Control Flow-Based Symmetry Reduction for Parameterised Boolean Equation Systems*

Menno Bartels ✉ [0009−0007−1400−5100], Maurice Laveaux[0000−0001−8732−7580], Thomas Neele[0000−0001−6117−9129], and Tim A.C. Willemse[0000−0003−3049−7962]

Eindhoven University of Technology, Eindhoven, Netherlands
{m.m.g.bartels, m.laveaux, t.s.neele, t.a.c.willemse}@tue.nl

Abstract. Symmetry reduction is a technique that combats the state space explosion problem for model checking. The main idea of symmetry reduction is to merge symmetric states. Two states are symmetric if they can be mapped onto each other such that their semantics is preserved. To carry out symmetry reduction, one first has to identify symmetries for the system at hand. In this work we introduce the concept of symmetries for systems of fixed point equations called parameterised Boolean equation systems. We propose a technique that extracts a symmetry group from the control flow found in such an equation system. We implemented a prototype tool with which we highlight that our technique can effectively find symmetries and in turn achieve substantial reductions.

Keywords: Model checking · Parameterised Boolean equation system · Symmetry reduction · Control flow · Parity games

1 Introduction

Within the area of model checking, symmetry reduction is a technique that aims to combat the well-known state space explosion problem. To reduce the effort spent on answering the model checking problem, symmetry reduction merges symmetric states of a system. Assume that states are represented as vectors of values. Then two states are symmetric, if there exists a mapping that rearranges the state vectors into one another. During solving, a reduced state space is generated that takes into account the symmetries between states. This reduced state space is sufficient to answer the model checking problem fully. In turn, the model checking problem can be more efficiently answered.

Model checking tools like CADP [9] and mCRL2 [2] use *parameterised Boolean equation systems* (PBESs) [10] to encode the μ-calculus model checking problem. PBESs are systems of fixpoint equations ranging over predicate formulas. In mCRL2, a PBES is constructed from a system specification and a μ-calculus formula to be verified. *Solving* a PBES and obtaining the answer to the model checking problem is achieved by converting it to a parity game [14]. This conversion resembles state space exploration. Then, the solving of the generated parity

* Artefact available at https://doi.org/10.5281/zenodo.19333256

game is done using standard techniques, like the recursive algorithm [30]. Unfortunately, this approach suffers from the state space explosion. The underlying parity game tends to grow exponentially with the number of parallel, possibly identical, components in the system which makes it infeasible to generate it.

When a model contains parallel processes that are instantiated using the same process definition, the associated PBES typically contains a symmetric structure. However, the notion of concurrent processes is lost when translating the model to a PBES. This makes it challenging to identify these symmetries in a PBES. Checking all possible permutations of state vectors is intractable, and the state space itself cannot be used in the process: we need to compute symmetries *a priori* in order to truly avoid the state space explosion. To address this, we develop the theory for symmetry reduction for PBESs. An advantage of this approach is that both the system model and the property to verify are automatically taken into account, since they are together captured in the PBES, possibly yielding better reductions.

We make the following contributions. (1) We define the notion of symmetry in the context of PBESs. This involves applying permutations to predicate formulas. We show that symmetries preserve the solution of a PBES (Theorem 1). Building on [17], this also holds in the context of the reduced parity game (Theorem 2). (2) We present a technique that extracts symmetries from a PBES using control flow. We constructively build up a set of symmetries, such that at every stage in the construction the control flow information is taken into account. The resulting *syntactic symmetries* qualify for reducing the parity game (Theorem 3). (3) We implemented a prototype tool with which we highlight that our technique can effectively find symmetries. Our results showcase the substantial reductions that can be achieved in terms of state space size and solving times.

Related work Symmetry reduction for model checking has been studied extensively [25]. Although symmetry reduction does not give guarantees on effectiveness, it has been applied successfully in practice [12,16,11,29,28]. One approach to achieve symmetry reduction is to extend the data structures at hand with functionality to encode symmetry [26,12]. This encoding then enables pruning the generation of the state space, if the property that is being checked allows for this. Alternatively, bottom up approaches have been developed [5,4]. These first consider the semantics of the property and then generate a tailored quotient structure. This structure is then used to answer the model checking problem fully, with respect to the provided formula.

The notion of control flow has been studied in the context of linear processes in [23] and on PBESs in [15]. It provides a way to identify parameters that can be used as program counters of the internal processes of a system. In [23,15] this information is used to perform a liveness analysis. There, a reduction is achieved by discarding parameters that are identified as not alive. Detection of control flow can be done statically by inspecting the syntax of the model. This means that control flow can be identified before the underlying state space is generated.

Previously, several other approaches have been developed for reducing PBESs or parity games. Some of these operate purely on syntax, such as elimination

of constant or non-influential parameters [21] or quantifiers [18]. Other approaches, like symbolic bisimulation quotienting [19] and partial-order reduction (POR) [20], operate on semantics (i.e., parity games) and are therefore more akin to our approach. In particular, both POR and symmetry reduction involve first performing static analysis on the input PBES before reducing the generated parity game *on-the-fly*.

Outline We work out preliminary definitions in Section 2. In Section 3 we discuss permutations and symmetries. We introduce control flow in Section 4.1. Section 4.2 explains our proposed technique to extract symmetries. We give brief implementation details and show the results of our experiments in Section 5.

2 Preliminaries

We use abstract data types, denoted $D, E, \ldots$, each having a semantic domain, written $\mathbb{D}, \mathbb{E}, \ldots$. For Booleans we use B and for natural numbers we use N, with $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$ and $\mathbb{N}$ as their semantic counterparts. We fix a set of variables $\mathcal{D}$. To denote that a variable $d \in \mathcal{D}$ is of type D we write $d : D$. If v is a value in $\mathbb{D}$ we write $v \in \mathbb{D}$. We use bold font to denote vectors. With $\boldsymbol{d}$ we denote a vector of variables, with $\boldsymbol{D}$ a vector of types, with $\boldsymbol{v}$ a vector of values and with $\mathbb{\boldsymbol{D}}$ a vector of semantic domains. For a vector $\boldsymbol{x}$, we write x_k for its k'th parameter.

Definition 1. *A* predicate formula *is a first-order formula consisting of* terms *and Boolean connectives, generated by the following grammar.*

$$\varphi := b \mid \varphi \oplus \varphi \mid \mathcal{Q}e : D \,.\, \varphi \mid X(\boldsymbol{t}) \qquad\qquad b, t := x \mid f \mid t(t, \ldots, t)$$

Here b is a *term* of type B, X is a predicate variable from a predicate variable set $\mathcal{X}$ and $\boldsymbol{t}$ is a vector of terms t, each of type D. Moreover, x is a parameter of type D and f is a function of type $D^n \to D$ for $n \in \mathbb{N}$. Lastly $\mathcal{Q} \in \{\forall, \exists\}$ and $\oplus \in \{\vee, \wedge\}$. The set of all predicate formulas is denoted with Φ. The set of all terms is written as T, and the set of vectors of terms is $\boldsymbol{T}$. Without loss of generality, we consider any non-Boolean term to have arbitrary type D.

