof system ends when the configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v^*)$ is such that $\mathscr{C} \cup \mathscr{D}$ contains contradiction $\bot \doteq 0 \geq 1$. In that case, either $v^* = \infty$ and F is unsatisfiable, or v^* is the optimal value (or $v^* = 0$ for a satisfiable decision problem). We state this as a formal theorem.

Theorem 7. *Let F be a pseudo-Boolean formula and f an objective function. If $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v^*)$ is an (F, f)-valid configuration with $\{0 \geq 1\} \subseteq \mathscr{C} \cup \mathscr{D}$, then*

- *F is unsatisfiable if and only if $v^* = \infty$; and*
- *if F is satisfiable, then there is an f-minimal solution α of F with objective value $f{\restriction_\alpha} = v^*$.*

Proof. If F is unsatisfiable, then we must have $v^* = \infty$ due to item 1 of (F, f)-validity.

If F is satisfiable, let α be an f-optimal assignment of F. We will show that $v^* = f{\restriction_\alpha}$. Clearly, $v^* \geq f{\restriction_\alpha}$, otherwise item 1 of (F, f)-validity would yield a strictly better assignment than α, contradicting optimality. If $v^* > f{\restriction_\alpha}$, then α satisfies $F \cup \{f \leq v^* - 1\}$. Hence, item 2 yields an α' that satisfies $\mathscr{C} \cup \{f \leq v^* - 1\}$ and item 3 an α'' that satisfies $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v^* - 1\}$, which contradicts the assumption that $0 \geq 1$ is in $\mathscr{C} \cup \mathscr{D}$. It follows that $v^* \leq f\!\restriction_\alpha$ and thus $v^* = f\!\restriction_\alpha$, as desired. □

We are now ready to give a formal description of the rules in our proof system and argue that these rules preserve (F, f)-validity.

Implicational Derivation Rule

If we can exhibit a derivation of the pseudo-Boolean constraint C from $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v - 1\}$ in our (slightly extended) version of cutting planes as described in Section 2 (i.e., in formal notation, if $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v-1\} \vdash C$), then we can go from the configuration $(\mathscr{C}, \mathscr{D}, \mathcal{O}_\preceq, \vec{z}, v)$ to the configuration $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathcal{O}_\preceq, \vec{z}, v)$ by the *implicational derivation rule*. By the soundness of the cutting planes proof system, this means that $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v - 1\} \models C$, and so (F, f)-validity is preserved, but, more importantly, the cutting planes derivation provides a simple and efficient way for an algorithm to *verify* that this implication holds. This is a key feature of all rules in our proof system—not only are they sound, but the soundness of every rule application can be efficiently verified by checking a simple, syntactic object.

When doing proof logging, the solver would need to specify by which sequence of cutting planes derivation rules C was obtained. For practical purposes, though, it greatly simplifies matters that in many cases the verifier can figure out the required proof details automatically, meaning that the proof logger can just state the desired constraint without any further information. One important example of this is when C is a reverse unit propagation (RUP) constraint with respect to $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v - 1\}$. Another case is when C is literal-axiom-implied by some other constraint.

Objective Bound Update Rule

The *objective bound update rule* allows improving the estimate of what value can be achieved for the objective function f. We can go from $(\mathscr{C}, \mathscr{D}, \mathcal{O}_\preceq, \vec{z}, v)$ to $(\mathscr{C}, \mathscr{D}, \mathcal{O}_\preceq, \vec{z}, v')$ if we know an assignment α satisfying $\mathscr{C}$ such that $f\!\restriction_\alpha = v' < v$. When actually doing proof logging, the solver would specify such an assignment α, which would then be checked by the proof verifier (in our case *VeriPB*).

To argue that this rule preserves (F, f)-validity, we note that the last two items are trivially satisfied (they are weaker after applying the rule than before). The first item is satisfied since item 2 guarantees the existence of an α' satisfying F with an objective value that is at least as good as v'. Note that we have no guarantee that α' will be a solution to F. However, although we will not emphasize this point here, it follows from our formal treatment below that the proof system guarantees that such an f-minimal solution α' to the original formula F can be efficiently reconstructed from the proof (where efficiency is measured in the size of the proof).

Redundance-Based Strengthening Rule

The redundance-based strengthening rule allows deriving a constraint C from $\mathscr{C} \cup \mathscr{D}$ even if C is not implied, provided that it can be shown that any assignment α that satisfies $\mathscr{C} \cup \mathscr{D}$ can be transformed into another assignment $\alpha' \preceq_f \alpha$ that satisfies both $\mathscr{C} \cup \mathscr{D}$ and C (in case $\mathcal{O}_\preceq = \mathcal{O}_\top$, the condition $\alpha' \preceq_f \alpha$ just means that $f\!\restriction_{\alpha'} \leq f\!\restriction_\alpha$). This rule is borrowed from (Gocht and Nordström 2021), which in turn relies heavily on (Heule, Kiesl, and Biere 2017; Buss and Thapen 2019). We extend this rule here from decision problems to optimization problems in the natural way.

Formally, we say that C can be derived from $(\mathscr{C}, \mathscr{D}, \mathcal{O}_\preceq, \vec{z}, v)$ by *redundance-based strengthening*, or just *redundance* for brevity, if there is a substitution ω (which we will refer to as the *witness*) such that

$$\begin{aligned}&\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \\ &\quad (\mathscr{C} \cup \mathscr{D} \cup C)\!\restriction_\omega \cup \{f\!\restriction_\omega \leq f\} \cup \mathcal{O}_\preceq(\vec{z}\!\restriction_\omega, \vec{z}) \ .\end{aligned} \tag{18}$$

Intuitively, (18) says that if some assignment α satisfies $\mathscr{C} \cup \mathscr{D}$ but falsifies C, then $\alpha' = \alpha \circ \omega$ still satisfies $\mathscr{C} \cup \mathscr{D}$ and also satisfies C. In addition, the condition $f\!\restriction_\omega \leq f$ ensures that $\alpha \circ \omega$ achieves an objective function value that is at least as good as that for α. This together with the constraints $\mathcal{O}_\preceq(\vec{z}\!\restriction_\omega, \vec{z})$ guarantees that $\alpha' \preceq_f \alpha$. For proof logging purposes, the witness ω as well as any non-immediate cutting planes derivations of constraints on the right-hand side of (18) would have to be specified, but, e.g., all RUP constraints or literal-axiom-implied constraints can be left to the verifier to check.

Proposition 8. *If C is derivable from an (F, f)-valid configuration $(\mathscr{C}, \mathscr{D}, \mathcal{O}_\preceq, \vec{z}, v)$ by redundance-based strengthening, then $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathcal{O}_\preceq, \vec{z}, v)$ is (F, f)-valid as well.*

Proof. Items 1 and 2 of (F, f)-validity remain satisfied since F, v, and $\mathscr{C}$ are unchanged. Our proof of item 3 extends proofs of similar properties for decision problems (Heule, Kiesl, and Biere 2017; Buss and Thapen 2019; Gocht and Nordström 2021). Consider an assignment ρ satisfying $\mathscr{C} \cup \{f \leq v - 1\}$. We will construct an assignment $\rho' \preceq_f \rho$, i.e., such that $f\!\restriction_{\rho'} \leq f\!\restriction_\rho$ and $\mathcal{O}_\preceq(\vec{z}\!\restriction_{\rho'}, \vec{z}\!\restriction_\rho)$ hold, that also satisfies $\mathscr{C} \cup \mathscr{D} \cup \{C\}$.

Without loss of generality (due to item 3 of (F, f)-validity, which holds for the first configuration), we can assume that ρ also satisfies $\mathscr{D}$. If ρ satisfies C, then we use $\rho' = \rho$ and all conditions are satisfied (recall that $\mathcal{O}_\preceq$ induces a preorder, and hence a reflexive relation: for any ρ, $\mathcal{O}_\preceq(\vec{z}\!\restriction_\rho, \vec{z}\!\restriction_\rho)$ holds). Otherwise, choose $\rho' = \rho \circ \omega$. We know that ρ satisfies $\mathscr{C} \cup \mathscr{D} \cup \neg C$, and hence by (18) ρ also satisfies

$$(\mathscr{C} \cup \mathscr{D} \cup C)\!\restriction_\omega \cup \{f\!\restriction_\omega \leq f\} \cup \mathcal{O}_\preceq(\vec{z}\!\restriction_\omega, \vec{z}) \ . \tag{19}$$

Clearly, for any constraint D, it holds that $(D\!\restriction_\omega)\!\restriction_\rho = D\!\restriction_{\rho\circ\omega}$ and thus if ρ satisfies $D\!\restriction_\omega$, then $\rho' = \rho \circ \omega$ satisfies D. Therefore, ρ' satisfies $\mathscr{C} \cup \mathscr{D} \cup C$. Additionally, ρ satisfies $\{f\!\restriction_\omega \leq f\}$, and hence $f\!\restriction_{\rho'} \leq f\!\restriction_\rho$. Similarly, ρ satisfies $\mathcal{O}_\preceq(\vec{z}\!\restriction_\omega, \vec{z})$ and $(\mathcal{O}_\preceq(\vec{z}\!\restriction_\omega, \vec{z}))\!\restriction_\rho \doteq \mathcal{O}_\preceq(\vec{z}\!\restriction_{\rho'}, \vec{z}\!\restriction_\rho)$ thus holds, which concludes our proof. □

Deletion Rule

We also need to be able to delete previously derived constraints. From a configuration $(\mathscr{C}, \mathscr{D}, \mathcal{O}_\preceq, \vec{z}, v)$ we can transition to $(\mathscr{C}', \mathscr{D}', \mathcal{O}_\preceq, \vec{z}, v)$ using the *deletion rule* if

1. $\mathscr{D}' \subseteq \mathscr{D}$ and
2. $\mathscr{C}' = \mathscr{C}$ or $\mathscr{C}' = \mathscr{C} \setminus \{C\}$ for some constraint C derivable via the redundance rule from $(\mathscr{C}', \emptyset, \mathscr{O}_{\preceq}, \vec{z}, v)$.

