On Symmetries and Transformations

Daimy Van Caudenberg
KU Leuven, Dept. Computer Science, Leuven, Belgium

Markus Anders
RPTU Kaiserslautern-Landau, Germany

Bart Bogaerts
KU Leuven, Dept. Computer Science, Leuven, Belgium
Vrije Universiteit Brussel, Brussels, Belgium

Abstract

When solving constraint problems, symmetry handling is a crucial optimization. However, it is well-known that several preprocessing methods and encodings change the syntactic symmetries of the problem, and hence make it more difficult to exploit the symmetries that were present in the original problem. As a consequence, when currently using such methods, one should either handle the symmetries on the original specification or live with the fact that efficiency is lost since this structure is no longer visible. In this paper, we take a different approach: we develop a framework in which symmetry information is explicitly part of the specification and can be passed along through different transformations. One subtle, but important point in this respect is that transformations can change the set of variables. We study theoretical properties of transformations that preserve symmetry information, and we analyze existing transformations from the literature in this framework. We experimentally evaluate our framework on translations of pseudo-Boolean constraints into CNF and show that in practice, simply passing on the symmetry information can lead to significant speed-ups in solving time.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Mathematics of computing → Permutations and combinations

Keywords and phrases Symmetries, Theory, Modelling & Modelling Languages

Digital Object Identifier 10.4230/LIPIcs.CP.2026.39

Supplementary Material *Software*: `Source code on GitHub`

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

1 Introduction

Exploiting symmetry is a crucial optimization in many areas of automated reasoning [15, 23, 31]. By detecting and exploiting symmetries in a problem instance, redundant work can often be avoided. In practice, solvers typically focus on *syntactic* symmetries, that is, symmetries that are visible in the structure of the instance. Such symmetries can often be modelled as graph automorphisms, and highly efficient tools for computing them are available [29, 8].

However, the problem description that reaches the solver is often not the original one. Before solving starts, the instance may go through several preprocessing or compilation steps. These transformations can haphazardly remove or obscure symmetries that were explicit in the high-level description of the problem. Hence, in practice, these symmetries can no longer be detected and used at the solver level.

This issue arises in many different contexts across constraint programming and other fields of automated reasoning. We discuss several examples here, though the list is by no means

exhaustive. A first place where this phenomenon occurs is translations between formalisms; for instance, many paradigms offer tools for translating search problems into SAT. This is done for instance for pseudo-Boolean (PB) solving [10] and constraint problems [35]. In such translations, important types of symmetry that are easily detected at a high abstraction level become difficult to detect, or even vanish completely, in the resulting encoding [19]. The reason is that even if the original problem has a simple symmetric structure, the encoding often introduces auxiliary variables in an asymmetric way, which can break symmetries. A second example is the decomposition of global constraints into networks of simpler constraints [1]. Again, when doing so, often arbitrary, asymmetric choices are made, which can break symmetries. Sorting-network decompositions compile constraints such as all-different or cardinality constraints into a fixed network of comparators, each operating on a specific pair of variables; a symmetry of the original constraint exchanging two variables will generally not correspond to an automorphism of these networks. Similarly, decompositions based on binary (or multi-valued) decision diagrams build a diagram by fixing a variable order; any symmetry permuting variables maps the chosen order to a structurally different one, making the symmetry unexploitable. A third and final example is the introduction of auxiliary variables for solver performance—not as a replacement encoding, but *alongside* the original constraints. This is done for instance in structure-based extended resolution in lazy clause generation [14]. There, new variables are introduced in order to extend the *language of learning* (the set of variables over which the solver can learn new clauses). E.g., if the input specification contains a sum constraint $\sum_{i=1}^{n} x_i \geq k$, then the solver will introduce auxiliary variables s_k representing $\sum_{i=1}^{k} x_i$, thereby breaking all symmetries between variables x_i.

Two common ways of dealing with the fact that transformations can destroy symmetries are (1) either breaking symmetries at the model level before compilation, or (2) detecting symmetries from the compiled output after the fact. Our framework offers a third path that neither approach makes available: by detecting symmetries at the model level, and making transformations symmetry-aware, symmetry information directly finds its way to the solver. As such, it leaves all decisions to the final solver without losing symmetries during the compilation step. The solver can then choose among different lex-leader orderings [25], apply dynamic symmetry-breaking techniques [30, 17], or exploit symmetries during search [22], none of which is possible when symmetry breaking is committed to before compilation.

Contributions. In this paper, we propose a new approach to preserve symmetry. Instead of requiring the user to provide symmetries manually to the final solver, or restricting ourselves to transformations that happen to preserve symmetries, we develop a framework in which symmetry is an explicit part of the specification. This information is then passed automatically through transformations that occur between modelling and solving.

A key challenge is that transformations may change the set of variables, for example by introducing auxiliary variables or by removing existing ones. To address this issue, we introduce the notion of a *symmetry-aware problem*, which consists of three components: a constraint problem, a designated set of *projected variables*, and a group of symmetries. The projected variables are the variables on which the symmetry group is allowed to act, while the remaining variables are not affected by the symmetry.

Within this framework, we study how symmetry information behaves under common transformations. We analyze several transformations from the literature, such as symmetry breaking, symmetry detection, PB-to-CNF encoding, and CNF simplification techniques. In particular, we study whether they can be made to preserve symmetry information and whether they can be used safely in settings such as counting and enumeration.

We experimentally evaluate our approach for translations of pseudo-Boolean constraints

to CNF. In particular, we pass symmetry information along the transformation using an auxiliary graph. On the resulting CNF encodings, we run the symmetry breaking tool SATSUMA [7]. We compare two settings: one where SATSUMA detects symmetries from the CNF itself, and one where it uses the symmetries provided by our auxiliary graph. We then solve the resulting instances with the state-of-the-art SAT solver KISSAT. On benchmark instances from the PB Competition 2025, we observe that the additional symmetry information reduces the average solving time by 12.6% compared to breaking symmetries on the CNF encoding.

Related Work. Previous work in SAT has studied which preprocessing techniques preserve symmetry and which ones destroy it [5]. Our framework goes beyond this work in that we show how to preserve symmetry information even when applying transformations that do not preserve it. One example of this is *bounded variable addition*, a crucial CNF simplification technique in modern SAT solvers [26], which we discuss in Section 4.3.1.

Our framework builds on ideas that have existed for several years and in fact, several existing methods fit well in our framework. One example is the CONJURE tool [3], which translates (high-level) Essence [21] specifications into lower level constraints. In doing so, by assigning names to objects, it introduces symmetries that were not present in the original specification and that need to be dealt with, which is not always easy to do completely [2]. In our framework, instead of doing the symmetry breaking, symmetry information is explicitly part of the specification and can be passed along to the solver, which can then exploit it directly. Another example of such ideas is that certain tools require that symmetry information should be provided by the user. For example, Junttila et al. [28] rely on a user-provided graph that captures the symmetries of a CNF formula, instead of relying on the solver to detect them from the formula itself. A final example is symmetry handling in Benders decomposition in mixed integer programming (MIP) [27]. To achieve this, the symmetries are typically handed directly to the solver instead of being detected from a problem encoding. Consider, for instance, solving a Hamiltonian cycle problem using MIP. A compact encoding of the connectivity constraints typically introduces auxiliary variables, which can obscure the original graph symmetries and complicate symmetry detection, while formulations that preserve symmetries explicitly are often much less practical. A simpler and more effective approach is to pass the symmetries of the original graph directly to the solver. In CP terminology, what this means is that in some cases the symmetries of a global constraint (here, circuit) are very easy to detect, while these symmetries are obscured or even destroyed by a decomposition of that global constraint. As a consequence, algorithms that use such a decomposition should better be provided with the symmetry information of the original global constraint, instead of relying on detecting it from the decomposition.

Structure of the Paper. The rest of this paper is structured as follows. In Section 2, we briefly introduce the necessary preliminaries. We describe our framework in Section 3 and analyze existing transformations in Section 4. We present our experimental evaluation in Section 5 and conclude in Section 6.

2 Preliminaries

As usual, we will work with variables, each with an associated domain; we write $\mathcal{D}(v)$ for the domain of variable v. A set of variables is called a *vocabulary*. An *assignment* α to a vocabulary $\mathcal{V}$ is a mapping sending variables $v \in \mathcal{V}$ to values from their domain. If $\mathcal{V}' \subseteq \mathcal{V}$ and α is a $\mathcal{V}$-assignment, we write $\alpha|_{\mathcal{V}'}$ for the restriction of α to $\mathcal{V}'$. Additionally, given two vocabularies $\mathcal{V}, \mathcal{V}'$ with $\mathcal{V}' \subseteq \mathcal{V}$, we will say that a $\mathcal{V}$-assignment α *extends* a $\mathcal{V}'$-assignment α' if α and α' agree on all variables in $\mathcal{V}'$, i.e., if $\alpha|_{\mathcal{V}'} = \alpha'$. A *constraint* over $\mathcal{V}$ of variables

is a set of assignments to $\mathcal{V}$. As usual, in practice this set is often defined implicitly.
A *constraint problem* $\mathcal{P}$ is a tuple $\langle \mathcal{V}_\mathcal{P}, \mathcal{F}_\mathcal{P} \rangle$, consisting of:

- a vocabulary $\mathcal{V}_\mathcal{P}$,
- and a set of constraints $\mathcal{F}_\mathcal{P}$, where each constraint is over a subset $\mathcal{V}_C$ of $\mathcal{V}_\mathcal{P}$.

A set of constraints will be referred to as a *formula* below. If $\mathcal{P}$ is a constraint problem, we will write $\text{voc}(\mathcal{P})$ and $\text{form}(\mathcal{P})$ to refer to its vocabulary and formula respectively. A *solution* of a constraint problem is an assignment α that satisfies the formula (denoted $\alpha \models \mathcal{F}_\mathcal{P}$), i.e., such that for every constraint $\text{C} \in \mathcal{F}_\mathcal{P}$, $\alpha|_{\mathcal{V}_\text{C}} \in \text{C}$.