Definition 2. *Let $d, d' \in \mathcal{D}$ be variables.* Syntactic substitution *on a predicate formula φ, written $\varphi[d := d']$, is defined as follows.*

$$\begin{aligned}
x[d := d'] &= \begin{cases} d' & \text{if } x = d \\ x & \text{otherwise} \end{cases} \\
f[d := d'] &= f \\
t(t, \ldots, t)[d := d'] &= t[d := d'](t[d := d'], \ldots, t[d := d']) \\
(\varphi_1 \oplus \varphi_2)[d := d'] &= \varphi_1[d := d'] \oplus \varphi_2[d := d'] \\
(\mathcal{Q}e : D.\varphi)[d := d'] &= \mathcal{Q}e : D.(\varphi[d := d']) \\
X(\boldsymbol{t})[d := d'] &= X(\boldsymbol{t}[d := d'])
\end{aligned}$$

For a vector of terms we have $(t_1, \ldots, t_n)[d := d'] = (t_1[d := d'], \ldots, t_n[d := d'])$. When φ contains a quantifier that binds e, we require that $d \neq e$ and $d' \neq e$. The

semantics of a predicate formula φ is defined in the context of a *data environment* $\delta : \mathcal{D} \to \mathbb{D}$ and a *predicate environment* $\eta : \mathcal{X} \to 2^{\mathbb{D}}$. We write $[\![\varphi]\!]\eta\delta$ where:

$$\begin{aligned}
[\![b]\!]\eta\delta &= [\![b]\!]\delta \\
[\![\varphi \oplus \psi]\!]\eta\delta &= [\![\varphi]\!]\eta\delta \text{ and/or } [\![\psi]\!]\eta\delta \\
[\![\mathcal{Q}e : D \,.\, \varphi]\!]\eta\delta &= \text{for all/some } v \in \mathbb{D}, [\![\psi]\!]\eta\delta[v/e] \\
[\![X(\boldsymbol{t})]\!]\eta\delta &= [\![\boldsymbol{t}]\!]\delta \in \eta(X)
\end{aligned}$$

For terms, the semantics is defined in the context of a data environment δ.

$$[\![x]\!]\delta = \delta(x) \qquad [\![f]\!]\delta = [\![f]\!] \qquad [\![t(t,\ldots,t)]\!]\delta = [\![t]\!]\delta([\![t]\!]\delta,\ldots,[\![t]\!]\delta)$$

We assume any function symbol f to have a defined semantics that does not depend on any environment. For a vector of terms, we have $[\![\boldsymbol{t}]\!]\delta = ([\![t_1]\!]\delta,\ldots,[\![t_n]\!]\delta)$. To update an environment we write $\delta[v/d]$ where $\delta[v/d](x) = v$ if $x = d$, else $\delta[v/d](x) = \delta(x)$. For multiple updates we write $\delta[v/d][w/e]$ as $\delta[v/d, w/e]$. For vectors of equal length, $\delta[\boldsymbol{v}/\boldsymbol{d}]$ denotes the update of every d_k with v_k.

A *parameterised Boolean equation system* (PBES) [10] is a sequence of fixpoint equations of the form $\sigma X(\boldsymbol{d} : \boldsymbol{D}) = \varphi_X$. Here $X \in \mathcal{X}$ is a predicate variable, φ_X is a predicate formula and σ is either the least fixpoint operator μ or the greatest fixpoint operator ν. To denote an arbitrary PBES we write $\mathcal{E}$. For an equation $(\sigma X(\boldsymbol{d}\colon \boldsymbol{D}) = \varphi_X) \in \mathcal{E}$ we call the predicate formula φ_X the *right-hand side* of bound predicate variable X. We write $\mathsf{bnd}(\mathcal{E})$ for the set of all predicate variables bound by a fixed point operator in $\mathcal{E}$. We consider PBESs that are *closed* and *well-formed*, i.e., no free variables occur and every predicate variable occurring is bound by exactly one equation. Every equation in $\mathcal{E}$ is assigned a *rank*. This is a natural number such that $\mathsf{rank}_{\mathcal{E}}(X) \leq \mathsf{rank}_{\mathcal{E}}(Y)$ if the equation for X occurs before the equation for Y in $\mathcal{E}$. Moreover, $\mathsf{rank}_{\mathcal{E}}(X)$ is an even number if and only if the equation for X has a greatest fixed point. For some bound predicate variable $\hat{X}$, each PBES has an *initial valuation* $\hat{X}(\hat{\boldsymbol{v}})$, where for parameter d_k in $\boldsymbol{d}$ the initial value is denoted with $\mathsf{init}(d_k) = \hat{v}_k$.

We examine PBESs in *standard recursive form* (SRF) [19]. This normal form requires that all right-hand side formulas φ_X occurring in the PBES are either disjunctive or conjunctive, meaning they are in of one of the two following shapes:

$$\bigvee_{j \in J_X} \exists \boldsymbol{e}_j : \boldsymbol{E}_j .\, f_j(\boldsymbol{d}, \boldsymbol{e}_j) \wedge X_j(\boldsymbol{g}_j(\boldsymbol{d}, \boldsymbol{e}_j)) \text{ or } \bigwedge_{j \in J_X} \forall \boldsymbol{e}_j : \boldsymbol{E}_j .\, f_j(\boldsymbol{d}, \boldsymbol{e}_j) \vee X_j(\boldsymbol{g}_j(\boldsymbol{d}, \boldsymbol{e}_j))$$

The disjuncts and conjuncts occurring in the formulas are called *clauses*. In the equation for a predicate variable X the clauses range over a finite index set $J_X \subset \mathbb{N}$. Any $f_j(\boldsymbol{d}, \boldsymbol{e}_j)$ is a term of sort B and any $\boldsymbol{g}_j(\boldsymbol{d}, \boldsymbol{e}_j)$ is a vector of terms, each of sort D. We write $g_{j,k}(\boldsymbol{d}, \boldsymbol{e}_j)$ for the k'th entry of $\boldsymbol{g}_j(\boldsymbol{d}, \boldsymbol{e}_j)$. For any PBES-SRF $\mathcal{E}$ we define the mapping $\mathsf{op}_{\mathcal{E}}(X)\colon \mathsf{bnd}(\mathcal{E}) \to \{\wedge, \vee\}$ which returns whether the right-hand side a predicate variable is disjunctive or conjunctive.

Example 1. Consider the following PBES-SRF $\mathcal{E}$. Here Z denotes integers.

$$\begin{aligned}\mu X(c_1{:}B, d_1{:}Z, c_2{:}B, d_2{:}Z) = {} & (\neg c_1 \vee Y(c_1, 1, c_2, d_2)) \wedge (\neg c_2 \vee Y(c_1, d_1, c_2, 1))\\ \nu Y(c_1{:}B, d_1{:}Z, c_2{:}B, d_2{:}Z) = {} & (c_1 \wedge Y(\neg c_1, 2, c_2, d_2)) \vee (c_2 \wedge Y(c_1, d_2, \neg c_2, 2))\\ & \vee (\neg c_1 \wedge \neg c_2 \wedge (d_1 + d_2 \geq 4)\\ & \quad \wedge X(\neg c_1, d_1 - d_2, \neg c_2, d_2 - d_1))\end{aligned}$$

We assume that every equation has exactly the same vector of data parameters $\boldsymbol{d}$ of type $\boldsymbol{D}$, where the length of $\boldsymbol{d}$ is set to $n \in \mathbb{N}$. Any parameter that does not occur in an equation can be added and copied in all predicate variables occurring throughout the PBES. This means that this assumption does not impact generality. Additionally, because disjunction and conjunction are idempotent, we may also safely assume that within one equation there exists only one copy of each clause. The latter assumption is essential for the existence of bijections between clauses. This will become relevant in Section 4.2.