This last condition above perhaps seems slightly odd, but it is there since deleting arbitrary constraints could violate (F, f)-validity in two different ways. Firstly, it would allow finding better-than-optimal solutions. Secondly, and perhaps surprisingly, in combination with the dominance-based strengthening rule, which we will discuss below, arbitrary deletion is unsound, as it can turn satisfiable instances into unsatisfiable ones. This is illustrated in Example 12 further below.

To see that deletion preserves (F, f)-validity, it is clear that item 1 remains satisfied by deletion, as does the direction of item 2 that claims satisfiability of $\mathscr{C} \cup \{f \leq v'\}$. Now let α be an assignment that satisfies $\mathscr{C}' \cup \{f \leq v'\}$ for $v' < v$; we use this to construct a satisfying assignment α' for $F \cup \{f \leq v'\}$. If $\mathscr{C}' = \mathscr{C}$, we get α' from the (F, f)-validity of the original configuration, so assume $\mathscr{C}' = \mathscr{C} \setminus \{C\}$. If α satisfies C, it satisfies $\mathscr{C}$, and again the claim follows from (F, f)-validity of the original configuration. Assume therefore that α does not satisfy C. Since C is derivable via redundance from $(\mathscr{C}', \emptyset, \mathscr{O}_{\preceq}, \vec{z}, v)$, it holds that

$$\mathscr{C}' \cup \{\neg C\} \vdash (\mathscr{C}' \cup C)\restriction_{\omega} \cup \{f\restriction_{\omega} \leq f\} \cup \mathscr{O}_{\preceq}(\vec{z}\restriction_{\omega}, \vec{z}) \ . \tag{20}$$

This yields an assignment $\alpha'' = \alpha \circ \omega$ satisfying $\mathscr{C} = \mathscr{C}' \cup \{C\}$ such that $f\restriction_{\alpha''} \leq f\restriction_{\alpha} \leq v'$, showing that $\mathscr{C} \cup \{f \leq v'\}$ is satisfiable. Appealing to the (F, f)-validity of the original configuration, we then find an α' with $f\restriction_{\alpha'} \leq v'$ that satisfies F, proving that indeed the second item holds. The proof for item 3 is similar: $\alpha \circ \omega$ satisfies $\mathscr{C}' \cup \{C\}$, and applying (F, f)-validity of the original configuration yields an α' with $\alpha' \preceq_f \alpha \circ \omega \preceq_f \alpha$ that satisfies $\mathscr{D}$.

An alternative to condition 2 would be to enforce the more restrictive demand $\mathscr{C}' \vdash \mathscr{C}$. However, this would prevent the use of some SAT preprocessing techniques such as bounded variable elimination (Eén and Biere 2005).

In practice, checking deletions can make it more difficult to implement proof logging, or could have negative effects on performance. An alternative is to use a more liberal deletion rule, which also allows deleting constraints from $\mathscr{C}$ if $\mathscr{D}$ is empty. In this case, unsatisfiable instances can become satisfiable and better than optimal solutions can be introduced, but we can still verify a lower bound on the best objective value. This means that if the solver provides a solution to the original formula F that matches the verified lower bound, then this solution is guaranteed to be optimal. To prove that the proof system remains sound with this more liberal deletion rule, we need to adjust our invariant.

Definition 9. *A configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ is* weakly (F, f)-valid *if the following conditions hold:*

1. *For every $v' < v$, it holds that if $F \cup \{f \leq v'\}$ is satisfiable then $\mathscr{C} \cup \{f \leq v'\}$ is satisfiable.*
2. *For every total assignment ρ satisfying the constraints $\mathscr{C} \cup \{f \leq v - 1\}$, there is a total assignment $\rho' \preceq_f \rho$ satisfying $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v - 1\}$.*

We will only show that each rule preserves (F, f)-validity, because the same proofs can be used to show that weak (F, f)-validity is preserved as well and while deleting from $\mathscr{C}$ if $\mathscr{D}$ is empty does not preserve (F, f)-validity, it is easy to see that weak (F, f)-validity is preserved. With this weaker invariant, we also get a weaker result for the final configuration.

Theorem 10. *Given a formula F and an objective function f, let $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v^*)$ be a weakly (F, f)-valid configuration with $\{0 \geq 1\} \subseteq \mathscr{C} \cup \mathscr{D}$. It holds that*

- *for any solution α of F we have $f\restriction_{\alpha} \geq v^*$, and especially,*
- *if $v^* = \infty$ then F is unsatisfiable.*

Proof. If F is satisfiable, then let α be a satisfying assignment of F. If $v^* > f\restriction_{\alpha}$, then α satisfies $F \cup \{f \leq v^* - 1\}$. Hence, item 1 yields an α' that satisfies $\mathscr{C} \cup \{f \leq v^* - 1\}$ and item 2 an α'' that satisfies $\mathscr{C} \cup \mathscr{D} \cup \{f \leq v^* - 1\}$, which contradicts the assumption that $0 \geq 1$ is in $\mathscr{C} \cup \mathscr{D}$. It follows that $v^* \leq f\restriction_{\alpha}$. □

Transfer Rule

Constraints can always be moved from the derived set $\mathscr{D}$ to the core set $\mathscr{C}$ using the *transfer rule*, which allows a transition from $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ to $(\mathscr{C}', \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ if $\mathscr{C} \subseteq \mathscr{C}' \subseteq \mathscr{C} \cup \mathscr{D}$. This clearly preserves (F, f)-validity.

The transfer rule together with deletion allows replacing constraints in the original formula with stronger constraints. For example, assume that $x + y \geq 1$ is in $\mathscr{C}$ and that we derive $x \geq 1$. Then we can move $x \geq 1$ from $\mathscr{D}$ to $\mathscr{C}$ and then delete $x + y \geq 1$. The required redundance check $\{x \geq 1, \neg(x + y \geq 1)\} \vdash \perp$ is immediate.

The rules discussed so far do not change $\mathscr{O}_{\preceq}$, and so any derivation using these rules only will operate with the trivial preorder $\mathscr{O}_{\top}$ imposing no conditions. The proof system defined in terms of these rules is a straightforward extension of *VeriPB* as developed in (Elffers et al. 2020; Gocht, McCreesh, and Nordström 2020; Gocht et al. 2020; Gocht and Nordström 2021) to an optimization setting. We next discuss the main contribution of this paper, namely the new dominance rule making use of the preorder $\mathscr{O}_{\preceq}$.

Dominance-Based Strengthening Rule

Any preorder $\preceq$ induces a strict order $\prec$ defined by $\alpha \prec \beta$ if $\alpha \preceq \beta$ and $\beta \not\preceq \alpha$. The relation $\prec_f$ obtained in this way from the preorder (17) coincides with what Chu and Stuckey (2015) call a *dominance relation* in the context of constraint optimisation. Our dominance rule allows deriving a constraint C from $\mathscr{C} \cup \mathscr{D}$ even if C is not implied, similar to the redundance rule. However, for the dominance rule an assignment α satisfying $\mathscr{C} \cup \mathscr{D}$ but falsifying C need only to be mapped to an assignment α' that satisfies $\mathscr{C}$, but not necessarily $\mathscr{D}$ or C. On the other hand, the new assignment α' should satisfy the strict inequality $\alpha' \prec_f \alpha$ and not just $\alpha' \preceq_f \alpha$ as in the redundance rule. To show that this new dominance rule preserves (F, f)-validity, we will prove that it is possible to construct an assignment that satisfies $\mathscr{C} \cup \mathscr{D} \cup \{C\}$ by iteratively applying the witness of the dominance rule, in combination with (F, f)-validity of the configuration before application of the dominance rule. As our base case, if α' satisfies $\mathscr{C} \cup \mathscr{D} \cup \{C\}$, we are done. Otherwise, since α' satisfies $\mathscr{C}$, by (F, f)-validity we are guaranteed the existence of an assignment α'' satisfying $\mathscr{C} \cup \mathscr{D}$ for which $\alpha'' \prec_f \alpha' \prec_f \alpha$ holds. If α'' still does not satisfy C, we can repeat the argument. In this way, we get a strictly decreasing sequence (with respect to $\prec_f$) of assignments. Since the set of possible assignments is finite, this sequence will eventually terminate.