A *semantic symmetry* of a constraint problem is a permutation π of assignments that preserves satisfaction, i.e., for all assignments α, it holds that $\pi(\alpha) \models \mathcal{F}_\mathcal{P}$ if and only if $\alpha \models \mathcal{F}_\mathcal{P}$. In practice, often only restricted classes of these symmetries are considered. In CP, these symmetries are often induced by either a permutation of the variables (i.e., *variable symmetries*) or a permutation of the values in some domains (i.e., *value symmetries*). Furthermore, it is common to focus on *syntactic symmetries*: symmetries that can be detected from the syntactic representation of $\mathcal{F}_\mathcal{P}$. An in-depth review of symmetry in CP is given in the Handbook of Constraint Programming [24].

3 A Framework for Symmetry and Transformations

We are interested in studying *symmetry-aware* transformations. Usually transformations are viewed as taking one constraint problem and producing another one. Such a transformation is "perfect" if it essentially preserves the set of solutions.

For our work, however, we do not study a problem $\mathcal{P}$ in isolation, but we add two more components. The first component is a group G of symmetries of $\mathcal{P}$. The intuition is that we are not interested in *all* solutions of $\mathcal{P}$, but only in solutions that are "structurally different". Formally, we say two solutions are *equivalent modulo* G if one can be obtained from the other by applying a permutation of G. In a symmetry-aware setting, we seek one representative per equivalence class of this relation; that is, we care about solutions *modulo* G. This corresponds to common practice. For example, when we are only interested in the satisfiability of $\mathcal{P}$, it suffices to find one satisfying solution modulo G. Also in enumeration tasks, one often wants to list solutions only up to isomorphism, for instance when studying combinatorial objects. The situation where G is the trivial group corresponds to the classical setting where we are interested in all solutions of $\mathcal{P}$. When we study transformations, it will not be important that they preserve all solutions, but only that they preserve the equivalence classes of solutions modulo G. The second component we add is a set $\mathcal{V}$ of *projection variables*. The intuition underlying this is that we are not interested in full solutions of $\mathcal{P}$, but only in the values they take on $\mathcal{V}$. All variables outside of $\mathcal{V}$ can be thought of as auxiliary variables that might be useful for defining $\mathcal{P}$, but are irrelevant for the user.

The interaction between the group G and projection variables $\mathcal{V}$ is non-trivial: if we only care about the value of solutions on $\mathcal{V}$, then it might not make sense to consider solutions modulo symmetries that map *complete* assignments. This leads to the following definitions.[1]

▶ **Definition 1** (Projected Symmetry). *Let $\mathcal{P}$ be a constraint problem and $\mathcal{V} \subseteq \text{voc}(\mathcal{P})$. A permutation π of $\mathcal{V}$-assignments is called a* projected symmetry of $\mathcal{P}$ with respect to $\mathcal{V}$ *(or, in brief, a $\mathcal{V}$-symmetry of $\mathcal{P}$) if for each $\mathcal{V}$-assignment α, it holds that α can be extended to a solution of $\mathcal{P}$ if and only if $\pi(\alpha)$ can be extended to a solution of $\mathcal{P}$.*

[1] Bleukx et al. [13] used the terminology *partial symmetry* for what is called a *projected symmetry* here.

▶ **Definition 2** (Symmetry-Aware Problem). *A* symmetry-aware problem *is a triple* $\langle \mathcal{P}, \mathcal{V}, G \rangle$ *where* $\mathcal{P}$ *is a constraint problem,* $\mathcal{V} \subseteq \text{voc}(\mathcal{P})$ *and* G *is a group of* $\mathcal{V}$*-symmetries of* $\mathcal{P}$*.*

Given a symmetry-aware problem $\langle \mathcal{P}, \mathcal{V}, G \rangle$, we call two solutions α and β of $\mathcal{P}$ equivalent (and denote this as $\alpha \equiv_G \beta$) if there is a symmetry $\pi \in G$ such that $\pi(\alpha|_{\mathcal{V}}) = \beta|_{\mathcal{V}}$. With a symmetry-aware problem, we are interested in solutions of $\mathcal{P}$ modulo this equivalence relation. A symmetry-aware problem can be used for different purposes; sometimes one might want to *enumerate* all equivalence classes of solutions of $\mathcal{P}$, other times one might want to *count* them and in other applications we might only be interested in the *satisfiability* of the problem (whether it admits at least one solution). In the last case, $\mathcal{V}$ and G can essentially be ignored: this will not influence the satisfiability. However, it is still useful to include them since symmetry information can be exploited by solvers to speed up search. Naturally, depending on the purpose, some transformations can be well-behaved or not. In the rest of this section, we study properties of such transformations.

Given the generality of our framework, transformations of (symmetry-aware) problems will not be defined for *all* possible problems but only for restricted classes. For instance, in Section 4, we will define transformations that start from problems where all variables are Boolean and all constraints are expressed as pseudo-Boolean formulas. One important aspect of transformations in practice is that we should always be able to translate solutions of the result back to solutions of the original problem. This leads us to the following definition.

▶ **Definition 3.** *Let* $\mathcal{C}$ *be a class of symmetry-aware constraint problems. A* transformation τ on $\mathcal{C}$ *is a function that maps each* $\langle \mathcal{P}, \mathcal{V}, G \rangle \in \mathcal{C}$ *to*

1. *another symmetry-aware problem* $\langle \mathcal{P}', \mathcal{V}', G' \rangle$ *denoted as* $\tau(\langle \mathcal{P}, \mathcal{V}, G \rangle)$ *and*
2. *a mapping* ρ_τ *from* $\mathcal{V}'$*-assignments to* $\mathcal{V}$*-assignments.*

▶ Remark 4. The intuition underlying ρ_τ is that given a (projected) solution of $\mathcal{P}'$, it reconstructs a projected solution of $\mathcal{P}$. Formally speaking, we defined ρ_τ here as a total function on $\mathcal{V}'$-assignments, but in practice it will often only be defined on a set of assignments that includes at least all solutions of $\mathcal{P}'$. To see why, consider a transformation that encodes integer variables V as Boolean variables with indicator variables $b_{V=k}$ (also known as a one-hot encoding). In this case, $\mathcal{P}'$ will only admit solutions in which exactly one indicator variable is true, and it suffices to define ρ_τ on such assignments. However, for mathematical convenience, we will consider it to be a total function (for all definitions that follow, the value of ρ_τ on non-solutions of $\mathcal{P}'$ is in this case irrelevant).

Obviously, not every transformation is useful; so far we have not included a single restriction on what a transformation is allowed to produce. The following definition establishes some potentially desired properties of reasonable transformations.

▶ **Definition 5.** *Let* τ *be a transformation on* $\mathcal{C}$*, and assume* τ *maps* $\langle \mathcal{P}, \mathcal{V}, G \rangle$ *to* $\langle \mathcal{P}', \mathcal{V}', G' \rangle$ *and* ρ_τ*. We call the application of* τ *on* $\langle \mathcal{P}, \mathcal{V}, G \rangle$

reconstructing *if for each projected solution* α' *of* $\mathcal{P}'$*,* $\rho_\tau(\alpha')$ *is a projected solution of* $\mathcal{P}$*;*

solution preserving *if for each projected solution* α *of* $\mathcal{P}$*, there exists a projected solution* α' *of* $\mathcal{P}'$ *such that* $\rho_\tau(\alpha') \equiv_G \alpha$*;*

backwards symmetry invariant *if for all two projected solutions* α' *and* β' *of* $\mathcal{P}'$*, if* $\alpha' \equiv_{G'} \beta'$*, then also* $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$*;*

forwards symmetry invariant *if for all two projected solutions* α' *and* β' *of* $\mathcal{P}'$*, whenever* $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$*, it also holds that* $\alpha' \equiv_{G'} \beta'$*.*

For any of the above properties, we say τ *satisfies the property if all applications of* τ *(on any problem in* $\mathcal{C}$*) satisfy it.*

Let us motivate the properties we just defined. The core idea of the reconstruction function ρ_τ is that it should make the transformation usable in practice. In particular, when applying a transformation τ, followed by a call to a solver on the resulting problem, it should be possible to reconstruct a solution to the original problem from a solution given by the solver. The *reconstructing* property says precisely this: ρ_τ maps any solution of the resulting problem back to a solution of the original problem. Vice versa, we may also want the other direction, namely that every solution of the original problem is essentially preserved. Formulating this property is more subtle, however, due to the involvement of symmetries. Indeed, the *solution preserving* property does not require that *any* solution of $\mathcal{P}$ should be recoverable, but only that its projection should be recoverable up to a symmetry in G.

The final two properties are about the preservation of the symmetry relation. First, the *backwards symmetry invariant* property says that whenever two solutions of the resulting problem are symmetric, then their reconstructions should also be symmetric. This means in particular that if we are only interested in a single representative of each equivalence class of solutions modulo G' (by the definition of symmetry-aware problems), we would not lose any equivalence classes when reconstructing the solutions. Vice versa, the *forwards symmetry invariant* property says that whenever two reconstructions of solutions are symmetric, then the original solutions should also have been symmetric. This means in particular that we are not creating new equivalence classes of solutions out of thin air.

The following proposition establishes some basic relations between the properties we just defined. Proofs of all results are deferred to Appendix A.

▶ **Proposition 6.** *The following properties hold.*

- *If τ is reconstructing then it is unsatisfiability preserving, meaning that it maps unsatisfiable problems to unsatisfiable problems.*
- *If τ is solution preserving then it is satisfiability preserving, i.e., it maps satisfiable problems to satisfiable problems.*

The following proposition essentially tells us that transformations that satisfy all the above properties indeed preserve the set of projected solutions modulo G.