The standard semantics for PBESs is the denotational fixed point semantics [10]. We forego using this and instead consider the *parity game semantics* of PBESs. This game semantics provides an operational view on PBESs and is provably equivalent to the denotational semantics [6]. From the outcome of the parity game one can conclude whether a value is a solution to the corresponding PBES-SRF. A *parity game* is a two player game played on a directed graph. The two players are called *even* ($\diamond$) and *odd* ($\Box$). The objective of the two players is to *win* as many vertices as possible.

Definition 3. *A* parity game *is a directed graph $M = (V, E, \Omega, \mathcal{P})$, where V is the set of vertices, E the edge relation, $\Omega\colon V \to \mathbb{N}$ the priority function and $\mathcal{P}\colon V \to \{\diamond, \Box\}$ the ownership function.*

The rules of a parity game are as follows. Each vertex in the game has a priority and an owner, as defined by Ω and $\mathcal{P}$, respectively. Initially a token is placed on some vertex in the graph. The player that owns the vertex on which the token is placed moves the token over an outgoing edge to a next vertex. This process continues indefinitely or until the token is on a vertex without any outgoing edges in which case the token is stuck. During the game, the token moves along a possibly infinite path called a *play*. If a play is finite, it is won by the player that does *not* own the vertex on which the token is stuck. If a play is infinite, then it is won by player $\diamond$ if the minimal priority that occurs infinitely often along the play is even. Else it is won by player $\Box$.

To denote an arbitrary player, we write p. A *strategy* is partial function $\sigma_p\colon V \to V$ that prescribes how p moves the token to a successive vertex in the game. For vertices owned by p, a strategy σ_p may be undefined, for vertices not owned by p, it is always undefined. If $\sigma_p(v)$ is defined, then it must be the case that $(v, \sigma_p(v)) \in E$. A play $v_1 v_2 \ldots$ is consistent with a strategy σ_p if for any v_i that is owned by p and for which $\sigma_p(v_i)$ is defined it holds that $\sigma_p(v_i) = v_{i+1}$. The player p wins a vertex v if and only if there exists a strategy σ_p such that all plays starting in v that are consistent with σ_p are won by p.

We now define the solution to a PBES-SRF using the *associated parity game*.

Definition 4. *The* solution *of $\mathcal{E}$ is the mapping $[\![\mathcal{E}]\!]\colon \mathsf{bnd}(\mathcal{E}) \to 2^{\mathbb{D}}$ where*

$$[\![\mathcal{E}]\!](X) = \{\boldsymbol{v} \in \mathbb{D} \mid X(\boldsymbol{v}) \text{ won by } \Diamond \text{ in } M_{\mathcal{E}}\}$$

Here $M_{\mathcal{E}} = (V, E, \Omega, \mathcal{P})$ is the *associated parity game* for $\mathcal{E}$. We have that $V = \{X(\boldsymbol{v}) \mid X \in \mathsf{bnd}(\mathcal{E}), \boldsymbol{v} \in \mathbb{D}^n\}$ and $(X(\boldsymbol{v}), Y(\boldsymbol{w})) \in E$ if there exists $j \in J_X$ and $\boldsymbol{v}_j \in \mathbb{E}_j$ such that $X_j = Y$, $[\![f_j(\boldsymbol{d}, \boldsymbol{e}_j)]\!]\delta[\boldsymbol{v}/\boldsymbol{d}, \boldsymbol{v}_j/\boldsymbol{e}_j]$ holds and $\boldsymbol{w} = [\![g_j(\boldsymbol{d}, \boldsymbol{e}_j)]\!]\delta[\boldsymbol{v}/\boldsymbol{d}, \boldsymbol{v}_j/\boldsymbol{e}_j]$. Moreover, $\Omega(X(\boldsymbol{v})) = \mathsf{rank}_{\mathcal{E}}(X)$ and $\mathcal{P}(X(\boldsymbol{v})) = \Box$ iff $\mathsf{op}_{\mathcal{E}}(X) = \wedge$ else $\mathcal{P}(X(\boldsymbol{v})) = \Diamond$.

We recall that since the associated parity game is the semantics underlying a PBES-SRF, deciding the winner of a vertex $X(\boldsymbol{v})$ is equivalent to checking whether $\boldsymbol{v}$ is an element of the solution to the equation for X in $\mathcal{E}$ [19]. In turn, this means that the model checking problem that is encoded in the PBES-SRF can be solved by determining the winner of the initial valuation $\hat{X}(\hat{\boldsymbol{v}})$ in the associated parity game.

Example 2. The associated parity game for Example 1 is depicted in Figure 1. We use t and f as shorthand for the semantic values true and false.

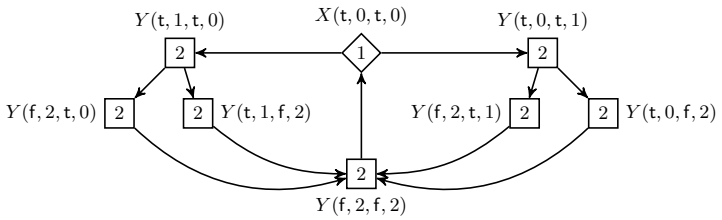


Fig. 1. The associated parity game $M_{\mathcal{E}}$ for for Example 1

3 Symmetries

A symmetry is a mapping of an object onto itself such that the semantics of the object is preserved. The mappings we use are *permutations*. We define how to apply permutations to predicate formulas and show that they form a *group*. A detailed treatment of group theory can be found in [24].

A *permutation* on a set S is a bijection $\pi\colon S \to S$. We define the $\mathsf{per}(S)$ as the set of all permutations on S. Together with function composition, $\mathsf{per}(S)$ is a *group*. This means that for $\alpha, \beta \in \mathsf{per}(S)$, $\alpha \circ \beta$ is again a permutation on S. Also for every permutation π there exists a unique inverse, denoted π^{-1}. We define how to apply group elements to other objects by means of a *group action*.

Definition 5. *Let $(G, \circ)$ be a group and S be a set. A* group action *is a function $\mathsf{act}\colon G \times S \to S$ such that for any $\alpha, \beta \in G$ and $x \in S$ we have:*

$$\mathsf{act}(\mathrm{id}, x) = x \qquad \mathsf{act}(\alpha, \mathsf{act}(\beta, x)) = \mathsf{act}(\alpha \circ \beta, x)$$

If a group action exists we say that the group G acts on the set S. As shorthand, we write $\alpha(x)$ to refer to the action $\mathsf{act}(\alpha, x)$.