Formally, we can derive C by dominance-based strengthening provided that there exists a substitution ω such that

$$\begin{aligned}&\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \\ &\quad \mathscr{C}\restriction_\omega \cup \mathscr{O}_\preceq(\vec{z}\restriction_\omega, \vec{z}) \cup \neg\mathscr{O}_\preceq(\vec{z}, \vec{z}\restriction_\omega) \cup \{f\restriction_\omega \leq f\} \ ,\end{aligned} \tag{21}$$

where $\mathscr{O}_\preceq(\vec{z}\restriction_\omega, \vec{z})$ and $\neg\mathscr{O}_\preceq(\vec{z}, \vec{z}\restriction_\omega)$ together state that $\alpha \circ \omega \prec \alpha$ for any assignment α. A minor technical problem is that the pseudo-Boolean formula $\mathscr{O}_\preceq(\vec{z}, \vec{z}\restriction_\omega)$ may contain multiple constraints, so that the negation of it is no longer a PB formula. To get around this, we split (21) into two separate conditions and shift $\neg\mathscr{O}_\preceq(\vec{z}, \vec{z}\restriction_\omega)$ to the premise of the implication, which eliminates the negation. Thus, the formal version of our *dominance-based strengthening rule*, or just *dominance rule* for brevity, says that we can go from $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ to $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathscr{O}_\preceq, \vec{z}, v)$ if there is a substitution ω such that the conditions

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \mathscr{C}\restriction_\omega \cup \mathscr{O}_\preceq(\vec{z}\restriction_\omega, \vec{z}) \cup \{f\restriction_\omega \leq f\} \tag{22a}$$

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \cup \mathscr{O}_\preceq(\vec{z}, \vec{z}\restriction_\omega) \vdash \bot \tag{22b}$$

are satisfied. Just as for the redundance rule, the witness ω as well as any non-immediate derivations would have to be specified in the proof log.

Proposition 11. *If C is derivable from an (F, f)-valid configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ by dominance-based strengthening, then $(\mathscr{C}, \mathscr{D} \cup \{C\}, \mathscr{O}_\preceq, \vec{z}, v)$ is also (F, f)-valid.*

Proof. The first two items of (F, f)-validity are clearly satisfied, since F, $\mathscr{C}$, and v are unchanged. Assume towards contradiction that the last item *does not* hold. Let S denote the set of assignments α that (1) satisfy $\mathscr{C} \cup \{f \leq v - 1\}$ and (2) admit no $\alpha' \preceq_f \alpha$ satisfying $\mathscr{C} \cup \mathscr{D} \cup \{C\}$. By our assumption, S is non-empty.

Let α be some $\prec_f$-minimal assignment in S. Since $(\mathscr{C}, \mathscr{D}, \mathscr{O}_\preceq, \vec{z}, v)$ is (F, f)-valid, there exists some $\alpha_1 \preceq_f \alpha$ that satisfies $\mathscr{C} \cup \mathscr{D}$. We know that α_1 cannot satisfy C since $\alpha \in S$. Hence, α_1 satisfies $\mathscr{C} \cup \mathscr{D} \cup \{\neg C\}$. From (22a) it follows that α_1 satisfies $\mathscr{O}_\preceq(\vec{z}\restriction_\omega, \vec{z}) \cup \{f\restriction_\omega \leq f\}$ and thus that $\mathscr{O}_\preceq(\vec{z}\restriction_{\alpha_1 \circ \omega}, \vec{z}\restriction_{\alpha_1})$ and $f\restriction_{\alpha_1 \circ \omega} \leq f\restriction_{\alpha_1}$ are satisfied. In other words, $\alpha_1 \circ \omega \preceq_f \alpha_1$. By (22b), it follows that α_1 does not satisfy $\mathscr{O}_\preceq(\vec{z}, \vec{z}\restriction_\omega)$, i.e., $\mathscr{O}_\preceq(\vec{z}\restriction_{\alpha_1}, \vec{z}\restriction_{\alpha_1 \circ \omega})$ does not hold and thus $\alpha_1 \not\preceq_f \alpha_1 \circ \omega$. Now let α_2 be $\alpha_1 \circ \omega$. We showed that $\alpha_2 \prec_f \alpha_1 \preceq_f \alpha$. Furthermore, since α_1 satisfies $\mathscr{C} \cup \mathscr{D} \cup \{\neg C\}$, (7a) yields that α_2 satisfies $\mathscr{C}$. Thus α_2 satisfies $\mathscr{C} \cup \{f \leq v - 1\}$. Since $\alpha_2 \prec_f \alpha$, and α is a minimal element of S, it cannot be that $\alpha_2 \in S$. Thus, there must exist a $\alpha' \preceq_f \alpha_2$ that satisfies $\mathscr{C} \cup \mathscr{D} \cup \{C\}$. However, it is also so that $\alpha' \preceq_f \alpha$, and since $\alpha \in S$ this means that α' cannot satisfy $\mathscr{C} \cup \mathscr{D} \cup \{C\}$. This yields a contradiction, thereby finishing our proof. ☐

When introducing the deletion rule, we already mentioned that deleting arbitrary constraints can be unsound in combination with dominance-based strengthening. We now illustrate this phenomenon.

Example 12. *Consider the formula $F = \{p \geq 1\}$ with objective $f \doteq 0$ and the configuration*

$$(\mathscr{C}_1 = \{p \geq 1\}, \mathscr{D}_1 = \{p \geq 1\}, \mathscr{O}_\preceq, \{p\}, \infty) \ , \tag{23}$$

where $\mathscr{O}_\preceq(u, v)$ is defined as $\{v + \overline{u} \geq 1\}$. This configuration is (F, f)-valid and $\mathscr{C} \cup \mathscr{D}$ is satisfiable. If *we were allowed to delete constraints arbitrarily from $\mathscr{C}$, we could derive a configuration with $\mathscr{C}_2 = \emptyset$ and $\mathscr{D}_2 = \{p \geq 1\}$. However, now the dominance rule can derive $C \doteq \overline{p} \geq 1$, using the witness $\omega = \{p \mapsto 0\}$. To see that all conditions for applying dominance-based strengthening are indeed satisfied, we notice that conditions* (7a)–(7b) *simplify to*

$$\emptyset \cup \{p \geq 1\} \cup \{p \geq 1\} \vdash \emptyset \cup \{p + 1 \geq 1\} \cup \emptyset \tag{24a}$$

$$\emptyset \cup \{p \geq 1\} \cup \{p \geq 1\} \cup \{0 + \overline{p} \geq 1\} \vdash \bot \tag{24b}$$

respectively. Both claims clearly hold, meaning that we arrive at a configuration that contains both $p \geq 1$ and $\overline{p} \geq 1$.

Preorder Encodings

As mentioned before, $\mathscr{O}_\preceq$ is shorthand for a pseudo-Boolean formula $\mathscr{O}_\preceq(\vec{u}, \vec{v})$ over two sets of formal placeholder variables $\vec{u} = \{u_1, \ldots, u_n\}$ and $\vec{v} = \{v_1, \ldots, v_n\}$ of equal size, which should also match the size of $\vec{z}$ in the configuration. To use $\mathscr{O}_\preceq$ in a proof, it is required to show that this formula encodes a preorder. This is done by providing (in a proof preamble) cutting planes derivations establishing

$$\emptyset \vdash \mathscr{O}_\preceq(\vec{u}, \vec{u}) \tag{25a}$$

$$\mathscr{O}_\preceq(\vec{u}, \vec{v}) \cup \mathscr{O}_\preceq(\vec{v}, \vec{w}) \vdash \mathscr{O}_\preceq(\vec{u}, \vec{w}) \tag{25b}$$

where (25a) formalizes reflexivity and (25b) transitivity (and where notation like $\mathscr{O}_\preceq(\vec{v}, \vec{w})$ is shorthand for applying to $\mathscr{O}_\preceq(\vec{u}, \vec{v})$ the substitution ω that maps u_i to v_i and v_i to w_i, as discussed in Section 2). These two conditions guarantee that the relation $\preceq$ defined by $\alpha \preceq \beta$ if $\mathscr{O}_\preceq(\vec{z}\restriction_\alpha, \vec{z}\restriction_\beta)$ forms a preorder on the set of assignments.

By way of example, to encode the lexicographic order $u_1 u_2 \ldots u_n \preceq_{\text{lex}} v_1 v_2 \ldots v_n$, we can use a single constraint

$$\mathscr{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{v}) \doteq \textstyle\sum_{i=1}^{n} 2^{n-i} \cdot (v_i - u_i) \geq 0 \ . \tag{26}$$

Reflexivity is vacuously true since $\mathscr{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{u}) \doteq 0 \geq 0$, and transitivity also follows easily since adding $\mathscr{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{v})$ and $\mathscr{O}_{\preceq_{\text{lex}}}(\vec{v}, \vec{w})$ yields $\mathscr{O}_{\preceq_{\text{lex}}}(\vec{u}, \vec{w})$ (where we tacitly assume that the constraint resulting from this addition is implicitly simplified by collecting like terms, performing any cancellations, and shifting any constants to the right-hand side of the inequality, as mentioned in Section 2).

A potential concern with encodings such as (26) is that coefficients can become very large as the number of variables in the order grows. It is perfectly possible to address this by allowing order encodings using auxiliary variables in addition to $\vec{u}$ and $\vec{v}$. We have chosen not to develop the theory for this in the current paper, however, since we feel that it makes the exposition unnecessarily complicated without adding anything of real significance to the scientific contribution.