▶ **Proposition 7.** *Assume τ is a transformation that is reconstructing, solution preserving, backwards symmetry invariant and forwards symmetry invariant. In this case, ρ_τ induces a bijection between the equivalence classes of projected solutions of $\mathcal{P}$ modulo G and the equivalence classes of projected solutions of $\mathcal{P}'$ modulo G'.*

We call a transformation that satisfies all four properties of Definition 5 a *perfect transformation*. A consequence of the previous proposition is that a perfect transformation can safely be used to enumerate or count the equivalence classes of solutions modulo G.

4 Analysis of Existing Transformations

We now describe how existing transformations fit within our symmetry-aware framework, starting with symmetry detection and breaking, and continuing with translations, simplifications, and variable-introduction techniques.

4.1 Symmetry-Based Transformations

Complete Symmetry Breaking Symmetry breaking extends the original problem with a set of additional constraints that eliminate symmetric solutions. We will refer to the set of symmetry breaking constraints as SBC. This set of symmetry breaking constraints is *sound*

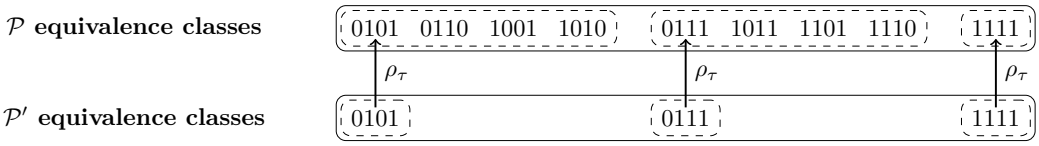


Figure 1 Complete symmetry breaking: each equivalence class of $\mathcal{P}$ corresponds to exactly one solution in $\mathcal{P}'$. Each bitstring represents an assignment to the variables (x_1, x_2, x_3, x_4), where the leftmost bit corresponds to x_1.

if it preserves at least one assignment of each equivalence class modulo G, and it is *complete* if it preserves at most one assignment of each equivalence class modulo G.

Given the symmetry-aware problem $\langle \mathcal{P}, \mathcal{V}, G \rangle$, symmetry breaking aims to obtain a new symmetry-aware problem that preserves exactly one solution per equivalence class modulo G. Concretely, symmetry breaking constraints are appended to the original formula, and if necessary, the vocabulary is extended with new variables used to express these constraints.

A *complete* symmetry breaking transformation adds a sound and complete set of symmetry breaking constraints to the original problem: it maps $\langle \mathcal{P}, \mathcal{V}, G \rangle$ to $\langle \mathcal{P} \cup \mathrm{SBC}, \mathcal{V}, \{id\} \rangle$, where SBC is a sound and complete set of symmetry breaking constraints for G. The following proposition states that this kind of transformation satisfies all desired properties.

▶ **Proposition 8.** *Every complete symmetry breaking transformation τ with the reconstruction function $\rho_\tau = id$ is perfect.*

We illustrate complete symmetry breaking as a transformation in the following example.

▶ **Example 9.** Consider the symmetry-aware problem $\langle \mathcal{P}, \mathcal{V}, G \rangle$ where $\mathcal{P} = \langle \mathcal{F}_\mathcal{P}, \mathcal{V} \rangle$ is a CNF formula $\mathcal{F}_\mathcal{P} = \{x_1 \vee x_2, x_3 \vee x_4\}$ over the vocabulary $\mathcal{V} = \{x_1, x_2, x_3, x_4\}$, and G is the group of syntactic variable symmetries of $\mathcal{P}$ (i.e., the permutations of $\mathcal{V}$ that map the formula $\mathcal{F}$ to itself). Concretely, G is generated by the set of permutations swapping the variables within the clauses, or the clauses themselves; $G = \langle (x_1\ x_2), (x_3\ x_4), (x_1\ x_3)(x_2\ x_4) \rangle$. The goal is to enumerate all solutions of $\mathcal{P}$ modulo G. To do this, we apply a complete symmetry breaking transformation τ, obtaining the new symmetry-aware problem $\langle \mathcal{P}', \mathcal{V}, \{id\} \rangle$ where $\mathcal{P}' = \langle \mathcal{F}_\mathcal{P} \cup \mathrm{SBC}, \mathcal{V} \rangle$, and SBC is the following set of clauses:

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

It can be checked that SBC is indeed a sound and complete set of symmetry breaking constraints for G. In fact, these clauses are constructed following a standard approach to post *lex-leader* constraints [16], which preserve the lexicographically (w.r.t. a fixed variable ordering) smallest representative from each equivalence class.

Because SBC is sound and complete, the remaining symmetry group is trivial, and $\mathcal{P}'$ contains exactly one solution for each equivalence class of $\mathcal{P}$ modulo G. These equivalence classes are illustrated in Figure 1. The reconstruction function $\rho_\tau = id$ maps each solution of $\mathcal{P}'$ to the chosen representative (concretely, the lexicographically minimal solution) of each corresponding equivalence class in $\mathcal{P}$.

Partial Symmetry Breaking For some particular groups, it is indeed possible to efficiently generate complete symmetry breaking constraints in practice [20]. However, achieving complete symmetry breaking is considered computationally infeasible in most cases [6, 16]. As an alternative, partial symmetry breaking techniques can be used, which add a set of

symmetry breaking constraints that is sound but not necessarily complete for G, preserving at least one solution per equivalence class of solutions of $\mathcal{P}$ modulo G.

In the case of complete symmetry breaking, all symmetries are broken, hence the remaining symmetries are reduced to the trivial group. In the case of partial symmetry breaking there are no such guarantees. A partial symmetry breaking transformation adds a sound set of symmetry breaking constraints to the original problem, transforming the original symmetry-aware problem $\langle \mathcal{P}, \mathcal{V}, G \rangle$ to a new symmetry-aware problem $\langle \mathcal{P}', \mathcal{V}, G' \rangle$ with $G' \subseteq G$ a subgroup of the original group of symmetries. The following proposition summarizes which properties are satisfied by this transformation.

▶ **Proposition 10.** *Partial symmetry breaking is reconstructing, solution preserving and backwards symmetry invariant, but not necessarily forwards symmetry invariant.*

We illustrate two partial symmetry breaking transformations in the following example.

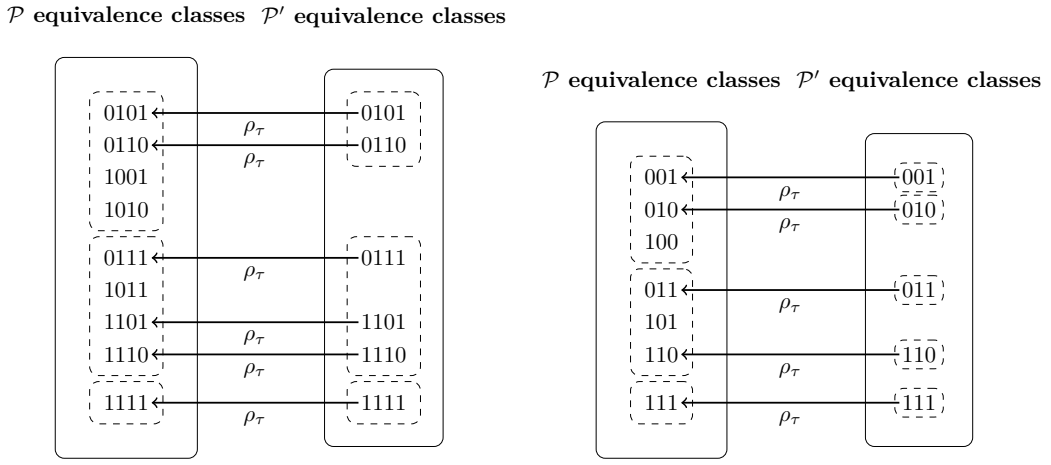
▶ **Example 11.** This example builds on Example 9, illustrating partial symmetry breaking. Consider the case where a complete and sound subset of the lexicographic symmetry breaking constraints are added for a subgroup of G. Concretely, let SBC be the set of constraints $\{\neg x_1 \vee x_2\}$, which is satisfied precisely by those α for which $\alpha \leq_{\text{lex}} (x_1\ x_2)(\alpha)$, breaking only the symmetry $(x_1\ x_2)$ between the variables in the first clause.

Because the symmetries between clauses and the variables in the second clause were not broken, one can define G' to be the largest subgroup of G that contains only unbroken symmetries (i.e., those that are still symmetries of $\mathcal{P}'$). Concretely, here this would mean that $G' = \langle (x_3\ x_4) \rangle$; no other permutations in G are symmetries of $\mathcal{P}'$. For instance, for the permutation $\pi = (x_1\ x_3)(x_2\ x_4)$, we see that it maps the assignment 0110 (a solution of $\mathcal{P}'$) to the non-assignment 1001. From Proposition 10, we know that this transformation is reconstructing, solution preserving and backwards symmetry invariant. In this case, the transformation is also forwards symmetry invariant as shown in Figure 2a, but as we will illustrate in the next example that this is not the case in general.

In the previous example, we were able to find a non-trivial subgroup of G that captures the symmetries that remain after partial symmetry breaking. However, in general, the unbroken symmetries of $\mathcal{P}$ are not necessarily symmetries of $\mathcal{P}'$. The following example illustrates a case where even though not all symmetries are broken, the only subgroup of G we can choose for G' is the trivial group.

▶ **Example 12.** Consider the symmetry-aware problem $\langle \mathcal{P}, \mathcal{V}, G \rangle$ where $\mathcal{P} = \langle \mathcal{F}_\mathcal{P}, \mathcal{V} \rangle$ is the CNF formula $\mathcal{F}_\mathcal{P} = \{x_1 \vee x_2 \vee x_3\}$ over the vocabulary $\mathcal{V} = \{x_1, x_2, x_3\}$, and G is the group of syntactic variable symmetries of $\mathcal{P}$ (i.e., all possible permutations of $\mathcal{V}$). We now apply a partial symmetry breaking transformation τ to this problem, by adding $\text{SBC} = \{\neg x_1 \vee x_2\}$, i.e., a single constraint that breaks the symmetry between x_1 and x_2. We know that SBC is sound, and hence preserves at least one solution per equivalence class of $\mathcal{P}$ modulo G. However, because SBC is not complete w.r.t. G, multiple solutions per equivalence class might remain. For instance the symmetry $(x_1\ x_3)$ is not broken by the added constraints. However, none of the symmetries of $\mathcal{P}$ are symmetries of $\mathcal{P}'$ and hence τ is not forwards symmetry invariant. We illustrate this in Figure 2b.