Example 3. We write permutations in disjoint cycle notation. Let $S = \{1, 2, 3\}$. Then $\alpha = (123)$ is defined as $\alpha(1) = 2, \alpha(2) = 3$ and $\alpha(3) = 1$. It maps 1 and 2 to the element on their right, and 3 is mapped back to the first. Likewise for $\beta = (12)(3) = (12)$, $\beta(1) = 2, \beta(2) = 1$ and $\beta(3) = 3$. We omit identity cycles.

Let $\mathsf{var} = \{d_1, \ldots, d_n\}$ be the set of free variables that may occur in predicate formulas and let $[n] := \{1, \ldots, n\}$. Given a term t and a permutation $\pi \in \mathsf{per}([n])$ we define $t[\pi] = t[d_1 := d_{\pi(1)}, d_2 := d_{\pi(2)}, \ldots, d_n := d_{\pi(n)}]$. Here each free variable d_i occurring in t is syntactically substituted by $\pi(d_i)$, simultaneously. Sometimes we write $t[d_i := d_{\pi(i)}]_{i \in [n]}$ to denote $t[\pi]$. Using this we define how permutation act on predicate formulas. We write $\pi(\varphi)$ and define:

$$\begin{aligned} \pi(b) &= b[\pi] \\ \pi(\varphi_1 \oplus \varphi_2) &= \pi(\varphi_1) \oplus \pi(\varphi_2) \\ \pi(\mathcal{Q}e : D.\varphi) &= \mathcal{Q}e : D.\, \pi(\varphi) \\ \pi(X(t_1, \ldots, t_n)) &= X(t_{\pi^{-1}(1)}[\pi], \ldots, t_{\pi^{-1}(n)}[\pi]) \end{aligned}$$

This definition yields a group action on the set of predicate formulas Φ, i.e. the set of predicate formulas itself is closed under inverses and composition of permutations. Note that for a vector of terms, a permutation affects only the *positions* of the entries and not the *indices*. This is important for proving compositionality of the group action. For a vector of values $\boldsymbol{v} = (v_1, \ldots, v_n)$, the application of a permutation is $\pi((v_1, \ldots, v_n)) = (v_{\pi^{-1}(1)}, \ldots, v_{\pi^{-1}(n)})$. For a data environment a permutation remaps the image of every parameter in the domain of the environment, to the image of the permuted parameter. For a predicate environment applying a permutation substitutes for every vector of values, its permuted counterpart. We give the definition of applying a permutation to an environment.

$$\pi(\delta) = \delta[\delta(d_{\pi(i)})/d_i]_{i \in [n]} \qquad \pi(\eta) = \eta[\{\pi^{-1}(\boldsymbol{v}) \mid \boldsymbol{v} \in \eta(X)\}/X]_{X \in \mathcal{X}}$$

We point out that interpreting a permuted predicate formula in some environments coincides with interpreting the original formula in permuted environments.

Lemma 1. *Let $\pi \in \mathsf{per}([n])$ then for any $\varphi \in \Phi$ and environments η and δ:*

$$[\![\pi(\varphi)]\!]\eta\delta = [\![\varphi]\!]\pi(\eta)\pi(\delta)$$

Proof. This can be proven by structural induction on predicate formula φ.

A permutation is a *semantic symmetry* for $\mathcal{E}$ if for every equation the solution of the bound predicate variable is preserved after applying the permutation. Moreover, a permutation is a *syntactic symmetry* if it leaves intact the syntactic structure of the PBES-SRF. This in turn also preserves its semantics. First we give the definition of a semantic symmetry.

Definition 6. *A permutation* $\pi \in \mathsf{per}([n])$ *is a* semantic symmetry *for* $\mathcal{E}$ *if for for every* $(\sigma X(\boldsymbol{d}\colon \boldsymbol{D}) = \varphi_X) \in \mathcal{E}$ *we have that* $[\![\pi(\varphi_X)]\!]\eta\delta = [\![\varphi_X]\!]\eta\delta$.

When checking whether a value is in the solution of a PBES-SRF, we can use the result of its symmetric counterparts. For each set of symmetric values, we only inspect whether one of them is in the solution.

Theorem 1. *Let* π *be a symmetry for* $\mathcal{E}$. *Then for* $X \in \mathsf{bnd}(X)$ *and* $\boldsymbol{v} \in \mathbb{D}$ *it holds that* $X(\boldsymbol{v})$ *is won by* $\diamond$ *in* $M_\mathcal{E}$ *if and only if* $X(\pi(\boldsymbol{v}))$*is won by* $\diamond$ *in* $M_\mathcal{E}$.

We fix a group of symmetries G. The *orbit* of a vector of values $\boldsymbol{v} \in \mathbb{D}$ is defined as $\theta(\boldsymbol{v}) = \{\boldsymbol{v}' \in \mathbb{D} \mid \exists \pi \in G\ .\ \pi(\boldsymbol{v}) = \boldsymbol{v}'\}$. Vectors of values that are in the same orbit are called *symmetric*. We use orbits to define *quotient parity games* [17]. To check if $\boldsymbol{v}$ is in the solution of X in $\mathcal{E}$, it suffices to solve the quotient parity game associated with $\mathcal{E}$. The quotient game typically has fewer vertices, as the construction involves merging vertices that contain symmetric vectors of values.

Definition 7. *Consider the associated parity game* $M_\mathcal{E}$. *Let* G *be a symmetry group. The* quotient parity game *with respect to* G *is defined as* $M_\mathcal{E}^G = (V^G, E^G, \Omega^G, \mathcal{P}^G)$, *where we have vertex set* $V^G = \{X(\theta(\boldsymbol{v})) \mid \boldsymbol{v} \in \mathbb{D}\}$, *edge relation* $E^G = \{(X(\theta(\boldsymbol{v})), Y(\theta(\boldsymbol{w}))) \mid (X(\boldsymbol{v}), Y(\boldsymbol{w})) \in E\}$, *priority function* $\Omega^G(X(\theta(\boldsymbol{v}))) = \Omega(X(\boldsymbol{v}))$ *and ownership function* $\mathcal{P}^G(X(\theta(\boldsymbol{v}))) = \mathcal{P}(X(\boldsymbol{v}))$.

Example 4. Consider the symmetry group $G = \{\mathrm{id}, (13)(24)\}$. The quotient parity game $M_\mathcal{E}^G$ for Example 2, is depicted in Figure 2. For every orbit only a representative vertex is shown. For example, when solving for $Y(f, 2, t, 1)$ we also get the result for the vertex $Y(t, 1, f, 2)$, as these vertices are symmetric in $M_\mathcal{E}$.

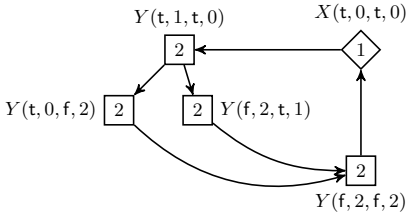


Fig. 2. The quotient parity game $M_\mathcal{E}^G$ for Example 4

In our framework, generating the quotient game corresponds to carrying out symmetry reduction. Instead of generating the whole parity game, we take into account the symmetric structure of the PBES-SRF to generate states that represent orbits. This speeds up solving the model checking problem since the reduction often leaves us with fewer states. In fact, the effort of solving is dependent on the size of the parity game underlying the PBES-SRF.