Order Change Rule

The final proof rule that we need is a rule for introducing a nontrivial order, and it turns out that it can also be convenient to be able to use different orders at different points in the proof. Switching orders is possible, but to maintain soundness it is important to first clear the set $\mathscr{D}$ (after transferring the constraints we want to keep to $\mathscr{C}$). The reason for this is simple: if we allow arbitrary order changes, then the third item of (F, f)-validity would no longer hold, but when $\mathscr{D} = \emptyset$, it is trivially true.

Formally, provided that $\mathscr{O}_{\preceq^2}$ has been established to be a preorder (via cutting planes proofs for (25a) and (25b)), and provided that $\vec{z}_2$ is a list of variables of the size required by this order, it is allowed to go from the configuration $(\mathscr{C}, \emptyset, \mathscr{O}_{\preceq^1}, \vec{z}_1, v)$ to the configuration $(\mathscr{C}, \emptyset, \mathscr{O}_{\preceq^2}, \vec{z}_2, v)$ using the *order change* rule. As explained above, it is clear that this rule preserves (F, f)-validity.

This concludes the presentation of our proof system. Each rule has been shown to preserve (F, f)-validity, and the initial configuration is clearly (F, f)-valid. Therefore, by Theorem 7 our proof system is sound: whenever we can derive a configuration $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ such that $\mathscr{C} \cup \mathscr{D}$ contains $0 \geq 1$, it holds that v is the value of f in any f-minimal solution of F (or, for a decision problem, we have $v < \infty$ precisely when F is satisfiable). As mentioned above, in this case the full sequence of configurations $(\mathscr{C}, \mathscr{D}, \mathscr{O}_{\preceq}, \vec{z}, v)$ together with annotations about the derivation steps—including, in particular, any witnesses ω—contains all information needed to efficiently reconstruct such an f-minimal solution of F. It is also straightforward to show that our proof system is complete: after using the bound update rule to log an optimal solution v^*, it follows from the implicational completeness of cutting planes that contradiction can be derived from $F \cup \{f \leq v^* - 1\}$.

C Technical appendix on Symmetry Breaking in SAT Solvers

In the "Symmetry Breaking in SAT Solvers" subsection, we discuss the core ideas that underlie most modern symmetry breaking tools for SAT. Devriendt et al. (2016) extend these ideas further in a couple of ways. In this technical appendix, we briefly discuss these techniques and how and why they fit in our proof system.

The most important contribution of Devriendt et al. (2016) is detecting so-called *row interchangeability*. The goal of this optimization is to not just take an arbitrary set of generators of the symmetry group and an arbitrary lexicographic order, but to choose "the right" set of generators and "the right" variable order (with which to define the lexicographic order). Devriendt et al. (2016) showed that for groups that exhibit a certain structure, breaking symmetries of a good set of *generators*, with a matching order, can guarantee that the entire symmetry *group* is broken completely. Since our logging techniques simply use the same lexicographic order as the breaking tool, and work for an arbitrary generator set, this automatically works with the techniques described in Section 4.

Another (optional) modification *BreakID* implements is writing out a more *compact encoding*. The authors observed that the definitions the y-variables can be weakened: the clauses (12c) and (12d) in the "Symmetry Breaking in SAT Solvers" subsection can be omitted. Since our definition of $C_{LL}(k)$ uses these clauses, we cannot simply omit them in our proof. However, all the symmetry breaking constraints are added in the set $\mathscr{D}$, and so we can remove these constraints from $\mathscr{D}$ as soon as they are no longer needed for the proof logging derivations.

Next, *BreakID* has an optimization based on *stabilizer subgroups* to detect a plethora of binary clauses. Since these binary clauses are all clauses of the form (12b) with $j = 1$, the described proof logging techniques also work for this optimization, provided we keep track of which symmetry is used for each such binary clause. However, *BreakID* currently does no such bookkeeping. While it is in principle possible to do so, we did not implement this yet.

Finally, *BreakID* supports *partial symmetry breaking*. That is, instead of adding the constraints (12b)–(12f) for *every* j, this is only done for $j < L$ with L a limit that can be chosen by the user. The reasoning behind this is that the larger j, gets, the weaker the added breaking constraint is. By only doing this, for instance for $j < 100$, the size of the added constraints can get significantly smaller without losing too much breaking power. Since we only need to do proof logging for the clauses that are actually added by *BreakID*, this optimization works out-of-the-box. However, there is an important caveat here: in benchmarks where there are huge symmetries, e.g., symmetries permuting all the variables in the problem, even when this optimisation is used, a naive implementation of our proof logging technique suffers from serious performance problems. The reason is that in principle the order $\mathscr{O}_{\preceq}$ is defined on all variables that are permuted by the symmetries. If there are many such variables, this order in itself can get huge (the largest coefficient is exponentially large in terms of the number of variables). Luckily, there is a simple solution to this problem, namely not taking $\vec{x}$ to be the set of all variables that are permuted, but only the set of variables on which we will actually do breaking (for each symmetry, the first L variables in its support); this solution was implemented in the experiments we presented.

A Complete Example of Proof Logging Symmetry Breaking

We now present a complete example of proof logging for symmetry for the well-known pigeon-hole problem. We consider an instance of this problem with 4 pigeons and 3 holes. We use variables p_{ij} to represent that pigeon i resides in hole j. The input for the symmetry breaking preprocessor consists of the constraints

$$p_{11} + p_{12} + p_{13} \geq 1 \tag{C1}$$
$$p_{21} + p_{22} + p_{23} \geq 1 \tag{C2}$$
$$p_{31} + p_{32} + p_{33} \geq 1 \tag{C3}$$
$$p_{41} + p_{42} + p_{43} \geq 1 \tag{C4}$$
$$\overline{p_{11}} + \overline{p_{21}} \geq 1 \tag{C5}$$
$$\overline{p_{11}} + \overline{p_{31}} \geq 1 \tag{C6}$$
$$\overline{p_{11}} + \overline{p_{41}} \geq 1 \tag{C7}$$
$$\overline{p_{21}} + \overline{p_{31}} \geq 1 \tag{C8}$$
$$\overline{p_{21}} + \overline{p_{41}} \geq 1 \tag{C9}$$
$$\overline{p_{31}} + \overline{p_{41}} \geq 1 \tag{C10}$$
$$\overline{p_{12}} + \overline{p_{22}} \geq 1 \tag{C11}$$
$$\overline{p_{12}} + \overline{p_{32}} \geq 1 \tag{C12}$$
$$\overline{p_{12}} + \overline{p_{42}} \geq 1 \tag{C13}$$
$$\overline{p_{22}} + \overline{p_{32}} \geq 1 \tag{C14}$$
$$\overline{p_{22}} + \overline{p_{42}} \geq 1 \tag{C15}$$
$$\overline{p_{32}} + \overline{p_{42}} \geq 1 \tag{C16}$$
$$\overline{p_{13}} + \overline{p_{23}} \geq 1 \tag{C17}$$
$$\overline{p_{13}} + \overline{p_{33}} \geq 1 \tag{C18}$$
$$\overline{p_{13}} + \overline{p_{43}} \geq 1 \tag{C19}$$
$$\overline{p_{23}} + \overline{p_{33}} \geq 1 \tag{C20}$$
$$\overline{p_{23}} + \overline{p_{43}} \geq 1 \tag{C21}$$
$$\overline{p_{33}} + \overline{p_{43}} \geq 1 \tag{C22}$$

where the first four constraints represent that each pigeon resides in at least one hole, and the rest that each hole is occupied by at most one pigeon.

Introducing the order A *VeriPB* proof starts with a proof header (stating which version of the proof system is used) and an instruction to load the input formula

```
pseudo-Boolean proof version 1.2
f 22
```

where the 22 is the number of formulas in the input (to ensure consistent constraint numbering). To do symmetry breaking, the proof *BreakID* yields, then contains the definition of the pre-order

```
pre_order exp22
  vars
    left u1 u2 u3 u4 u5 u6 u7 u8 u9 u10 u11 u12
    right v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12
    aux
  end

  def
    -1 u12 1 v12 -2 u11 2 v11 -4 u10 4 v10 -8 u9 8 v9 -16 u8 16 v8 -32 u7 32 v7 -64 u6 64
       ↪ v6 -128 u5 128 v5 -256 u4 256 v4 -512 u3 512 v3 -1024 u2 1024 v2 -2048 u1 2048
       ↪ v1 >= 0;
  end

  transitivity
    vars
      fresh_right w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w11 w12
    end
    proof
```

```
L19         proofgoal #1
L20           p 1 2 + 3 +
L21           c -1
L22         qed
L23       qed
L24     end
L25   end
```

The pre-order is given a name (`exp22`) in Line 3. Lines 5 and 6 introduce two times twelve auxiliary variables to define the order over. Line 11 then provides the well-known exponential encoding of the fact that the `u`-variables are lexicographically smaller than the `v`-variables. To prove transitivity, another set of variables (called `w`) is introduced. Formally, we need to show that equation (10b) holds. To prove this, we assume $\mathcal{O}_{\preceq}(\vec{u},\vec{v})$, $\mathcal{O}_{\preceq}(\vec{v},\vec{w})$ and $\neg\mathcal{O}_{\preceq}(\vec{u},\vec{w})$ hold. When instantiated with the specified order, these three constraints are

$$-u_{12} + v_{12} - 2u_{11} + 2v_{11} - 4u_{10} + 4v_{10} - \cdots \geq 0 \tag{T1}$$

$$-v_{12} + w_{12} - 2v_{11} + 2w_{11} - 4v_{10} + 4w_{10} - \cdots \geq 0 \tag{T2}$$

$$u_{12} - w_{12} + 2u_{11} - 2w_{11} + 4u_{10} - 4w_{10} + \cdots \geq -1, \tag{T3}$$

where we use the T-numbering to emphasize that these are not constraints learned in the proof system, but temporary constraints, local to the proof of transitivity. Line 20 is an instruction to add constraints (T1–T3), resulting (after simplification) in the constraint

$$0 \geq -1, \tag{T4}$$

Line 21 then states that the last derived constraint (the -1 stands for the last derived constraint) is a conflicting constraint, thereby concluding the proof of transitivity. Note that no proof for reflexivity is given since for simple orders such as the one specified here, veriPB's autoproving can construct a proof itself.