Symmetry Detection Next, we use the framework to analyse symmetry detection. Symmetry detection consists of finding a group G representing the syntactic symmetries of a given problem. This is usually done by transforming the problem into a graph in such a way



(a) Partial symmetry breaking, case 1 **(b)** Partial symmetry breaking, case 2

Figure 2 Partial symmetry breaking for case 1 (left) and case 2 (right). Bitstrings represent assignments to (x_1, x_2, x_3, x_4) in case 1 and (x_1, x_2, x_3) in case 2, with the leftmost bit corresponding to x_1.

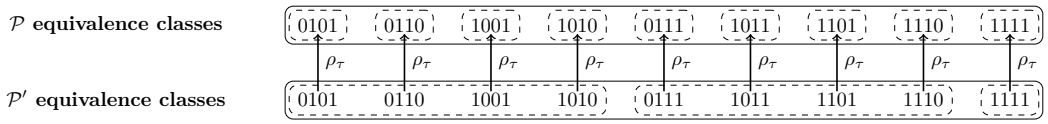


Figure 3 Symmetry detection: each solution of $\mathcal{P}$ corresponds to a solution of $\mathcal{P}'$, however, equivalent solutions in $\mathcal{P}'$ can map to non-equivalent solutions in $\mathcal{P}$. Each bitstring represents an assignment to the variables (x_1, x_2, x_3, x_4), where the leftmost bit corresponds to x_1.

that the automorphisms of said graph correspond to the syntactic symmetries of the problem. Then, a graph automorphism tool such as NAUTY [29], TRACES [29], or DEJAVU [8] is used.

We model this process as a transformation τ that maps a symmetry-aware problem $\langle \mathcal{P}, \mathcal{V}, \{\text{id}\}\rangle$ with the trivial group to a symmetry-aware problem $\langle \mathcal{P}, \mathcal{V}, G\rangle$ where G is a subgroup of the semantic symmetries of $\mathcal{P}$. Because the original problem has the trivial group as its symmetries, each projected solution of $\mathcal{P}$ forms its own equivalence class (modulo $\{\text{id}\}$). After the transformation, these solutions are partitioned into equivalence classes modulo G. The following proposition states the properties of this transformation.

▶ **Proposition 13.** *Symmetry detection is a transformation that is reconstructing, solution preserving, and forwards symmetry invariant, but not backwards symmetry invariant.*

The symmetry detection transformation is illustrated in the following example.

▶ **Example 14** (Example 9 continued)**.** Imagine we start with the symmetry-aware problem $\langle \mathcal{P}, \mathcal{V}, \{\text{id}\}\rangle$. The transformation τ maps this problem to the symmetry-aware problem $\langle \mathcal{P}, \mathcal{V}, G\rangle$, where $G = \langle (x_1\ x_2), (x_3\ x_4), (x_1\ x_3)(x_2\ x_4)\rangle$ is the group of syntactic symmetries of $\mathcal{P}$. Because the original problem has the trivial group as its symmetries, each projected solution of $\mathcal{P}$ forms its own equivalence class. The transformed problem $\mathcal{P}'$ on the other hand, contains the same projected solutions partitioned into equivalence classes modulo G as illustrated in Figure 3.

4.2 Formalism-Based Transformations

The framework is also well-suited to analyze transformations that do not explicitly focus on symmetries. A prominent example of such transformations are encodings of one formalism into another formalism. We take encodings of PB constraints into CNF here as a guiding example. These encodings typically introduce fresh variables and constraints to represent the original PB constraints in CNF form. Most commonly-used encodings will (inadvertently) remove certain symmetries of the original PB problem, as we will illustrate later. By analyzing these encodings within our framework, we can guarantee that the symmetry information is passed on through the encoding.

One encoding, often used to encode cardinality constraints is the totalizer encoding [9]. In this encoding, fresh variables are added, with as goal to count the cardinality of a subset of the original variables. However, by choosing those subsets to count, the encoding actually enforces additional structure on the variables, and most symmetries of the original PB problem are no longer symmetries of the resulting CNF. As a consequence, if the symmetry information is not passed on through the transformation (which is the de facto standard nowadays), it can no longer be recovered from the CNF and hence cannot be exploited.

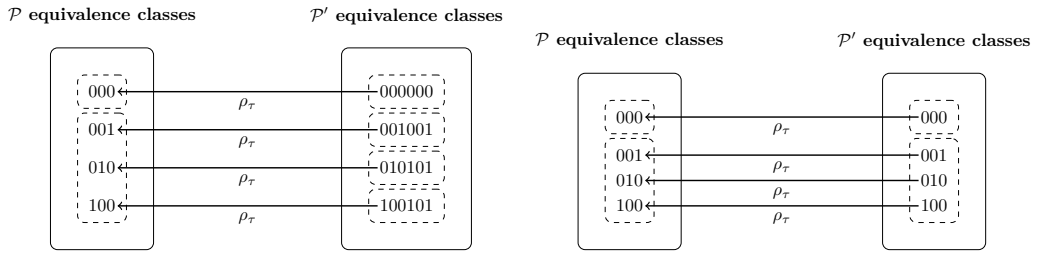
Making this a bit more formal, the *totalizer encoding* [9] transforms a cardinality constraint $\sum_{i=1}^{n} x_i \leq k$ into CNF using a binary tree structure. The variables form the leaves of a binary tree, and each node i of the tree is associated with the set V_i of variables in its subtree. For each internal node i, auxiliary variables $t_{i,j}$ (for $j \in \{1, \ldots, |V_i|\}$) are introduced, where $t_{i,j}$ holds if at least j variables in V_i are true. Clauses then ensure that these counts are propagated from children to parents; for an internal node i with children l and r, the clause $(\neg t_{l,a} \vee \neg t_{r,b} \vee t_{i,a+b})$ is added for all valid a, b. The cardinality constraint itself is enforced by the unit clause $(\neg t_{root,k+1})$.

▶ **Example 15.** The goal is to transform the PB constraint $x_1 + x_2 + x_3 \leq 1$ into CNF using the totalizer encoding. First, the variables are organized into a binary tree, where x_1 and x_2 are children of an internal node V_1, and V_1 together with x_3 are children of the root V. The encoding then introduces the auxiliary Boolean variables $t_{i,j}$ for $1 \leq i \leq 2$ and $1 \leq j \leq \mid V_i \mid$, where $V_1 = \{x_1, x_2\}$ and $V = V_2 = \{x_1, x_2, x_3\}$. The variable $t_{i,j}$ is true if at least j variables in V_i are true. Using these auxiliary variables, the constraint can be translated in CNF as:

$(\neg x_1 \vee t_{1,1}), (\neg x_2 \vee t_{1,1}),$	at least 1 true variable in V_1
$(\neg x_2 \vee \neg x_1 \vee t_{1,2}),$	at least 2 true variables in V_1
$(\neg t_{1,1} \vee t_{2,1}), (\neg x_3 \vee t_{2,1}),$	at least 1 true variable in V_2
$(\neg t_{1,2} \vee t_{2,2}), (\neg t_{1,1} \vee \neg x_3 \vee t_{2,2}),$	at least 2 true variables in V_2
$(\neg t_{2,2})$	at most 1 true variable overall

We define two ways of viewing the totalizer encoding as a transformation in our framework. The first and naive way, corresponds to the standard way of doing things, where we do not explicitly pass on the symmetry information, and use all variables in the resulting CNF as projection variables. The second way corresponds to a more careful way of doing things, where we explicitly pass on the symmetry information, and only take the original PB variables as projection variables. Formally, these transformations are defined as follows.

▶ **Definition 16.** *The* naive totalizer transformation *takes as input a symmetry-aware PB problem $\langle \mathcal{P}, \mathcal{V}, G \rangle$ where $\mathcal{P}$ is a set of pseudo-Boolean constraints over $\mathcal{V}$ and maps it to the symmetry-aware CNF problem $\langle \mathcal{P}', \mathcal{V}', \{id\} \rangle$, where $\mathcal{P}'$ is the result of applying the totalizer encoding to $\mathcal{P}$, and $\mathcal{V}'$ is the set of all variables in $\mathcal{P}'$. The* symmetry-aware totalizer



(a) Naive totalizer transformation **(b)** Symmetry-aware totalizer transformation

Figure 4 Totalizer transformation. Each bitstring represents an assignment to the variables $(x_1, x_2, x_3, t_{1,1}, t_{1,2}, t_{2,2})$ (left) and (x_1, x_2, x_3) (right), where the leftmost bit corresponds to x_1.

transformation *maps* $\langle \mathcal{P}, \mathcal{V}, G\rangle$ *to* $\langle \mathcal{P}', \mathcal{V}, G\rangle$*, where again* $\mathcal{P}'$ *is the result of applying the totalizer encoding to* $\mathcal{P}$*.*

In other words, when using the naive totalizer transformation, all symmetry information is lost in translation, reducing the remaining symmetries to the trivial group. By projecting out the auxiliary variables the symmetry can be preserved, as illustrated next.

▶ **Example 17** (Example 15 continued)**.** Note that in the original constraint, all variables are interchangeable, i.e., any permutation of the variables is a symmetry of the PB constraint. After applying the totalizer encoding, the variable x_3 is treated differently in the binary tree structure, and hence it is no longer interchangeable with x_1 and x_2. Intuitively speaking, the introduction of auxiliary variables messes up the symmetry structure.