Theorem 2. *Let* G *be a symmetry group. Then for* $X \in \mathsf{bnd}(\mathcal{E})$ *and* $v \in \mathbb{D}$ *it holds that* $X(\boldsymbol{v})$ *is won by* $\diamond$ *in* $M_\mathcal{E}$ *if and only if* $X(\theta(\boldsymbol{v}))$ *is won by* $\diamond$ *in* $M_\mathcal{E}^G$.

4 Extracting Symmetries from Control Flow Graphs

To extract symmetries from a system we consider *syntactic symmetries*. This kind of symmetry does not involve a semantic check. It is based on a purely syntactical equivalence. This choice serves as a practical heuristic – searching for semantics symmetries involves a check for semantic equivalence, which is undecidable in general [27,3]. Nevertheless, the theory presented in Section 3 can still be applied, as any syntactic symmetry is a semantic symmetry.

Definition 8. *A permutation* $\pi \in \mathsf{per}([n])$ *is a* syntactic symmetry *for PBES-SRF* $\mathcal{E}$ *if for every* $(\sigma X(\boldsymbol{d}\colon \boldsymbol{D}) = \varphi_X) \in \mathcal{E}$ *and for all* $j \in J_X$ *there exists* $j' \in J_X$ *such that* $X_j = X_{j'}$, $\pi(f_j(\boldsymbol{d}, \boldsymbol{e}_j)) = f_{j'}(\boldsymbol{d}, \boldsymbol{e}_{j'})$ *and* $\pi(\boldsymbol{g}_j(\boldsymbol{d}, \boldsymbol{e}_j)) = \boldsymbol{g}_{j'}(\boldsymbol{d}, \boldsymbol{e}_{j'})$.

Here J_X is the index set for φ_X. Then $f_j(\boldsymbol{d}, \boldsymbol{e}_j)$ is the Boolean term that occurs in clause $j \in J_X$. Likewise $\boldsymbol{g}_j(\boldsymbol{d}, \boldsymbol{e}_j)$ is the vector of terms that occurs in j.

Syntactic symmetries act on predicate formulas and preserve the set of clauses that occur in right-hand side formulas of a PBES-SRF. Hence, any syntactic symmetry is indeed a semantic symmetry. From hereon out we consider only syntactic symmetries and refer to them simply as a *symmetry*.

4.1 Control Flow Graphs

We introduce control flow parameters (CFP) and controls flow graphs (CFG) [23,15] for PBES-SRFs. The search for symmetries exploits control flow to select only permutations that preserve the CFG structure. The control flow of a PBES-SRF is defined using the values of parameters before and after passing through a given clause. To make this explicit we define *source* and *target* functions.

Definition 9. *Consider a partial function* $h\colon \mathsf{bnd}(\mathcal{E}) \times \mathbb{N} \times \mathbb{N} \to \mathbb{D}$. *If for all* $X \in \mathsf{bnd}(\mathcal{E})$, $j \in J_X$, $k \leq n$ *and* δ, *we have that* $h(X, j, k) = v$, *then* h *is a:*

$$\begin{array}{ll} \text{source } \textit{function if:} & [\![f_j(\boldsymbol{d}, \boldsymbol{e}_j)]\!]\delta \implies [\![d_k]\!]\delta = v \\ \text{target } \textit{function if:} & [\![f_j(\boldsymbol{d}, \boldsymbol{e}_j)]\!]\delta \implies [\![g_{j,k}(\boldsymbol{d}, \boldsymbol{e}_j)]\!]\delta = v \end{array}$$

A *source* (resp. *target*) function returns the unique value, if it exists, of parameter d_k right before (resp. after) passing through predicate X_j in φ_X. The definitions allow for under-approximating source and target functions. In practice, we exploit this by performing a purely syntactical analysis on f_j and $\boldsymbol{g}_j$, respectively. From here on, for any given PBES-SRF $\mathcal{E}$ we fix functions src and tgt.

In a PBES-SRF $\mathcal{E}$, a parameter *rules* a clause if both src and tgt are defined for this parameter. The set of all clauses ruled by d_k in φ_X is written $\mathsf{rules}(d_k, X) = \{j \in J_X \mid d_k \text{ rules } j\}$. A clause j may also *copy* a parameter d_k. This means that value of the parameter gets copied over in the recursive call to the predicate variable within the clause, i.e. $[\![g_{j,k}(\boldsymbol{d}, \boldsymbol{e}_j)]\!]\delta = [\![d_k]\!]\delta$ for all δ.

A *control flow parameter* (CFP) is a parameter for which at every point in the system we can deduce its value.

Definition 10. *A parameter d_k is a* control flow parameter *if for all $X \in \mathsf{bnd}(\mathcal{E})$ and $j \in J_X$ we have that either d_k rules j or d_k is copied by j.*

With $\mathsf{cfp}(\mathcal{E})$ we denote the set of CFPs of a PBES-SRF $\mathcal{E}$. A parameter that is not a CFP is called a *data parameter* (DP). For the set of DPs that occur in $\mathcal{E}$ we write $\mathsf{dp}(\mathcal{E})$. The set of semantic values that a CFP c_k can attain is the set $\mathsf{values}(c_k)$ that is defined as

$$\{\mathsf{init}(c_k)\} \cup \bigcup_{\substack{X \in \mathsf{bnd}(\mathcal{E}),\\ j \in \mathsf{rules}(c_k, X)}} \{v \in D \mid v = \mathsf{src}(X, j, k) \vee v = \mathsf{tgt}(X, j, k)\}$$

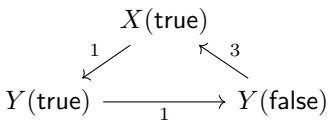
We define *control flow graphs* (CFG). These make explicit the information that we can statically deduce for the CFPs. In a CFG the vertices consist of predicate variable and the values a CFP attains. The edges depict the clauses in the PBES-SRF that are responsible for a change in value for a CFP.

Definition 11. *Let $\mathcal{E}$ be a PBES-SRF. The control flow graph (CFG) of a control flow parameter $c_k \in \mathsf{cfp}(\mathcal{E})$ is a directed graph $\mathsf{G}_k = (\mathsf{V}_k, \mathsf{E}_k)$ where*

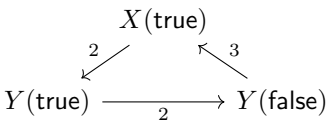
$$\begin{aligned}\mathsf{V}_k &= \{X(v) \mid X \in \mathsf{bnd}(\mathcal{E}), v \in \mathsf{values}(c_k)\}\\ \mathsf{E}_k &= \{(X(v), j, Y(w)) \mid j \in \mathsf{rules}(c_k, X) \wedge \mathsf{src}(X, j, k) = v \wedge\\ &\qquad\qquad \mathsf{tgt}(X, j, k) = w \wedge Y = X_j\}\end{aligned}$$

Example 5. We reconsider Example 1. First we remark that c_1 and c_2 are the control flow parameters of $\mathcal{E}$. These have G_{c_1} and G_{c_2} as their control flow graphs. The labeling on the edges denotes the index of the clause in the right-hand side formula that is responsible for changing the value of the control flow parameter. For example, in the first clause of φ_X the predicate variable Y and the value of c_1 is copied. Hence the edge $(X(\mathsf{true}), 1, Y(\mathsf{true}))$ is in G_{c_1}.