The proof continues with the instruction

```
L26   load_order exp22 p21 p22 p23 p11 p12 p13 p31 p32 p33 p41 p42 p43
```

stating that the order should be instantiated with the variables from the input. Do note that in the the chosen instantiation, (all variables related to) pigeon 2 are ordered before pigeon 1, then pigeons 3 and 4. In other words, in the lex-leader order, pigeon 2 has the highest importance.

Logging the breaking of a first symmetry The next step is to log constraints for breaking symmetries. The first symmetry considered is the symmetry

$$\pi := (p_{11}p_{43})(p_{12}p_{42})(p_{13}p_{41})(p_{21}p_{23})(p_{31}p_{33}),$$

which is the symmetry that swaps pigeons 1 and 4, and simultaneously swaps holes 1 and 3. In this work, we just take the set of symmetries to break on for granted and we will not elaborate on the possible reasons why this peculiar symmetry was chosen. As explained in our section on symmetry breaking, in order to break this symmetry, first an exponential encoding of a lex-leader constraint is added using the dominance rule, as follows

```
L27   dom -1 p43 1 p11 -2 p42 2 p12 -4 p41 4 p13 -8 p33 8 p31 -32 p31 32 p33 -64 p13 64 p41
        ↪ -128 p12 128 p42 -256 p11 256 p43 -512 p23 512 p21 -2048 p21 2048 p23  >= 0 ;  p11
        ↪ -> p43 p12 -> p42 p13 -> p41 p21 -> p23 p23 -> p21 p31 -> p33 p33 -> p31 p41 -> p13
        ↪  p42 -> p12 p43 -> p11 ;  begin
L28     proofgoal #2
L29       p -1 -2 +
L30       c -1
L31     qed
```

These instruction tell *VeriPB* to use the dominance rule to derive (and add to $\mathcal{D}$) the constraint in Line 27[4], which expresses that the assignment in question is lexicographically smaller than it's symmetric counterpart. As expected, the variables related to pigeon 2 occur with the highest coefficients (since when instantiating the order, they were given the highest priority).

The actual instruction for *VeriPB* does not just contain the constraint to be derived by dominance, but also specifies

- The witness, which in this case is just the symmetry, in Line 27, and
- A subproof of one of the proof obligations in Lines 28–31.

As far as the subproof is concerned: to apply the dominance rule, we need to show that the two implications

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash \mathcal{C}\!\upharpoonright_{\omega} \cup\, \mathcal{O}_{\preceq}(\vec{z}\!\upharpoonright_{\omega}, \vec{z}) \cup \{f\!\upharpoonright_{\omega} \leq f\} \tag{27a}$$

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \cup \mathcal{O}_{\preceq}(\vec{z}, \vec{z}\!\upharpoonright_{\omega}) \vdash \bot \tag{27b}$$

hold. To do so, *VeriPB* generates the following proof obligations:

1. $\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash \mathcal{O}_{\preceq}(\vec{z}\!\upharpoonright_{\omega}, \vec{z})$

[4]Notice that this constraint contains some duplicate variables, because of being generated automatically; later on this will be simplified.

2. $\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \neg \mathscr{O}_{\preceq}(\vec{z}, \vec{z}\!\upharpoonright_{\omega})$
3. $\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \{f\!\upharpoonright_{\omega} \leq f\}$
4–25 $\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash B$ for each $B \in \mathscr{C}\!\upharpoonright_{\omega}$.

Except for the second proof obligation, all of them can be proved automatically, for instance since we are working in the context of a decision problem where $f = 0$, the third one is trivial. Since ω is a syntactic symmetry of $\mathscr{C}$ (which is at this point still equal to the input), also the last ones are trivial. The proof of this second proof obligation goes as follows. First, *VeriPB* (automatically) adds the constraint $\neg C$, which (after simplification) equals:

$$255p_{11} + 126p_{12} + 60p_{13} + 1536p_{21} + 1536\overline{p_{23}} + 24p_{31} + 24\overline{p_{33}} + 60\overline{p_{41}} + 126\overline{p_{42}} + 255\overline{p_{43}} \geq 2002 \tag{C23}$$

Next, *VeriPB* (again, automatically) adds the constraint $\mathscr{O}_{\preceq}(\vec{z}, \vec{z}\!\upharpoonright_{\omega})$, which, after simplification, equals

$$255\overline{p_{11}} + 126\overline{p_{12}} + 60\overline{p_{13}} + 1536\overline{p_{21}} + 1536p_{23} + 24\overline{p_{31}} + 24p_{33} + 60p_{41} + 126p_{42} + 255p_{43} \geq 2001 \tag{C24}$$

Now the instruction 29 simply states that the last added constraint (i.e., Equation (C24)) and the one added before Equation (C23)) should be added resulting in

$$255 + 126 + 60 + 1536 + 1536 + 24 + 24 + 60 + 126 + 255 \geq 2002 + 2001, \tag{C25}$$

which is a contradiction. The line 30 states that the this is indeed a contradiction and the subproof for this proof obligation is ended. Finally, when this proof is finished and all other proof obligation have been automatically checked, the new constraint

$$\begin{aligned} -p_{43} + 1p_{11} - 2p_{42} + 2p_{12} - 4p_{41} + 4p_{13} - 8p_{33} + 8p_{31} - 32p_{31} + 32p_{33} - 64p_{13} + 64p_{41} \\ -128p_{12} + 128p_{42} - 256p_{11} + 256p_{43} - 512p_{23} + 512p_{21} - 2048p_{21} + 2048p_{23} \geq 0 \end{aligned} \tag{C26}$$

is added to $\mathscr{D}$. Afterwards, constraints (C23), (C24), and (C25) are removed, since they are constraints that were only valid for the subproof.

This constraint, by itself, is a lex-leader constraint for the symmetry at hand. However, since we are in the context of SAT solving, it still has to be translated to a set of clauses, which is what happens next. First, (12a) is added with the redundance rule with the instruction

```
L32  red 1 y0  >= 1 ; y0 -> 1
```

which contains both the constraint

$$y_0 \geq 1 \tag{C27}$$

and the witness $y_0 \mapsto 1$ to apply the redundance rule. All proof obligations are checked automatically by *VeriPB*.

In our chosen lexicographic order, the most prominent variable is p_{21}. As such, the first clause for symmetry breaking is

$$\overline{p_{21}} \vee \pi(p_{21}) \doteq \overline{p_{21}} \vee p_{23},$$

which is a simplification of (12b), omitting the trivially true y_0. The constraint (C26) implies the above constraint (e.g., using weakening out all other variables in that constraint). Instead of giving the actual derivation, we can simply add it with the reverse unit propagation rule and let *VeriPB* figure out the details by

```
L33  u 1 ~p21 1 p23 >= 1 ;
```

resulting in

$$\overline{p_{21}} + p_{23} \geq 1. \tag{C28}$$

Next, the Tseitin variable y_1 is introduced with four redundance rule applications

```
L34  red 1 p23 1 ~y0 1 y1  >= 1 ; y1 -> 1
L35  red 1 ~p21 1 ~y0 1 y1  >= 1 ; y1 -> 1
L36  red 1 ~y1 1 y0  >= 1 ; y1 -> 0
L37  red 1 ~y1 1 ~p23 1 p21  >= 1 ; y1 -> 0
```

each of them also mentioning the witness mapping y_1 either to 0 or to 1, resulting in the constraints

$$p_{23} + \overline{y_0} + y_1 \geq 1 \tag{C29}$$

$$\overline{p_{21}} + \overline{y_0} + y_1 \geq 1 \tag{C30}$$

$$\overline{y_1} + y_0 \geq 1, \text{and} \tag{C31}$$

$$\overline{y_1} + \overline{p_{23}} + p_{21} \geq 1 \tag{C32}$$

corresponding to the constraints (12c)–(12f).

Before repeating this procedure for the next variable, we use the recently derived constrain to cancel out the dominant terms in constraint (C26) with the instructions

```
L38  p 26 32 2048 * +
L39  d 26
```

The first of these instructions results in adding (C32) 2048 times to (C26), resulting in

$$255\overline{p_{11}} + 126\overline{p_{12}} + 60\overline{p_{13}} + 512p_{21} + 512\overline{p_{23}} + 24\overline{p_{31}} + 24p_{33} + 60p_{41} + 126p_{42} + 255p_{43} + 2048\overline{y_1} \geq 977. \tag{C33}$$

The last of these instructions deletes (C26) from $\mathscr{D}$ since it will no longer be required.