When applying the *naive totalizer transformation*, the original problem is mapped to the symmetry-aware CNF problem $\langle \mathcal{P}', \mathcal{V}', \{id\}\rangle$, where $\mathcal{P}'$ is the totalizer encoding of the original PB constraint, and $\mathcal{V}'$ is the set of all variables in $\mathcal{P}'$. Define ρ_τ to be the function that maps each assignment of $\mathcal{V}'$ to the assignment of $\mathcal{V}$ by restricting it to the shared variables x_1, x_2, x_3. The freshly introduced variables destroy all symmetries of the original PB problem. As illustrated in Figure 4a, at least one solution per equivalence class of the original PB problem remains, but no symmetries are preserved: τ is solution preserving, reconstructing and backwards symmetry invariant but not forwards symmetry invariant.

When applying the *symmetry-aware totalizer transformation*, the original problem is mapped to the symmetry-aware CNF problem $\langle \mathcal{P}', \mathcal{V}, G\rangle$, where again $\mathcal{P}'$ is the totalizer encoding of the original PB constraint, but now $\mathcal{V}$ is the set of original PB variables, $\rho_\tau = id$ and G is the group of symmetries of the original PB constraint. As illustrated in Figure 4b, at least one solution per equivalence class of the original PB problem remains, and all symmetries are preserved, and hence τ is solution preserving, reconstructing, forwards symmetry invariant *and* backwards symmetry invariant. Hence, projecting out the auxiliary variables prevents the symmetries from getting lost in translation.

We formalize this in the following proposition.

▶ **Proposition 18.** *The naive totalizer transformation is reconstructing, solution preserving and backwards symmetry invariant, but not necessarily forwards symmetry invariant.*

We now show that when projecting out the auxiliary variables introduced by the totalizer encoding, all symmetries of the original PB problem can be preserved by the transformation. The following lemma states that the symmetries of the original PB problem are also projected symmetries of the CNF problem obtained by applying the totalizer encoding, when considering only the original PB variables as projection variables.

▶ **Lemma 19.** *When using the symmetry-aware totalizer transformation, symmetries of the original PB problem are also projected symmetries of the resulting CNF problem.*

Using the above lemma, we can show that when projecting out the variables, the totalizer transformation is a *perfect* transformation. This is formalized in the following proposition.

▶ **Proposition 20.** *The symmetry-aware totalizer transformation is reconstructing, solution preserving, backwards symmetry invariant and forwards symmetry invariant.*

4.3 Simplification Transformations

Last, we study pre- and inprocessing transformations using our framework. Such techniques aim to transform encoded problems, with the goal to decrease the solving time. This can be done, for example, by reformulating the problem with fewer constraints, or by decomposing global constraints into simpler forms that the final solver handles more efficiently. However, as was the case with formalism-based transformations, these transformations also risk removing syntactic symmetries from the problem.

4.3.1 Bounded Variable Addition

One popular preprocessing technique for SAT is *bounded variable addition* (BVA) [26]. BVA can be used to replace sets of clauses by smaller sets of clauses (at the cost of introducing new variables). It does so by identifying patterns in the formula that can easily be replaced. For example, the pattern

$$P = \left\{ \begin{matrix} (x_1 \vee y_1) & \dots & (x_1 \vee y_k) \\ (x_2 \vee y_1) & \dots & (x_2 \vee y_k) \\ & \dots & \\ (x_n \vee y_1) & \dots & (x_n \vee y_k) \end{matrix} \right\}$$

can easily be replaced by the equisatisfiable formula $\bigwedge_i^n (a \vee x_i) \wedge \bigwedge_j^k (\neg a \vee y_j)$. If $n+k < n \cdot k$, this is indeed a reduction of the number of clauses.

▶ **Example 21.** Consider a CNF encoding of the pseudo-Boolean constraint $x_1+x_2+\dots+x_6 \leq 1$. This constraint can be encoded in CNF using the pairwise encoding: $\mathcal{F} = \{(\neg x_i \vee \neg x_j) \mid i,j \in \{1,\dots,6\} \text{ and } i<j\}$, resulting in the following formula

$$\mathcal{F} = \left\{ \begin{matrix} (\neg x_1 \vee \neg x_2) & (\neg x_1 \vee \neg x_3) & (\neg x_1 \vee \neg x_4) & (\neg x_1 \vee \neg x_5) & (\neg x_1 \vee \neg x_6) \\ & (\neg x_2 \vee \neg x_3) & (\neg x_2 \vee \neg x_4) & (\neg x_2 \vee \neg x_5) & (\neg x_2 \vee \neg x_6) \\ & & (\neg x_3 \vee \neg x_4) & (\neg x_3 \vee \neg x_5) & (\neg x_3 \vee \neg x_6) \\ & & & (\neg x_4 \vee \neg x_5) & (\neg x_4 \vee \neg x_6) \\ & & & & (\neg x_5 \vee \neg x_6) \end{matrix} \right\}$$

We can use BVA on the highlighted clauses, resulting in the following formula

$$\mathcal{F}' = \left\{ \begin{matrix} (\neg x_1 \vee \neg x_2) & (\neg x_1 \vee \neg x_3) & (\neg x_1 \vee a) & (\neg x_2 \vee a) & (\neg x_3 \vee a) \\ & (\neg x_2 \vee \neg x_3) & (\neg x_4 \vee \neg a) & (\neg x_5 \vee \neg a) & (\neg x_6 \vee \neg a) \\ & & & (\neg x_4 \vee \neg x_5) & (\neg x_4 \vee \neg x_6) \\ & & & & (\neg x_5 \vee \neg x_6) \end{matrix} \right\}$$

Note that in this example, originally all of the variables are interchangeable. However, after applying the BVA pre-processing step, the variables x_1, x_2 and x_3 are no longer

interchangeable with the variables x_4, x_5 and x_6. Similar to the discussion on the totalizer encoding in Section 4.2, this is because the introduction of the auxiliary variable a enforces additional structure on the variables. Again, when projecting out the auxiliary variables introduced by the BVA, all symmetries of the original problem can be preserved.

First, we define the two ways of viewing the BVA transformation as a transformation in our framework.

▶ **Definition 22.** *A* naive BVA transformation *takes as input a symmetry-aware CNF problem* $\langle \mathcal{P}, \mathcal{V}, G \rangle$ *and maps it to the symmetry-aware CNF problem* $\langle \mathcal{P}', \mathcal{V}', \{id\} \rangle$*, where* $\mathcal{P}'$ *is the result of applying BVA pre-processing (for a given heuristics for choosing which patterns to apply it on) to* $\mathcal{P}$*, and* $\mathcal{V}'$ *is the set of all variables in* $\mathcal{P}'$*. A* symmetry-aware BVA transformation *maps* $\langle \mathcal{P}, \mathcal{V}, G \rangle$ *to* $\langle \mathcal{P}', \mathcal{V}, G \rangle$*, where again* $\mathcal{P}'$ *is the result of applying BVA pre-processing to* $\mathcal{P}$*.*

The following lemma states that when projecting out the auxiliary variables introduced by the BVA, all symmetries of the original problem can be preserved by the transformation.

▶ **Lemma 23.** *When using the symmetry-aware BVA transformation, all symmetries of the original problem are projected symmetries of the resulting problem.*

With this lemma we can prove that the BVA pre-processing technique is reconstructing, solution preserving, backwards symmetry invariant and forwards symmetry invariant when projecting out the auxiliary variables introduced by the BVA simplification. This is formalised in the following proposition.

▶ **Proposition 24.** *Symmetry-aware BVA transformations are reconstructing, solution preserving, backwards symmetry invariant and forwards symmetry invariant. Naive BVA transformations are reconstructing, solution preserving and backwards symmetry invariant, but not necessarily forwards symmetry invariant.*

4.4 Structure-Based Extended Resolution

A well-known technique in lazy clause generation is to introduce auxiliary variables representing information about a combination of existing variables [14]. A smart choice of variables can shorten proofs, however, as with the totalizer encoding, it introduces additional structure on the variables that can destroy syntactic symmetries. We illustrate this using an example where a fully symmetric sum constraint is *decomposed* into partial sums.

4.4.1 Sum Constraint Decomposition

Take for example a simple sum constraint $\sum_{i=1}^{n} x_i \geq N$. This constraint can be decomposed into prefix-sums using auxiliary sum variables s_k for $k \in [0, \ldots, n]$, where

$$s_0 = 0,$$
$$s_i = s_{i-1} + x_i, \quad \text{for } 1 \leq i \leq n, \text{ and}$$
$$s_n \geq N.$$

This encodes that each variable s_k should equal the partial sum $\sum_{i=1}^{k} x_i$. However, by adding these partial sums to the original encoding, all syntactic symmetries are destroyed. Similar to the previous discussion, there are two ways of viewing the sum decomposition as a transformation in our framework.

▶ **Definition 25.** *The* naive sum decomposition *takes as input a symmetry-aware constraint problem* $\langle \mathcal{P}, \mathcal{V}, G \rangle$ *and maps it to the symmetry-aware constraint problem* $\langle \mathcal{P} \cup \mathcal{P}', \mathcal{V}', \{id\} \rangle$*, where* $\mathcal{P}'$ *is the prefix-sum decomposition of* $\mathcal{P}$*, and* $\mathcal{V}'$ *is the set of all variables in* $\mathcal{P}'$*. The* symmetry-aware sum decomposition *maps* $\langle \mathcal{P}, \mathcal{V}, G \rangle$ *to* $\langle \mathcal{P} \cup \mathcal{P}', \mathcal{V}, G \rangle$*, where again* $\mathcal{P}'$ *is the result of applying the prefix-sum decomposition to* $\mathcal{P}$*.*

The following lemma states that when projecting out the auxiliary variables introduced by the decomposition, all symmetries of the original problem can be preserved by the transformation.

▶ **Lemma 26.** *When using the symmetry-aware sum decomposition, all symmetries of the original problem are projected symmetries of the resulting problem.*

This lemma now allows us to analyze prefix-sum decompositions in our framework, as formalised in the following proposition.