G_{c_1}:



G_{c_2}:



We point out how data parameters are related to clauses in a PBES-SRF. To this end we define when a data parameter is *used for*, *used in* and *changed by* a clause in a PBES-SRF. This allows us to compare CFGs in terms of the data parameters that are interacted with. We introduce $\mathsf{fv}(\varphi) \subseteq \mathcal{D}$ as the set of data parameters freely occurring in formula φ.

Definition 12. *Consider the PBES-SRF $\mathcal{E}$ and an equation $(\sigma X(\boldsymbol{d}\colon \boldsymbol{D}) = \varphi_X)$. Consider a clause j for some index $j \in J_X$. We say that a data parameter d_k is:*

- used for *clause j if $d_k \in \mathsf{fv}(f_j)$;*
- used in *clause j if there exists some $l \leq n$ such that $d_k \in \mathsf{fv}(g_{j,l}(\boldsymbol{d}, \boldsymbol{e}_j))$ where if $X = X_j$ then $k \neq l$;*
- changed by *clause j if $X = X_j$ and $d_k \neq g_{j,k}(\boldsymbol{d}, \boldsymbol{e}_j)$.*

We denote the data parameters that are used for, used in and changed by $j \in J_X$ as $\mathsf{uf}(X, j)$, $\mathsf{ui}(X, j)$ and $\mathsf{cb}(X, j)$, respectively. For an edge $e = (X(\boldsymbol{v}), j, Y(\boldsymbol{w}))$ we write $\mathsf{uf}(e)$ for $\mathsf{uf}(X, j)$, likewise we write $\mathsf{ui}(e)$ and $\mathsf{cb}(e)$.

Example 6. Recall Example 1. Here $\mathsf{cb}(X, 1) = \mathsf{cb}(Y, 1) = \{d_1\}$, $\mathsf{cb}(X, 2) = \mathsf{cb}(Y, 2) = \{d_2\}$ and $\mathsf{uf}(Y, 3) = \mathsf{ui}(Y, 3) = \mathsf{cb}(Y, 3) = \{d_1, d_2\}$.

4.2 Extracting Symmetries from CFGs

In this section we present our technique that extracts symmetries for a PBES-SRF. This technique can be summarized as follows. First the CFPs are identified and then partitioned based on the size and shape of their CFGs. Next, we generate a set of permutations that leaves intact this partition. This set is then refined by discarding the permutations that do not induce an isomorphism on the CFGs of the PBES-SRF. For those that are left, we check which are an actual symmetry by comparing against Definition 8. What remains is a set of permutations that is a symmetry group for the PBES-SRF at hand.

We define *compatibility* as a means to compare the CFPs. Two CFPs are compatible if their CFGs have equal vertex sets, the sizes of the data sets are equal and the number of edges between vertices is equal.

Definition 13. *Let c and c' be two CFPs. Then* $\mathsf{compatible}$ *is defined as*

$$\begin{aligned}\mathsf{compatible}(c, c') := (\mathsf{V}_c = \mathsf{V}_{c'}) \wedge \forall s, s' \in \mathsf{V}_c \cup \mathsf{V}_{c'}.(\mathsf{sizes}(c, s, s') = \mathsf{sizes}(c', s, s') \\ \wedge\ |\{j \mid (s, j, s') \in \mathsf{E}_c\}| = |\{j \mid (s, j, s') \in \mathsf{E}_{c'}\}|)\end{aligned}$$

where $\mathsf{sizes}(c, s, s') = \{(|\mathsf{uf}(e)|, |\mathsf{ui}(e)|, |\mathsf{cb}(e)|) \mid e = (s, j, s') \in \mathsf{E}_c\}$

In general there can be multiple edges between two vertices in a control flow graph, i.e., multiple clauses change the value of a CFP in the same way. Because of this sizes is defined as set. This way we can identify if there is a discrepancy in the sizes of the sets uf, ui and cb for any of the edges.

Example 7. Recall Example 5. Both CFGs have $\{X(\mathsf{true}), Y(\mathsf{true}), Y(\mathsf{false})\}$ as their vertex set. Between every two vertices there is only one edge in either CFG. This gives $\mathsf{sizes}(c_1, Y(\mathsf{false}), X(\mathsf{true})) = \mathsf{sizes}(c_2, Y(\mathsf{false}), X(\mathsf{true})) = \{(2, 2, 2)\}$. From Example 6 we see that the set sizes equal for all corresponding edges. Hence, c_1 and c_2 are compatible.

A maximal set of compatible CFPs is called a *clique*. The set of all cliques for a PBES-SRF is denoted $\mathsf{cliques}(\mathcal{E})$. As compatibility is an equivalence relation, this is a partition. Next we define the set of DPs that play a role for any of the CFPs within a clique.

Definition 14. *The set of data parameters for a clique I is defined as follows.*

$$\mathsf{dp}(I) = \bigcup_{c \in I, e \in \mathsf{E}_c} (\mathsf{cb}(e) \cup \mathsf{uf}(e) \cup \mathsf{ui}(e))$$

In the search for symmetries we use *clique permutation*. These permutations interchange CFPs within one clique. Also, they never swap a CFP with a DP.

Definition 15. *The set of clique permutations for a clique I is the following.*

$$\mathsf{cliqper}(I) = \{\alpha\beta \mid \alpha \in \mathsf{per}(I), \beta \in \mathsf{per}(\mathsf{dp}(I))\}$$

Example 8. Recall Example 5. For this example there is only one clique $I = \{c_1, c_2\}$ with $\mathsf{dp}(I) = \{d_1, d_2\}$. From this we get that $\mathsf{per}(I) = \{\mathrm{id}, (c_1c_2)\}$ and $\mathsf{per}(\mathsf{dp}(I)) = \{\mathrm{id}, (d_1d_2)\}$. So $\mathsf{cliqper}(I) = \{\mathrm{id}, (c_1c_2), (d_1d_2), (c_1c_2)(d_1d_2)\}$.

A clique permutation is a *candidate symmetry* if applying the permutation to PBES-SRF leaves intact the structure of the CFGs. To capture when this structure is left intact we first define when a permutation *complies* with a CFP.

Definition 16. *A clique permutation π* complies *with a CFP c if and only if there exists a bijection $\zeta \colon \mathsf{E}_c \to \mathsf{E}_{\pi(c)}$, such that for all $e \in E_c$ it holds that*

$$\pi(\mathsf{uf}(e)) = \mathsf{uf}(\zeta(e)) \wedge \pi(\mathsf{ui}(e)) = \mathsf{ui}(\zeta(e)) \wedge \pi(\mathsf{cb}(e)) = \mathsf{cb}(\zeta(e))$$

We write $\mathsf{complies}(\pi, c)$ when π complies with c and $\mathsf{complies}(\pi, I)$ is the conjunction of $\mathsf{complies}(\pi, c)$ for $c \in I$. Next we define the set of candidate symmetries.