After p_{21}, the next most important variable is p_{22}. However, since our symmetry π at hand maps p_{22} to itself, no symmetry breaking clauses are added for it. The next variable in the ordering is p_{23}, which is mapped to p_{21}, resulting in the (conditional on y_1) symmetry breaking constraint

$$\overline{y_1} + \overline{p_{23}} + p_{21} \geq 1 \tag{C34}$$

obtained by the instruction

```
u 1 ~y1 1 ~p23 1 p21 >= 1 ;
```

Next, as before, the next Tseitin variable y_2 is introduced with the redundance rule using

```
red 1 p21 1 ~y1 1 y2  >= 1 ; y2 -> 1
red 1 ~p23 1 ~y1 1 y2  >= 1 ; y2 -> 1
red 1 ~y2 1 y1  >= 1 ; y2 -> 0
red 1 ~y2 1 ~p21 1 p23  >= 1 ; y2 -> 0
```

resulting in the constraints

$$p_{21} + \overline{y_1} + y_2 \geq 1 \tag{C35}$$

$$\overline{p_{23}} + \overline{y_1} + y_2 \geq 1 \tag{C36}$$

$$y_1 + \overline{y_2} \geq 1, \text{ and} \tag{C37}$$

$$\overline{p_{21}} + p_{23} + \overline{y_2} \geq 1 \tag{C38}$$

As before, our pseudo-Boolean symmetry breaking constraint is simplified with

```
p 33 38 512 * +
d 33
```

where the first instruction again cancels out the dominant terms (replacing them by y-variables) to express a conditional symmetry breaking constraint, resulting in

$$255\overline{p_{11}} + 126\overline{p_{12}} + 60\overline{p_{13}} + 24\overline{p_{31}} + 24p_{33} + 60p_{41} + 126p_{42} + 255p_{43} + 2048\overline{y_1} + 512\overline{y_2} \geq 465 \tag{C39}$$

The next most important variable in our chosen order is p_{11}, which is mapped to p_{43}, resulting in the symmetry breaking constraint

$$\overline{p_{11}} + p_{43} + \overline{y_2} \geq 1 \tag{C40}$$

added by

```
u 1 ~y2 1 ~p11 1 p43 >= 1 ;
```

To see that this constraint indeed follows by reverse unit propagation, we observe that whenever y_2 holds, so do y_1 and y_0 (by (C37) and (C31). If furthermore $\overline{p_{43}}$ and p_{11} hold, (C39) simplifies to

$$126\overline{p_{12}} + 60\overline{p_{13}} + 24\overline{p_{31}} + 24p_{33} + 60p_{41} + 126p_{42} \geq 465,$$

which can never be satisifed since the coefficients on the left add up to 420.

The process of introducing a new Tseitin variable is the same as before, resulting in the addition of the following constraints to $\mathscr{D}$:

$$p_{43} + \overline{y_2} + y_3 \geq 1 \tag{C41}$$

$$\overline{p_{11}} + \overline{y_2} + y_3 \geq 1 \tag{C42}$$

$$y_2 + \overline{y_3} \geq 1 \tag{C43}$$

$$p_{11} + \overline{p_{43}} + \overline{y_3} \geq 1 \tag{C44}$$

This last constraint can then again be used to simplify (C39) with

```
p 39 44 256 * +
```

resulting in

$$p_{11} + 126\overline{p_{12}} + 60\overline{p_{13}} + 24\overline{p_{31}} + 24p_{33} + 60p_{41} + 126p_{42} + \overline{p_{43}} + 2048\overline{y_1} + 512\overline{y_2} + 256\overline{y_3} \geq 211 \tag{C45}$$

This process continues by considering all variables not stabilized by π in the order used for lexicographic ordering.

Logging the breaking of more symmetries Afterwards, more symmetries are broken, and the resulting clauses logged. The process is completely the same as with the first symmetry. In our example, *BreakID* decided to break the following symmetries next:

$$(p_{11}p_{12})(p_{21}p_{32})(p_{22}p_{31})(p_{23}p_{33})(p_{41}p_{42})$$
$$(p_{21}p_{11})(p_{22}p_{12})(p_{23}p_{13})$$
$$(p_{11}p_{31})(p_{12}p_{32})(p_{13}p_{33})$$
$$(p_{31}p_{41})(p_{32}p_{42})(p_{33}p_{43})$$
$$(p_{21}p_{22})(p_{11}p_{12})(p_{31}p_{32})(p_{41}p_{42}),\ \text{and}$$
$$(p_{22}p_{23})(p_{12}p_{13})(p_{32}p_{33})(p_{42}p_{43}).$$

The first of these symmetries swaps holes 1 and 2 and simultaneously swaps pigeons 2 and 3. It is important to note here that the breaking of these six symmetries by no means interacts with the previously derived breaking clauses: the order remains unchanged and all previously added constraints were added to $\mathscr{D}$, hence $\mathscr{C}$ still consists only of the input formula.

D Technical Appendix on Proof Logging for CP Symmetry Breaking

In the "Symmetries in Constraint Programming" subsection we describe how we can use proof logging in a constraint programming setting to verify an algorithm for solving the Crystal Maze puzzle. In this appendix we describe the key ideas behind an implementation of this algorithm; source code to run the demo is located in the `tools/crystal-maze-solver` directory of the code and data repository[5], and full instructions are given in the `tools/crystal-maze-solver/README.md` file. An outline of this work follows below.

We modelled the Crystal Maze puzzle as a constraint satisfaction problem in the natural way: we have a decision variable for each circle, whose values are the possible numbers that can be taken, and an all-different constraint over all decision variables. We use a table constraint for each edge, for simplicity. We also included symmetry elimination constraints.

We implemented this model inside a small proof-of-concept CP solver `src/crystal_maze.cc` that we created for this paper. (Full proof logging for CP is an entire research program in its own right, which we do not claim to have carried out—what we do claim, though, is that our contribution shows that symmetries do not stand in the way of this work.) When executed, the solver compiles this high level CP model to a pseudo-Boolean model, which it will output as `crystal_maze.opb`. This is done following the framework introduced by Elffers et al. (2020), but as well as using a one-hot (direct) encoding of CP decision variables to PB variables, it additionally creates channelled greater-or-equal PB variables for each CP variable-value. Note that the encoding of the table constraints also introduces additional auxiliary variables.

Then, as it solves the problem, the solver outputs `crystal_maze.veripb`, which provides a proof that it has found all non-symmetric solutions. (Note that our solver maintains generalised arc consistency on the all-different and table constraints, and so is performing propagation that requires explicit justification in the proof log.) These two outputs can be verified using *VeriPB*.

To verify that the symmetry constraints introduced in the high level model are actually valid, we can remove them from the pseudo-Boolean model and introduce them as part of the proof instead. We describe how to do this editing in `README.md`. We also include a script `make-symmetries.py` that will output the necessary proof fragment to reintroduce the symmetry constraints. The output of this script can be verified on top of the reduced pseudo-Boolean model using our modified version of *VeriPB*, with or without the remainder of the proof—that is, we can both verify that the constraints introduced are valid (in that they do not alter the satisfiability of the model), and that they line up with the actual execution of the solver.

[5]See https://doi.org/10.5281/zenodo.6373986.

E Technical Appendix on Proof Logging for Vertex Dominance in Max-Clique Solving

Throughout this last appendix we let $G = (V, E)$ denote an undirected, unweighted graph without self-loops with vertices V and edges E. We write $N(u)$ to denote the *neighbours* of a vertex $u \in V$, i.e., the set of vertices $N(u) = \{w \mid (u, w) \in E\}$ that are adjacent to u in the graph, and define neighbours of sets of vertices in the natural way by taking unions $N(U) = \bigcup_{u \in U} N(u)$.

We say that u *dominates* v if

$$N(u) \setminus \{v\} \supseteq N(v) \setminus \{u\} \tag{28}$$

holds, which intuitively says that the neighbourhood of u is at least as large as that of v. It is straightforward to verify that this domination relation is transitive.

When representing the maximum clique problem in pseudo-Boolean form, we overload notation and identify every vertex $v \in V$ with a Boolean variable, where $v = 1$ means that the vertex v is in the clique. The task is to maximize $\sum_{v \in V} v$ under the constraints that all chosen vertices should be neighbours, but since, syntactically speaking, we require an objective function to be minimized, we obtain

$$\min \textstyle\sum_{v \in V} \overline{v} \tag{29a}$$

$$\overline{v} + \overline{w} \geq 1 \qquad \text{[for all } (v, w) \notin E\text{]} \tag{29b}$$

as the formal pseudo-Boolean specification of the problem.

High-Level Description of the Max Clique Solver

At a high level, the maximum clique solver of McCreesh and Prosser (2016), but before addition of vertex dominance breaking, works as described in Algorithm 1. The first call to the $\mathsf{MaxCliqueSearch}$ algorithm is with parameters G, $V_{\mathrm{rem}} = V$, $C_{\mathrm{curr}} = \emptyset$, and $C_{\mathrm{best}} = \emptyset$.