▶ **Proposition 27.** *The symmetry-aware prefix-sum decomposition is perfect, while the naive prefix-sum decomposition is reconstructing, solution preserving and backwards symmetry invariant, but not necessarily forwards symmetry invariant.*

5 Validation on Pseudo-Boolean Translations

The framework we proposed is general and can be applied to a wide range of transformations, as discussed above. In this section, we show that it is not just of theoretical interest, but can result in more efficient solving. We showcase this on the specific case of translating pseudo-Boolean (PB) constraints into CNF.

5.1 Encoder with Auxiliary Graphs

To evaluate the proposed framework, we implemented a tool called FORKENCODER.[2] The tool's name reflects the fact that the tool transforms one input into two outputs. Concretely, FORKENCODER takes a PB formula in OPB format and outputs an equisatisfiable CNF formula and an auxiliary graph representing the symmetries of the original PB formula. In other words, this tool combines two of the transformations we discussed in Section 4: symmetry detection and the transformation from pseudo-Boolean constraints into CNF. For the translation from OPB to CNF, we use the PINDAKAAS library [12]. In our experiments we use the adder encoding for pseudo-Boolean constraints.

While performing this translation, the tool also constructs a graph that captures the symmetries of the original PB instance. Using a graph provides an efficient way to transfer this information without explicitly computing or manipulating the symmetries themselves. Concretely, following existing symmetry detection methods for pseudo-Boolean problems [4, 33], the tool constructs a (colored) graph which is a simplification of the parse tree of the original pseudo-Boolean instance. The automorphisms of this graph correspond exactly to the syntactic symmetries of the input instance and are preserved as *projected* symmetries of the translated instances.

To make use of this information, we modified the symmetry breaking tool SATSUMA [7]. Usually, SATSUMA constructs a graph representation of the input CNF formula and detects its symmetries. In our version, SATSUMA can instead read the auxiliary graph and use it directly for generating symmetry breaking constraints.

[2] The source code is available at `https://github.com/markusa4/forkencoder`.

5.2 Experimental Setup

To evaluate the proposed framework, we compare the effect of using the auxiliary graph generated by FORKENCODER to generate symmetry breaking constraints, against solely using the CNF. We obtain three different CNF formulas:

- `no_sbp`: the translated instance without symmetry breaking constraints,
- `sbp_graph`: the translated instance with symmetry breaking constraints generated using the auxiliary graph produced by the symmetry-aware transformation,
- `sbp_cnf`: the translated instance with symmetry breaking constraints generated using solely the CNF formula.

We then solve the obtained CNF formulas using the SAT solver KISSAT [11] to compare the performance of the added symmetry breaking clauses. The FORKENCODER and SATSUMA each get a time limit of 1800 seconds and a memory limit of 31GB. Similar to the SAT competition, KISSAT gets a time limit of 5000 seconds and a memory limit of 31GB as well. The experiments were performed on a machine with 2 Intel Xeon Gold 6240 CPUs running Rocky Linux 8.10 with Linux kernel 4.18.0.

We test the framework on the instances used in the decision track of the pseudo-Boolean competition 2025 [32][3] as our benchmark set. This set contains 502 instances, and covers a wide range of application domains and crafted benchmarks with varying levels of difficulty. The implementation of the encoder only supports coefficients up to 32 bits, excluding 24 instances from the competition that contain larger coefficients. Furthermore, one instance ran out of memory during encoding. Because our goal is to specifically compare the effect of the added symmetry breaking clauses, we filtered the remaining 477 instances to those where SATSUMA's underlying symmetry detection detects non-trivial symmetries (either using `sbp_graph` or `sbp_cnf`). This leaves us with 354 symmetric instances for the evaluation.

5.3 Benchmarks

The summary of our results can be found in Table 1 and Figure 5. We observe that using the auxiliary graph to generate the symmetry breaking constraints outperforms using solely the CNF. This is not only reflected in the number of solved instances, but also in the average solving time and the PAR-2 score. Furthermore, both configurations with symmetry breaking outperform the configuration without symmetry breaking.

In terms of preprocessing overhead, we note that SATSUMA encountered 6 out-of-memory errors when using the auxiliary graph, and 13 when using solely the CNF (including the 6 instances that caused out-of-memory errors when using the auxiliary graph). This indicates that detecting symmetries on the original problem formulation is easier than detecting them on the encoded CNF.

Furthermore, the SATSUMA algorithm searches for particular *structural symmetries*, so-called row, row-column and Johnson symmetries [18, 7], which can be broken more efficiently. In the scatter plot in Figure 5, we visualize whether more structural symmetry is found in the graph or CNF version. We observe that most benchmarks with improved performance coincide with more structural symmetry in the instance. Indeed, this goes both ways: most instances clearly above the diagonal have more structural symmetry in the CNF, whereas most instances clearly below the diagonal have more in the graph.

[3] `https://www.cril.univ-artois.fr/PB25/benchs/selected-PB25.tar`

	Num. Solved (out of 354)	Avg. Solving Time (s.)	PAR-2 score (s.)
`no_sbp`	247	1602.8	3114.1
`sbp_graph`	**276**	**1199.6**	**2301.3**
`sbp_cnf`	265	1372.5	2629.5

Table 1 SATSUMA and KISSAT runtime per configuration: The average solving time includes SATSUMA's running time. The PAR-2 score is computed by assigning a penalty of 2 times the time limit to each instance that was not solved within the time limit. The best results are marked in bold.

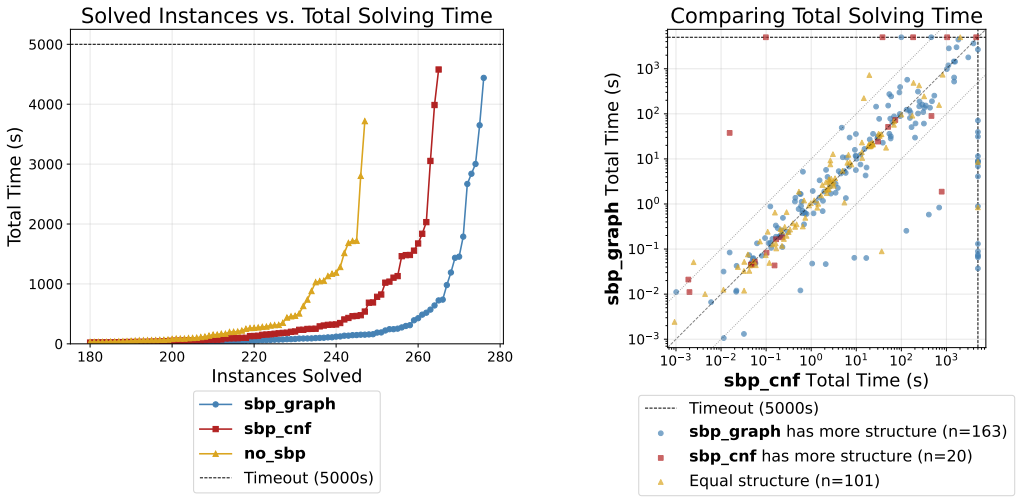


Figure 5 A cactus plot (left) and scatter plot (right) comparing the solving times of the configurations. Each point in the scatter plot is marked based on which configuration detected the most structural symmetry information. Specifically, a colored blue circle indicates that `sbp_graph` detected more structural symmetries than `sbp_cnf`, a red square indicates that `sbp_cnf` detected more structural symmetries than `sbp_graph`, and a yellow triangle indicates that both configurations detected the same amount of structural symmetries.

6 Conclusion

This paper describes a framework in which symmetry information is explicitly part of the specification, such that it can be passed along through different transformations. To do this, we introduce the notion of *symmetry-aware transformations*, and illustrate this new concept with several examples of transformations that occur in practice. To validate this framework, we implemented a tool that translates pseudo-Boolean constraints to CNF while also generating an auxiliary graph that captures the symmetries of the original PB problem. Experiments on instances from the pseudo-Boolean competition 2025 show that using the auxiliary graph to generate symmetry breaking constraints indeed preserves the structural symmetries of the original PB problem, which in turn leads to better performance when solving the resulting CNF formulas.

References

1. Ignasi Abío and Peter J. Stuckey. Conflict directed lazy decomposition. In Michela Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 70–85. Springer, 2012. doi:10.1007/978-3-642-33558-7_8.
2. Özgür Akgün, Mun See Chang, Ian P. Gent, and Christopher Jefferson. Breaking the symmetries of indistinguishable objects. In Guido Tack, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 22nd International Conference, CPAIOR 2025, Melbourne, VIC, Australia, November 10-13, 2025, Proceedings, Part I*, volume 15762 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2025. doi:10.1007/978-3-031-95973-8_10.
3. Özgür Akgün, Alan M. Frisch, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Conjure: Automatic generation of constraint models from problem specifications. *Artif. Intell.*, 310:103751, 2022. doi:10.1016/J.ARTINT.2022.103751.
4. Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. ShatterPB: symmetry-breaking for pseudo-Boolean formulas. In Masaharu Imai, editor, *Proceedings of the 2004 Conference on Asia South Pacific Design Automation: Electronic Design and Solution Fair 2004, Yokohama, Japan, January 27-30, 2004*, pages 883–886. IEEE Computer Society, 2004. doi:10.1109/ASPDAC.2004.179.
5. Markus Anders. SAT preprocessors and symmetry. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 1:1–1:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SAT.2022.1.
6. Markus Anders, Sofia Brenner, and Gaurav Rattan. The complexity of symmetry breaking beyond lex-leader. In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, Girona, Spain, September 2-6, 2024*, volume 307 of *LIPIcs*, pages 3:1–3:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.CP.2024.3.
7. Markus Anders, Sofia Brenner, and Gaurav Rattan. Satsuma: Structure-based symmetry breaking in SAT. *J. Artif. Intell. Res.*, 85, 2026. doi:10.1613/JAIR.1.18744.
8. Markus Anders and Pascal Schweitzer. Parallel computation of combinatorial symmetries. In *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.ESA.2021.6, doi:10.4230/LIPICS.ESA.2021.6.
9. Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of Boolean cardinality constraints. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003. doi:10.1007/978-3-540-45193-8_8.
10. Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New encodings of pseudo-Boolean constraints into CNF. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2009. doi:10.1007/978-3-642-02777-2_19.
11. Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In Marijn Heule, Ashlin Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, volume B-2024-1 of *Department of Computer Science Report Series B*, pages 8–10. University of Helsinki, 2024.
12. Hendrik Bierlee and Jip J. Dekker. Pindakaas. doi:10.5281/zenodo.10851855.