Definition 17. *Let I be a clique. The set of* candidate symmetries *for I is*

$$\mathsf{candidates}(I) = \{\pi \mid \pi \in \mathsf{cliqper}(I) \wedge \mathsf{complies}(\pi, I)\}$$

As the set of clique permutations is a group, we deduce that the complies predicate also preserves inverse and composition. From this we conclude that the set of candidate symmetries also is a group.

We specify how to *combine* the candidates for each individual clique, such that we generate candidate symmetries for $\mathcal{E}$ as a whole. We combine the candidate sets for two cliques in such a way that all the permutations created, again comply with both cliques by construction.

Definition 18. *For two cliques I_1 and I_2 we define:*

$$\mathsf{combine}(I_1, I_2) = \{\alpha_1\alpha_2\beta \mid \alpha_1\beta \in \mathsf{candidates}(I_1) \wedge \alpha_2\beta \in \mathsf{candidates}(I_2)\}$$

The operator combine is associative and commutative. Because of this, we can also write $\mathsf{combine}(\mathcal{I})$, where $\mathcal{I}$ is a non-empty set of cliques, to denote the iterative application of combine on the cliques in $\mathcal{I}$. If $\mathcal{I}$ only contains one clique, then combine just returns the candidates of that clique. If $\mathcal{I} = \emptyset$ then combine returns no candidates, i.e., the empty set. The set of candidate symmetries for $\mathcal{E}$ is now defined as $\mathsf{candidates}(\mathcal{E}) = \mathsf{combine}(\mathsf{cliques}(\mathcal{E}))$. Like the set of candidates for a clique, the set of candidates for a PBES-SRF is a group.

To construct the candidate symmetries for a PBES-SRF we only consider the structure of the CFGs. Therefore, the set of candidates we computed is an over-approximation of the set of syntactic symmetries. As a final step, we check each candidate against the definition To perform this check we define the predicate symcheck that resembles Definition 8. We have that $\mathsf{symcheck}(\pi, \mathcal{E}) = \mathsf{true}$ if and only if π is a syntactic symmetry.

Definition 19. *Consider the PBES-SRF $\mathcal{E}$ and a candidate symmetry π. We define the predicate* symcheck *as follows*

$$\mathsf{symcheck}(\pi, \mathcal{E}) := \bigwedge_{X \in \mathsf{bnd}(\mathcal{E})} \bigwedge_{j \in J_X} \bigvee_{j' \in J_X} (\pi(f_j) = f_{j'}) \wedge (X_j = X_{j'}) \wedge (\pi(g_j) = g_{j'})$$

The predicate symcheck performs a purely syntactic check leaving us with syntactical symmetries. We point out that this predicate can be weakened (e.g. by considering logical equivalence) to possibly obtain more symmetries.

Definition 20. *The set of* syntactic symmetries *for $\mathcal{E}$ is defined as follows.*

$$\mathsf{symmetries}(\mathcal{E}) = \{\pi \mid \pi \in \mathsf{candidates}(\mathcal{E}) \wedge \mathsf{symcheck}(\pi, \mathcal{E})\}$$

Example 9. Recall Example 5. We have that $\mathsf{complies}((c_1c_2)(d_1d_2), I)$ holds. Also the identity permutation trivially complies with I. The other clique permutations do not comply with I. Hence, $\mathsf{candidates}(I) = \{\mathrm{id}, (c_1c_2)(d_1d_2)\}$. As there is only one clique $\mathsf{combine}(\mathcal{E}) = \mathsf{candidates}(I)$. The permutation $(c_1c_2)(d_1d_2)$ is a syntactic symmetry for $\mathcal{E}$. This means that $\mathsf{symmetries}(\mathcal{E}) = \{\mathrm{id}, (c_1c_2)(d_1d_2)\}$.

Consider a PBES-SRF with n parameters. Naively, the set of permutations that have to be checked with symcheck is of size $n!$. When selecting the set of candidates, an upper bound for this number is $|\mathcal{I}_1|! \cdot \ldots |\mathcal{I}_m|! \cdot |\mathsf{dp}(\mathcal{E})|!$, for $1 \leq i \leq m$, $\mathcal{I}_i \in \mathsf{cliques}(E)$. This illustrates that selecting the candidates is worthwhile.

The set $\mathsf{symmetries}(\mathcal{E})$ is a subgroup of the set of candidates, meaning it is also a group. Moreover, every permutation in $\mathsf{symmetries}(\mathcal{E})$ is a syntactic symmetry for $\mathcal{E}$. Because of this, we can use $\mathsf{symmetries}(\mathcal{E})$: a PBES-SRF can now be solved by using the extracted set of symmetries to generate the quotient parity game. This is illustrated by the following theorem.

Theorem 3. *Let $G = \mathsf{symmetries}(\mathcal{E})$. Then for $X \in \mathsf{bnd}(\mathcal{E})$ and $v \in \mathbb{D}$ it holds that $\boldsymbol{v}$ is in the solution of X in $\mathcal{E}$ if and only if $X(\theta(\boldsymbol{v}))$ won by $\Diamond$ in $M^G_{\mathcal{E}}$.*

5 Experiments

Setup. We implemented our technique for extracting symmetries within the mCRL2 toolset. Our tool converts a given PBES into a PBES-SRF, then carries out a control flow analysis and subsequently generates and checks candidate symmetries. As listing all symmetries is often expensive, the tool outputs the first symmetry it comes across and then terminates. The generation of quotient games is implemented as an extension of the existing mCRL2 tool `pbessolve`. For effectively storing quotient games, we use orbit representatives. To compute these we use GAP [8] – a tool for computational discrete algebra with an emphasis on computational group theory. Using an external tool introduces computational overhead. To make a fair comparison, we call GAP in the original approach using the identity symmetry, i.e. no reduction is applied. The experiments were run five times on a machine with an Intel core i7-12700H processor and 16GB of

memory running Ubuntu 22.04. A reproduction artefact is available in the form of a preconfigured VM [1]. The source code of the mCRL2 tools used together with the models, properties and chosen symmetries are also available[1].

We compare three workflows using `pbessolve`. The first one serves as a baseline without symmetry reduction. The next workflow showcases the proposed technique. Our tool automatically identifies a symmetry and then applies symmetry reduction using the first symmetry it finds. As the tool outputs the first symmetry it comes across, it is often the case that there are different symmetries that give more reduction. Using the analysis the tool provides, we manually identify such a symmetry. For the last workflow we apply symmetry reduction with this well-chosen symmetry to showcase the potential reduction symmetry reduction can achieve. We set a timeout (t-o) of 30 minutes for both solving and detection. When solving timed out, the number of states is not computed and when detection timed out, solving was not done.