When $\mathsf{MaxCliqueSearch}$ is called with a candidate clique C_{curr}, the best solution so far C_{best}, and a subset of vertices V_{rem}, it considers the residual graph $G_{\mathrm{rem}} = (V_{\mathrm{rem}}, E_{\mathrm{rem}})$ assumed to be defined on all vertices in $V \setminus C_{\mathrm{curr}}$ that are neighbours of all $c \in C_{\mathrm{curr}}$. Thus, the set V_{rem} contains all vertices to which C_{curr} could possibly be extended. The algorithm produces a colouring of G_{rem}, which we assume results in m disjoint colour classes $(S_1, \ldots, S_m)$ such that $V_{\mathrm{rem}} = \bigcup_{i=1}^{m} S_i$. It is clear that any clique extending C_{curr} can contain at most one vertex from every colour class S_i. The clique search algorithm now iterates over all colour classes in the order $S_m, S_{m-1}, \ldots, S_1$. Whenever the clique is extended with a new vertex, a new recursive call to $\mathsf{MaxCliqueSearch}$ is made. Therefore, when we reach S_j in the loop, we are considering the case when all vertices in $S_m, S_{m-1}, \ldots, S_{j+1}$ have been rejected. For this reason, if the condition $|C_{\mathrm{curr}}| + j > |C_{\mathrm{best}}|$ fails to hold, we know that the current clique candidate cannot possibly be extended to a clique that is larger than what we have already found in C_{best}. At the end of the first call $\mathsf{MaxCliqueSearch}(G, V, C_{\mathrm{curr}} = \emptyset, C_{\mathrm{best}} = \emptyset)$, after completion of all recursive subcalls, the vertex set C_{best} will be a clique of maximum size in G. A certifying version of essentially this algorithm with *VeriPB* proof logging was

Algorithm 1: Max clique algorithm without dominance.

1 $\mathsf{MaxCliqueSearch}(G, V_{\mathrm{rem}}, C_{\mathrm{curr}}, C_{\mathrm{best}})$:
2 $E_{\mathrm{rem}} \leftarrow E(G) \cap (V_{\mathrm{rem}} \times V_{\mathrm{rem}})$;
3 $G_{\mathrm{rem}} \leftarrow (V_{\mathrm{rem}}, E_{\mathrm{rem}})$;
4 **if** $|C_{\mathrm{curr}}| > |C_{\mathrm{best}}|$ **then**
5 $\quad C_{\mathrm{best}} \leftarrow C_{\mathrm{curr}}$;
6 $(S_1, \ldots, S_m) \leftarrow$ colour classes in colouring of G_{rem} ;
7 $j \leftarrow m$;
8 **while** $j \geq 1$ **and** $|C_{\mathrm{curr}}| + j > |C_{\mathrm{best}}|$ **do**
9 $\quad$ **for** $v \in S_j$ **do**
10 $\quad\quad C_{\mathrm{best}} \leftarrow \mathsf{MaxCliqueSearch}(G, V_{\mathrm{rem}} \cap N(v), C_{\mathrm{curr}} \cup \{v\}, C_{\mathrm{best}})$;
11 $\quad V_{\mathrm{rem}} \leftarrow V_{\mathrm{rem}} \setminus S_j$;
12 $\quad j \leftarrow j - 1$;
13 **return** C_{best};

presented by Gocht et al. (2020). It might be worth noting in this context that one quite interesting challenge is to justify the backtracking performed when the condition $|C_{\mathrm{curr}}| + j > |C_{\mathrm{best}}|$ fails, and this is one place where the strength of the pseudo-Boolean reasoning in the cutting planes proof system is very helpful (as opposed to the clausal reasoning in, e.g., *DRAT*).

The vertex dominance breaking of McCreesh and Prosser (2016) is based on the following observation: If the algorithm is about to consider $v \in S_j$ in the innermost for loop on line 9 in Algorithm 1, but has previously considered a vertex $u \in \bigcup_{i=j}^{m} S_i$ that dominates v in the sense of (28), then it is safe to ignore v. If the algorithm would find a solution that includes v but not u, then we can swap u for v and obtain a solution that is at least as good.

In pseudo-Boolean notation, this would correspond to adding the constraint $u + \overline{v} \geq 1$ to the formula, but there is no way this can be semantically derived from the constraints (29a) or the requirement to minimize (29b). Therefore, the proof logging method in (Gocht et al. 2020) is inherently unable to deal with such constraints.

In general, the vertex dominance breaking as described above does not need to break ties consistently. By this we mean that if u and v dominate each other, in principle it might happen that in a given branch of the search tree, u is chosen to dominate v, while in another one, v is chosen to dominate u, simply because of the order in which nodes are considered. While in principle, our proof logging methods can be adapted to work in this case, the argument is subtle. Luckily, it turns out that in practical implementations, tie breaking only happens in a consistent manner.

Fact 13. *In the vertex dominance breaking of McCreesh and Prosser (2016), there exists a total order $\succ_G$ on the set V of vertices such that whenever v is ignored because u has previously been considered, $u \succ_G v$.*

Moreover, this order $\succ_G$ is known before the algorithm starts: $u \succ_G v$ holds if u has a larger degree than v, or in case they have the same degree but the identifier used to represent u internally is larger than that of v. To see that this order indeed guarantees consistent tie breaking, we provide some properties of the actual implementation of the algorithm.

1. If u and v dominate each other and are not adjacent, u and v are guaranteed to be in the same coloring class. If furthermore $u \succ_G v$, u is considered before v in the loop in Line 8 (due to the order in which this for loop iterates over the nodes).
2. If u and v dominate each other, *are* adjacent, and $u \succ_G v$, then u is assigned a *larger* coloring class than v (due to the order in which the (greedy) coloring algorithm in Line 6 iterates over the nodes). Hence, also in this case u will be considered before v.

In what follows below, we will explain

- first, how the redundance rule introduced to *VeriPB* by Gocht and Nordström (2021) could in principle be used to provide proof logging for vertex dominance breaking, but that this seems hard to get to work in practice; and
- then, how the dominance rule introduced in this paper can be used to resolve the practical problems in a very simple way.

An implementation for both techniques can be found in the code and data repository.[5]

Vertex Dominance with the Redundance Rule

In order to provide proof logging for vertex dominance breaking using the redundance rule, we could in theory proceed as follows. First, we let the solver check the vertex dominance condition (28) for all pairs of vertices u, v in V.

Before starting the solver, we add all pseudo-Boolean constraints for vertex dominance breaking using the redundance rule. For all u, v such that u dominates v and $u \succ_G v$, we derive the *vertex dominance breaking constraint*

$$u + \overline{v} \geq 1 \ , \tag{30}$$

doing so *in decreasing order* for u with respect to $\succ_G$. Our witness for the redundance rule derivation of (30) will be $\omega = \{u \mapsto v, v \mapsto u\}$, i.e., ω will simply swap the dominating and dominated vertices. Hence, the objective function (29a) is syntactically unchanged after substitution by ω, and so the condition in (5) that the objective should not increase is always vacuously satisfied.

We need to argue that deriving the vertex dominance breaking constraints (30) is valid in our proof system. Towards this end, suppose we are in the middle of the process of adding such constraints and are currently considering $u+\overline{v} \geq 1$ for u dominating v and $u \succ_G v$. Let $\mathscr{C} \cup \mathscr{D}$ be the set of constraints in the current configuration. In order to add $u + \overline{v} \geq 1$, we need to show that

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg(u + \overline{v} \geq 1)\} \tag{31a}$$

can be used to derive all constraints in

$$(\mathscr{C} \cup \mathscr{D} \cup \{u + \overline{v} \geq 1\})\restriction_{\omega} \tag{31b}$$

by the cutting planes method (i.e., without any extension rules).

Starting with the vertex dominance constraint being added, note that from the negated constraint $\neg(u+\overline{v} \geq 1) \doteq \overline{u} + v \geq 2$ in (31a) we immediately obtain

$$\overline{u} \geq 1 \tag{32a}$$

$$v \geq 1 \tag{32b}$$

as RUP constraints, meaning that the weaker constraint $(u + \overline{v} \geq 1)\restriction_{\omega} \doteq v + \overline{u} \geq 1$ is also RUP with respect to the constraints in (31a).

Consider next any non-edge constraints $\overline{x}+\overline{y} \geq 1$ in (29b) in the original formula. Clearly, such constraints are only affected by ω if $\{u, v\}\cap\{x, y\} \neq \emptyset$; otherwise they are present in both (31a) and (31b) and there is nothing to prove. Any non-edge constraint $\overline{v} + \overline{y} \geq 1$ containing $\overline{v}$ will after application of ω contain $\overline{u}$, and will hence be RUP with respect to (32a) and hence also with respect to (31a). For non-edge constraints $\overline{u} + \overline{y} \geq 1$ with $y \neq v$, substitution by ω yields $(\overline{u} + \overline{y} \geq 1)\restriction_{\omega} \doteq \overline{v} + \overline{y} \geq 1$. Since by assumption u dominates v and $y \neq v$ is not a neighbour of u, it follows from (28) that y is not a neighbour of v either. Hence, the input formula in (31a) already contains the desired non-edge constraint $\overline{v} + \overline{y} \geq 1$.