13 Ignace Bleukx, Hélène Verhaeghe, Bart Bogaerts, and Tias Guns. Exploiting symmetries in MUS computation. In Toby Walsh, Julie Shah, and Zico Kolter, editors, *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 11122–11130. AAAI Press, 2025. doi:10.1609/AAAI.V39I11.33209.

14 Geoffrey Chu and Peter J. Stuckey. Structure based extended resolution for constraint programming. *CoRR*, abs/1306.4418, 2013. URL: http://arxiv.org/abs/1306.4418, arXiv:1306.4418.

15 James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996*, pages 148–159. Morgan Kaufmann, 1996.

16 James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996*, pages 148–159. Morgan Kaufmann, 1996.

17 Jo Devriendt, Bart Bogaerts, and Maurice Bruynooghe. Symmetric explanation learning: Effective dynamic symmetry handling for SAT. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 83–100. Springer, 2017. doi:10.1007/978-3-319-66263-3_6.

18 Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. Improved static symmetry breaking for SAT. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 104–122. Springer, 2016. doi:10.1007/978-3-319-40970-2_8.

19 Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. On local domain symmetry for model expansion. *Theory Pract. Log. Program.*, 16(5-6):636–652, 2016. doi:10.1017/S1471068416000508.

20 Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models. In Van Hentenryck [34], pages 462–476. doi:10.1007/3-540-46135-3_31.

21 Alan M. Frisch, Warwick Harvey, Christopher Jefferson, Bernadette Martínez Hernández, and Ian Miguel. Essence : A constraint language for specifying combinatorial problems. *Constraints An Int. J.*, 13(3):268–306, 2008. doi:10.1007/s10601-008-9047-y.

22 Ian P. Gent, Warwick Harvey, and Tom W. Kelsey. Groups and constraints: Symmetry breaking during search. In Van Hentenryck [34], pages 415–430. doi:10.1007/3-540-46135-3_28.

23 Ian P. Gent, Karen E. Petrie, and Jean-François Puget. Symmetry in constraint programming. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 329–376. Elsevier, 2006. doi:10.1016/S1574-6526(06)80014-3.

24 Ian P. Gent, Karen E. Petrie, and Jean-François Puget. Symmetry in constraint programming. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 329–376. Elsevier, 2006. URL: https://www.sciencedirect.com/science/bookseries/15746526/2, doi:10.1016/S1574-6526(06)80014-3.

25 Andrew Grayland, Ian Miguel, and Colva M. Roney-Dougal. Snake lex: An alternative to double lex. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009,*

Proceedings, volume 5732 of *Lecture Notes in Computer Science*, pages 391–399. Springer, 2009. doi:10.1007/978-3-642-04244-7_32.

26 Andrew Haberlandt, Harrison Green, and Marijn J. H. Heule. Effective auxiliary variables via structured reencoding. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, Alghero, Italy, July 4-8, 2023*, volume 271 of *LIPIcs*, pages 11:1–11:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SAT.2023.11.

27 Christopher Hojny and Cédric Roy. A framework for handling and exploiting symmetry in benders' decomposition, 2025. URL: https://arxiv.org/abs/2511.22251, arXiv:2511.22251.

28 Tommi A. Junttila, Matti Karppa, Petteri Kaski, and Jukka Kohonen. An adaptive prefix-assignment technique for symmetry reduction. *J. Symb. Comput.*, 99:21–49, 2020. URL: https://doi.org/10.1016/j.jsc.2019.03.002, doi:10.1016/J.JSC.2019.03.002.

29 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. URL: https://doi.org/10.1016/j.jsc.2013.09.003, doi:10.1016/J.JSC.2013.09.003.

30 Christopher Mears, Maria Garcia de la Banda, Bart Demoen, and Mark Wallace. Lightweight dynamic symmetry breaking. *Constraints An Int. J.*, 19(3):195–242, 2014. doi:10.1007/S10601-013-9154-2.

31 Marc E. Pfetsch and Thomas Rehn. A computational comparison of symmetry handling methods for mixed integer programs. *Math. Program. Comput.*, 11(1):37–93, 2019. doi:10.1007/S12532-018-0140-Y.

32 Olivier Roussel, 2025. URL: https://www.cril.univ-artois.fr/PB25/.

33 Daimy Van Caudenberg and Bart Bogaerts. Symmetry and dominance breaking for pseudo-Boolean optimization. In Toon Calders, Celine Vens, Jefrey Lijffijt, and Bart Goethals, editors, *Artificial Intelligence and Machine Learning - 34th Joint Benelux Conference, BNAIC/Benelearn 2022, Mechelen, Belgium, November 7-9, 2022, Revised Selected Papers*, volume 1805 of *Communications in Computer and Information Science*, pages 149–166. Springer, 2022. doi:10.1007/978-3-031-39144-6_10.

34 Pascal Van Hentenryck, editor. *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-46135-3.

35 Neng-Fa Zhou, Håkan Kjellerstrand, and Jonathan Fruhman. *Constraint Solving and Planning with Picat*. Springer Briefs in Intelligent Systems. Springer, 2015. doi:10.1007/978-3-319-25883-6.

A Proofs

▶ **Proposition 6.** *The following properties hold.*

- *If τ is reconstructing then it is unsatisfiability preserving, meaning that it maps unsatisfiable problems to unsatisfiable problems.*
- *If τ is solution preserving then it is satisfiability preserving, i.e., it maps satisfiable problems to satisfiable problems.*

Proof of Proposition 6. If τ is reconstructing, it also preserves unsatisfiability. If this were not the case, the transformation τ could transform an unsatisfiable problem $\mathcal{P}$, into a satisfiable problem $\mathcal{P}'$. However, in this case, the reconstructing transformation τ must transform each projected solution α' of $\mathcal{P}'$ to a projected solution of $\mathcal{P}$, which is impossible since $\mathcal{P}$ is unsatisfiable.

Similarly, a transformation τ which is solution preserving, also preserves satisfiability. If this were not the case, the transformation τ could transform a satisfiable problem $\mathcal{P}$, into

an unsatisfiable problem $\mathcal{P}'$. However, τ is only solution preserving if for each projected solution α to $\mathcal{P}$, there exists an equivalent projected solution α' to $\mathcal{P}'$, which is impossible since $\mathcal{P}'$ is unsatisfiable. ◀

▶ **Proposition 7.** *Assume τ is a transformation that is reconstructing, solution preserving, backwards symmetry invariant and forwards symmetry invariant. In this case, ρ_τ induces a bijection between the equivalence classes of projected solutions of $\mathcal{P}$ modulo G and the equivalence classes of projected solutions of $\mathcal{P}'$ modulo G'.*

Proof of Proposition 7. We first show that ρ_τ induces a well-defined mapping between the equivalence classes. Assume α' and β' are two projected solutions of $\mathcal{P}'$ with $\alpha' \equiv_{G'} \beta'$. Because the transformation ρ_τ is reconstructing, it indeed maps α' (and β') to projected solutions of $\mathcal{P}$. We then have that $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$, since ρ_τ is also backwards symmetry invariant. Hence, ρ_τ induces a well-defined mapping between the equivalence classes.

Next, we show that this mapping is surjective. Assume α is a projected solution of $\mathcal{P}$. Since ρ_τ is solution preserving, there exists a projected solution α' of $\mathcal{P}'$ such that $\rho_\tau(\alpha') \equiv_G \alpha$. Hence, the mapping is surjective.

Finally, we show that the mapping is injective. Assume α' and β' are two projected solutions of $\mathcal{P}'$ such that $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$. Because ρ_τ is forwards symmetry invariant, we then have that $\alpha' \equiv_{G'} \beta'$. Hence, the mapping is injective. ◀

▶ **Proposition 8.** *Every complete symmetry breaking transformation τ with the reconstruction function $\rho_\tau = id$ is perfect.*

Proof of Proposition 8. Given a set SBC of sound and complete symmetry breaking constraints for G, define $\tau\colon \langle \mathcal{P}, \mathcal{V} \subseteq \mathcal{V}_\mathcal{F}, G\rangle \mapsto \langle \mathcal{P}', \mathcal{V}, G' = \{id\}\rangle$, where $\mathcal{P} = \langle \mathcal{F}, \mathcal{V}_\mathcal{F}\rangle$, $\mathcal{P}' = \langle \mathcal{F} \cup \text{SBC}, \mathcal{V}_\mathcal{F} \cup \mathcal{V}_\text{SBC}\rangle$, $\mathcal{V}_\text{SBC}$ is the set of fresh variables used in SBC and $\rho_\tau = id$.

Since $\mathcal{P}' \supseteq \mathcal{P}$ and $\rho_\tau = id$, every projected solution of $\mathcal{P}'$ is also a projected solution of $\mathcal{P}$, so τ is reconstructing.

Because SBC is sound, it preserves at least one projected solution per equivalence class of $\mathcal{P}$ modulo G. Hence, for each projected solution α of $\mathcal{P}$ there exists a projected solution α' of $\mathcal{P}'$ with $\rho_\tau(\alpha') \equiv_G \alpha$, so τ is solution preserving.

Since G' is trivial, we have that $\alpha' \equiv_{G'} \beta'$ implies that $\alpha' = \beta'$. Hence $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$, so τ is also backwards symmetry invariant.

Because SBC is sound and complete, $\mathcal{P}'$ has exactly one projected solution per equivalence class modulo G. Thus, for any two solutions α', β' of $\mathcal{P}'$, if $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$, then $\alpha' = \beta'$. Since G' is trivial, this implies $\alpha' \equiv_{G'} \beta'$, so τ is also forwards symmetry invariant.

◀

▶ **Proposition 10.** *Partial symmetry breaking is reconstructing, solution preserving and backwards symmetry invariant, but not necessarily forwards symmetry invariant.*

Proof of Proposition 10. Given a set SBC of sound symmetry breaking constraints for G, define $\tau\colon \langle \mathcal{P}, \mathcal{V} \subseteq \mathcal{V}_\mathcal{F}, G\rangle \mapsto \langle \mathcal{P}', \mathcal{V}, G'\rangle$, where $\mathcal{P} = \langle \mathcal{F}, \mathcal{V}_\mathcal{F}\rangle$, $\mathcal{P}' = \langle \mathcal{F} \cup \text{SBC}, \mathcal{V}_\mathcal{F} \cup \mathcal{V}_\text{SBC}\rangle$, and $G' \leq G$ is a group of $\mathcal{V}'$-symmetries of $\mathcal{P}'$. Last, define $\rho_\tau = id$.

For reconstructing and solution preserving, the proof is identical to that of complete symmetry breaking. We now show that τ is backwards symmetry invariant and not forwards symmetry invariant.

Assume α' and β' are two projected solutions of $\mathcal{P}'$ with $\alpha' \equiv_{G'} \beta'$. Since $\rho_\tau = id$, we have that $\rho_\tau(\alpha') = \alpha'$ and $\rho_\tau(\beta') = \beta'$. Because G' is a subgroup of G, we have that $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$. Hence, τ is backwards symmetry invariant.

However, because SBC is not complete, τ is not necessarily forwards symmetry invariant. Assume two solutions α', β' of $\mathcal{P}'$, with $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$. Because G' is a subgroup of G, it is not necessarily the case that $\alpha' \equiv_{G'} \beta'$, hence τ is not necessarily forwards symmetry invariant. ◀

▶ **Proposition 13.** *Symmetry detection is a transformation that is reconstructing, solution preserving, and forwards symmetry invariant, but not backwards symmetry invariant.*

Proof of Proposition 13. Let τ map $\langle \mathcal{P}, \mathcal{V}, \{\text{id}\}\rangle$ to $\langle \mathcal{P}' = \mathcal{P}, \mathcal{V}' = \mathcal{V}, G\rangle$ where G is a subset of the semantic symmetries of $\mathcal{P}$, and define $\rho_\tau = id$. Since $\mathcal{P}' = \mathcal{P}$ and $\rho_\tau = id$, every projected solution of $\mathcal{P}'$ is also a projected solution of $\mathcal{P}$, and vice versa. Hence, τ is reconstructing and solution preserving. Assume α' and β' are two projected solutions of $\mathcal{P}'$ such that $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$. This can only be the case if $\alpha' = \beta'$, since the original problem has the trivial group as its symmetries. Hence, we also have that $\alpha' \equiv_{G'} \beta'$, so τ is forwards symmetry invariant. Conversely, if $\alpha' \equiv_{G'} \beta'$, $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$ only holds if $\alpha' = \beta'$, since the original problem has the trivial group as its symmetries. Hence, τ is not backwards symmetry invariant. ◀

▶ **Proposition 18.** *The naive totalizer transformation is reconstructing, solution preserving and backwards symmetry invariant, but not necessarily forwards symmetry invariant.*

Proof of Proposition 18. Let τ be a naive totalizer transformation, mapping $\langle \mathcal{P}, \mathcal{V}, G\rangle$ to $\langle \mathcal{P}', \mathcal{V}', id\rangle$ where $\mathcal{P}'$ is the result of applying the totalizer encoding to $\mathcal{P}$ and $\mathcal{V}'$ contains all of the variables in $\mathcal{P}'$. Last, define ρ_τ to be the function that maps each assignment α' of $\mathcal{V}'$ to the assignment α of $\mathcal{V}$ obtained by restricting α' to the original PB variables. Every projected solution of $\mathcal{P}'$ is also a solution of $\mathcal{P}$, and vice versa. Hence, τ is reconstructing and solution preserving. Assume α' and β' are two projected solutions of $\mathcal{P}'$ with $\alpha' \equiv_{G'} \beta'$. Because $G' = \{id\}$, this can only be the case if $\alpha' = \beta'$. Hence we also have that $\rho_\tau(\alpha') = \rho_\tau(\beta')$, and as such that $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$. Hence, τ is backwards symmetry invariant.

It remains to show that τ is not forwards symmetry invariant. Consider two assignments α' and β' of $\mathcal{P}'$, where $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$ but $\alpha' \neq \beta'$. If $G' = \{id\}$, we have that $\alpha' \not\equiv_{G'} \beta'$. Hence, τ is not necessarily forwards symmetry invariant. ◀

▶ **Lemma 19.** *When using the symmetry-aware totalizer transformation, symmetries of the original PB problem are also projected symmetries of the resulting CNF problem.*

Proof of Lemma 19. A permutation $\pi \in G$ is a V'-symmetry of $\mathcal{P}'$ if for every V'-assignment α' of $\mathcal{P}'$, α' can be extended to a solution of $\mathcal{P}'$ if and only if $\pi(\alpha')$ can be extended to a solution of $\mathcal{P}'$. Hence, we need to show that each permutation of an assignment to the original PB variables indeed extends to a solutions of $\mathcal{P}'$ if and only if the original assignment extends to a solution of $\mathcal{P}'$.

This is indeed the case, since each satisfying assignment of the original PB problem can be extended to a satisfying assignment of the CNF problem by appropriately adjusting the auxiliary counter variables. Similarly, each non-satisfying assignment of the original PB problem cannot be extended to a satisfying assignment of the CNF problem, since the clauses enforcing the cardinality constraint will not be satisfied after adjusting the counters. As a result, we can safely preserve the original symmetry group G. ◀

▶ **Proposition 20.** *The symmetry-aware totalizer transformation is reconstructing, solution preserving, backwards symmetry invariant and forwards symmetry invariant.*

Proof of Proposition 20. Let τ map $\langle \mathcal{P}, \mathcal{V}, G \rangle$ to $\langle \mathcal{P}', \mathcal{V}', G' = G \rangle$ where $\mathcal{P}'$ is the result of applying the totalizer encoding to $\mathcal{P}$, $\mathcal{V}'$ contains only the original PB variables, and G' is the group of $\mathcal{V}'$-symmetries of $\mathcal{P}'$. Following Lemma 19, we have that G is a subgroup of the projected symmetries of $\mathcal{P}'$, and hence it is safe to define $G' = G$. Last, define $\rho_\tau = id$.

First, note that every projected solution of $\mathcal{P}'$ (i.e., every assignment to the original PB variables that can be extended to a solution of $\mathcal{P}'$) is also a solution of $\mathcal{P}$, and vice versa. Hence, τ is reconstructing and solution preserving.

Assume α' and β' are two projected solutions of $\mathcal{P}'$ with $\alpha' \equiv_{G'} \beta'$. Because $G' = G$ and $\rho_\tau = id$, we also have that $\rho_\tau(\alpha') \equiv_G \rho_\tau(\beta')$. Hence, τ is backwards symmetry invariant.

The opposite direction follows similarly. Hence, τ is also forwards symmetry invariant. ◀

▶ **Lemma 23.** *When using the symmetry-aware BVA transformation, all symmetries of the original problem are projected symmetries of the resulting problem.*

Proof of Lemma 23. We need to show that each permutation $\pi \in G$, is also a $\mathcal{V}$-symmetry of $\mathcal{P}'$, where $\mathcal{P}'$ is the result of applying BVA pre-processing to $\mathcal{P}$. This holds if for every $\mathcal{V}$-assignment α' of $\mathcal{P}'$, α' can be extended to a solution of $\mathcal{P}'$ if and only if $\pi(\alpha')$ can be extended to a solution of $\mathcal{P}'$.

This is the case, since each satisfying assignment of the original problem can be extended to a satisfying assignment of the problem after the BVA transformation, by appropriately setting the auxiliary variable introduced by the pattern replacement. Specifically, if in the original assignment a variable captured by the x_n variables is set true, a should be true as well, if a y_n variable is true a should be set to false. This also prevents non-satisfying assignments from becoming solutions. As a result, we can safely preserve the entire group G. ◀

▶ **Proposition 24.** *Symmetry-aware BVA transformations are reconstructing, solution preserving, backwards symmetry invariant and forwards symmetry invariant. Naive BVA transformations are reconstructing, solution preserving and backwards symmetry invariant, but not necessarily forwards symmetry invariant.*

Proof of Proposition 24. This proof is similar to the proofs for Proposition 18 and Proposition 20, using Lemma 23 instead of Lemma 19. ◀

▶ **Lemma 26.** *When using the symmetry-aware sum decomposition, all symmetries of the original problem are projected symmetries of the resulting problem.*

Proof of Lemma 26. We need to show that each permutation $\pi \in G$, is also a $\mathcal{V}$-symmetry of $\mathcal{P}'$, where $\mathcal{P}'$ is the result of applying the prefix-sum decomposition to $\mathcal{P}$. This holds if for every $\mathcal{V}$-assignment α' of $\mathcal{P}'$, α' can be extended to a solution of $\mathcal{P}'$ if and only if $\pi(\alpha')$ can be extended to a solution of $\mathcal{P}'$.

This is the case, since each satisfying assignment of the original problem can be extended to a satisfying assignment of the problem after decomposition, by assigning the appropriate values to the sum variables s_k. Concretely, each s_k should equal its partial sum. As a result, we can safely preserve the entire group G. ◀

▶ **Proposition 27.** *The symmetry-aware prefix-sum decomposition is perfect, while the naive prefix-sum decomposition is reconstructing, solution preserving and backwards symmetry invariant, but not necessarily forwards symmetry invariant.*

Proof of Proposition 27. This proof follows the same structure as the proofs of Proposition 18, Proposition 20, and Proposition 24, with Lemma 26 replacing the lemmas used therein. ◀