It remains to analyse what happens to vertex dominance breaking constraints

$$x + \overline{y} \geq 1 \tag{33}$$

that have already been added to $\mathscr{D}$ before the dominance breaking constraint $u + \overline{v} \geq 1$ that we are considering now. Again, such a constraint is only affected by ω if $\{u, v\} \cap \{x, y\} \neq \emptyset$; otherwise it is present in both (31a) and (31b). We obtain the following case analysis.

1. $x = u$: In this case, $(x + \overline{y} \geq 1)\restriction_{\omega} \doteq v + \overline{\omega(y)} \geq 1$, which is RUP with respect to $v \geq 1$ in (32b) and hence also with respect to (31a).
2. $x = v$: This is impossible, since $u \succ_G v$ and any dominance breaking constraints with $v = x$ as the dominating vertex will be added only once we are done with u as per the description right below (30).
3. $y = u$: In this case x dominates u. Since $x \succ_G u$, $u \succ_G v$, and u dominates v, by transitivity we have $x \succ_G v$ and also that x dominates v. Hence, the breaking constraint $x + \overline{v} \geq 1$ has already been added to $\mathscr{D}$. But since $u \neq x \neq v$, we see that our desired constraint is $(x + \overline{u} \geq 1)\restriction_{\omega} \doteq x + \overline{v} \geq 1$, which is precisely this previously added constraint.
4. $y = v$: Here we see that the desired constraint $(x + \overline{y} \geq 1)\restriction_{\omega} \doteq \omega(x) + \overline{u} \geq 1$ is again RUP with respect to (31a).

This concludes our proof that all vertex dominance breaking constraints that are consistent with our constructed linear order $\succ_G$ can be added and certified by the redundance rule before the solvers starts searching for cliques.

So all of this works perfectly fine in theory. The problem that rules out this approach in practice, however, is that the solver will not have the time to compute the dominance relation between vertices in advance, since this is far too costly and does not pay off in general. Instead the solver designed by McCreesh and Prosser (2016) will detect and apply vertex dominance relations on the fly during search. And from a proof logging perspective this is too late—during search, when $\mathscr{C} \cup \mathscr{D}$ will also contain constraints justifying any backtracking made, our proof logging approach above no longer works. The constraints added to the proof log to justify backtracking are no longer possible to derive when substituted by ω as in (31b), for the simple reason that they are not semantically implied by (31a). One possible way around this would be to run the solver twice—the first time to collect all information about what vertex dominance breaking will be applied, and then the second time to do the actual proof logging—but this seems like quite a cumbersome approach. Moreover, even when doing so, our argument of correctness uses that the set of pairs (u, v) for which a constraint $u+\overline{v} \geq 1$ is derived is transitively closed, i.e., we would still add more constraints than what the solver actually needs.

We deliberately discuss this problem in some detail here, because this is an example of an important and nontrivial challenge that shows up also in other settings when designing proof logging for other algorithms. It is not sufficient to just come up with a proof logging system that is strong enough in principle to certify the solver reasoning (which the redundance rule is for the clique solver with vertex dominance breaking, as shown above). It is also crucial that the solver have enough information available at the right time and can extract this information efficiently enough to actually be able to emit the required proof logging commands with low enough overhead. For constraint programming solvers, it is not seldom the case that the solver knows for sure that some variable should propagate to a value, because the domain has shrunk to a singleton, or that the search should backtrack because some variable domain is empty, but that the solver cannot reconstruct the detailed derivation steps required to certify this without incurring a massive overhead in running time (e.g., since the reasoning has been performed with bit-parallel logical operations). It is precisely for this reason that it is important that our proof system allow adding reverse unit propagation (RUP) constraints. This makes it possible for the solver to claim facts that it knows to be true, and that it knows can be easily verified, while leaving the work of actually producing a detailed justification to the proof checker.

Vertex Dominance with the Dominance Rule

Similar to the case of the redundance rule we will make use of Fact 13. Before staring the proof logging, we use the order change rule to activate the lexicographic order on the the assignments to the vertices/variables induced by $\succ_G$.

Suppose now that the solver is running and that the current candidate clique is C_{curr}. The solver has an ordered list of unassigned candidate vertices that it is iterating over when considering how to enlarge this clique, and this list is defined by the colour classes $(S_m, S_{m-1}, \ldots, S_1)$. (We note that this ordered list depends on C_{curr}, and would be different for a different clique C'_{curr}.) Suppose the next vertex in that list is v. Then when it is time to make the next decision on line 9 in Algorithm 1 about enlarging the clique, we can apply the following decision algorithm:

1. If there exists a vertex u that has already been considered in the current iteration and that dominates v (and hence for which $u \succ_G v$), then discard v by vertex dominance and add the constraint $u + \overline{v} \geq 1$ by the dominance rule with witness $\omega = \{u \mapsto v, v \mapsto u\}$. We will below explain in detail why this is possible.
2. Otherwise, enlarge C_{curr} with v and make a recursive call.

When the solver has explored all ways of enlarging C_{curr} and is about to backtrack, here is what will happen on the proof logging side (where we refer to (Gocht et al. 2020) for a more detailed description of how proof logging for backtracking CP solvers works in general):

1. For every u that was explored in an enlarged clique $C_{\mathrm{curr}} \cup \{u\}$, when backtracking the solver will already have added $\overline{u} + \sum_{w \in C_{\mathrm{curr}}} \overline{w} \geq 1$ as a RUP constraint.
2. The solver now inserts the explicit cutting planes derivation required to justify that $|C_{\mathrm{curr}}| + j > |C_{\mathrm{best}}|$ must hold.
3. After this, the solver adds the claim that $\sum_{w \in C_{\mathrm{curr}}} \overline{w} \geq 1$ is a RUP constraint.

We need to argue why $\sum_{w \in C_{\mathrm{curr}}} \overline{w} \geq 1$ will be accepted as a RUP constraint, allowing the solver to backtrack. The RUP check for $\sum_{w \in C_{\mathrm{curr}}} \overline{w} \geq 1$ propagates $w = 1$ for all $w \in C_{\mathrm{curr}}$. This in turn propagates $u = 0$ for all explored vertices u by the backtracking constraints for $C_{\mathrm{curr}} \cup \{u\}$ added in step 1. The vertex dominance breaking constraints then propagate $v = 0$ for all vertices v discarded because of vertex domination. At this point, the proof checker has the same information that the solver had when it detected that the colouring constraint forced backtracking. This means that the proof checker will unit propagate to contradiction, and so the backtracking constraint $\sum_{w \in C_{\mathrm{curr}}} \overline{w} \geq 1$ is accepted as a RUP constraint.

We still need to explain how and why the pseudo-Boolean dominance rule applications allow deriving the constraint $u + \overline{v} \geq 1$ in case u dominates v (and hence $u \succ_G v$). Recall that the order used in our proof is the lexicographic order induced by $\succ_G$. This means that if vertices/variables u and v are assigned by α in such a way as to violate a dominance breaking constraint $u + \overline{v} \geq 1$, then $\alpha \circ \omega$ will flip u to 1 and v to 0 to produce a lexicographically smaller assignment (since v is considered before u in the lexicographic order).

The conditions for the dominance rule are that we have to exhibit proofs of (7a) and (7b). In this discussion, let us focus on (7a) which says that starting with the constraints

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg(u + \overline{v} \geq 1)\} \tag{34a}$$

and using only cutting planes rules, we should be able to derive

$$\mathscr{C}\!\upharpoonright_\omega \cup\, \mathscr{O}_\preceq(\vec{z}\!\upharpoonright_\omega, \vec{z}) \cup \{f\!\upharpoonright_\omega\ \leq f\} \ . \tag{34b}$$

Note first that our lexicographic order in fact does *not* in itself respect the objective function (29b). However, since ω just swaps two variables it leaves the objective syntactically unchanged, meaning that the inequality $f\!\upharpoonright_\omega\ \leq f$ in (7a) is seen to be trivially true.

As in our analysis of the redundance rule, from (34a) we obtain $\overline{u} \geq 1$ and $v \geq 1$ as in (32a)–(32b), and $\mathscr{O}_\preceq(\vec{z}\!\upharpoonright_\omega, \vec{z})$ is easily verified to be RUP with respect to these constraints, since what the formula says after cancellation is precisely that $v \geq u$.

It remains to consider the pseudo-Boolean constraints in the solver constraint database $\mathscr{C} \cup \mathscr{D}$. The crucial difference from the redundance rule is that we no longer have to worry about proving $\mathscr{D}\!\upharpoonright_\omega$ in (34b)—we only need to show how to derive $\mathscr{C}\!\upharpoonright_\omega$. But this means that all we need to consider are the non-edge constraints in (29b), and we already explained in our analysis for the redundance rule derivation that the fact that u dominates v means that for any non-edge constraints affected by ω their substituted versions are already there as input constraints or are easily seen to be RUP constraints. In addition to these non-edge constraints there might also be all kinds of interesting derived constraints in $\mathscr{D}$, but the dominance rule says that we can ignore those constraints.

Finally, although we skip the details here, it is not hard to argue analogously to what has been done above to show that $\neg C \doteq \neg(u + \overline{v} \geq 1)$ and $\mathscr{O}_\preceq(\vec{z}, \vec{z}\!\upharpoonright_\omega)$ in (7b) together unit propagate to contradiction. This concludes our discussion of how to certify vertex dominance breaking in the maximum clique solver by McCreesh and Prosser (2016) using the pseudo-Boolean dominance rule introduced in this paper.