Cases. The benchmark set consists of mCRL2 models of Peterson's algorithm [22] (mutex), the dining philosophers (dining) and two models that take inspiration from models found in [7] (alloc and routing). As the numbers in the model names suggest, dining, alloc and routing are variable in the number of philosophers, clients and servers they include, respectively. We check the following properties: always some action can be performed (no_deadlock), any confirm action is preceded by a request action (no_conf_before_req), no infinite sequence of eat actions exists (no_inf_eat), every sequence of lookup actions is finite (fin_lu), there are no consecutive query actions (no_con_query). As shown in Table 1 some of these properties hold (✓) and some do not (×).

Discussion. Our findings are presented in Table 1 The numbers concern the three workflows of `pbessolve`: using no symmetry reduction (Original), symmetry reduction with detected symmetry (First) and symmetry reduction with hand-picked symmetry (Chosen). We show the number of states in the associated parity game ($|V|$), and the mean total runtime for solving over five runs (Time). Additionally, the time for symmetry detection is given (Detection).

The results show that symmetry reduction is always beneficial when applied. In every case where symmetry reduction is applied the size of the parity game ($|V|$) reduces (approximately) linearly with the amount of symmetric parameters in the symmetry used. Consequently, the running time of `pbessolve` (Time) decreases similarly. Adding up the time spent on symmetry detection shows that our tool outperforms the original tool in most cases. In two cases the time spent on symmetry detection exceeds the time saved in solving. In the first case the detected symmetry is of the same size as the hand-picked symmetry (♠). Because of this the tool might have to check a substantial amount of smaller candidates before an actual symmetry is found. In the second case, the candidates to be checked are compatible, but do not comply (■). To deduce compliance a semantic check is performed. This is more costly than the syntactic check needed

[1] `https://github.com/mmgbartels/pbes-symmetry-experiments`

Table 1. Results for symmetry detection and reduction. Numbers given for three runs of `pbessolve`: no symmetry reduction (Original), using detected symmetry (First), using hand-picked symmetry (Chosen). Depicted are the number of parity game states ($|V|$), mean total time in seconds (Time), and time for detection in seconds (Detection). Timeout (t-o) is 30 minutes.

			$\|V\|$			Time			
Model	Property	Result	Original	First	Chosen	Original	First	Detection	Chosen
mutex	no_deadlock	✓	125	64	64	0.11	0.11	+0.03	0.11
alloc4	no_conf_before_req	✓	33223	18919	8503	2.98	1.80	+0.04	0.88
	no_deadlock	✓	46515	26526	11910	5.50	3.21	+0.03	1.54
alloc5	no_conf_before_req	✓	237799	135511	47623	26.408	15.587	+0.039	5.637
	no_deadlock	✓	329601	188058	66009	49.241	28.256	+0.035	10.304
alloc6	no_conf_before_req	✓	1695751	966775	283631	232.27	133.93	+0.05	40.93
	no_deadlock	✓	2330643	1330122	389868	427.05	246.42	+0.04	75.20
alloc7	no_conf_before_req★	✓		- 6876343	1722487	t-o	1176.48	+0.10	307.69
	no_deadlock	✓		-	- 2350749	t-o	t-o	0.06	559.83
dining5	no_deadlock	×	5464	1100	1100	0.778	0.247	+1.521♠	0.251
	no_inf_eat	✓	3916	788	788	0.454	0.175	+1.126♠	0.177
dining6	no_deadlock	×	25167	4261	4261	4.10	0.81	+54.44♠	0.83
	no_inf_eat	✓	17952	3036	3036	2.07	0.48	+39.93♠	0.45
dining7	no_deadlock	×	115271	-	16475	21.85	-	t-o	3.40
	no_inf_eat	✓	82060	-	11728	11.14	-	t-o	1.81
routing4	fin_lu	✓	95656	51808	24088	12.19	6.85	+4.92	3.45
	no_con_query	✓	105973	57373	26683	11.66	6.48	+0.11	3.20
	no_deadlock	×	141738	76684	35684	20.51	11.62	+0.08	5.44
routing5	fin_lu	✓	1100920	598288	220216	179.17	98.76	+697.60■	37.31
	no_con_query	✓	1223773	664873	244789	175.11	96.81	+0.85	36.84
	no_deadlock	×	1625559	882806	325159	315.66	173.42	+0.66	65.67
routing6	fin_lu	✓		-	- 2052928	t-o	-	t-o	448.25
	no_con_query★	✓		- 7468873	2287363	t-o	1399.12	+24.39	445.09
	no_deadlock	×		-	- 3020467	t-o	t-o	23.13	794.21
routing7	fin_lu	-		-	-	t-o	-	t-o	t-o
	no_con_query	-		-	-	t-o	t-o	1093.68	t-o
	no_deadlock	-		-	-	t-o	t-o	1266.98	t-o

for compatibility. The experiments also show that using a hand-pick symmetry may further reduce the effort needed (Chosen). Moreover, our technique is able to solve the model checking problem in cases where the original approach times out (★).

6 Conclusions and Future work

We have defined the theory of symmetries for PBESs and showed that permutations act on predicate formulas. This enabled us to carry out symmetry based quotienting on the parity game semantics underlying PBESs. We presented a technique to extract syntactic symmetries using the control flow found in a PBES. With a prototype tool we have shown the effectiveness of our technique and the substantial reductions that can be obtained.

One direction for future work is to improve the selection and check of candidate symmetry. Other syntactic (and possibly semantic) notions can be used

to rule out candidates at an even earlier stage. This could make the technique more scalable – the number of candidates depends on the number of parameters, which typically grows with the number of components in a model. To improve the selection of candidates it would be interesting to investigate the use of colored graphs like is done in [7]. There GAP is used to obtain a group of candidates using a colored graph containing the structural information of the system at hand. This approach might speed up the selection of the candidates, as our current implementation generates and checks these in a naive manner. Moreover, this technique can generate more interesting symmetries, which in turn leads to more reduction. Another direction for future work could investigate, in addition to permuting only the parameters, also permuting the values that these parameters can attain. With this extension, symmetry reduction can also be applied in the presence of so-called data symmetries [13].

References

1. Bartels, M., Laveaux, M., Neele, T., Willemse, T.: Artefact for paper: "Control Flow-Based Symmetry Reduction for Parameterised Boolean Equation Systems" (2026). https://doi.org/10.5281/zenodo.19333256
2. Bunte, O., Groote, J.F., Keiren, J.J.A., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, W., Wijs, A., Willemse, T.A.C.: The mCRL2 Toolset for Analysing Concurrent Systems - Improvements in Expressivity and Usability. In: TACAS (2). pp. 21–39. LNCS, Springer (2019). https://doi.org/10.1007/978-3-030-17465-1_2
3. Church, A.: A Note on the Entscheidungsproblem. J. Symb. Log. **1**(1), 40–41 (1936). https://doi.org/10.2307/2269326
4. Clarke, E.M., Emerson, E.A., Jha, S., Sistla, A.P.: Symmetry Reductions in Model Checking. In: CAV. pp. 147–158. LNCS, Springer (1998). https://doi.org/10.1007/BFB0028741
5. Clarke, E.M., Jha, S., Enders, R., Filkorn, T.: Exploiting Symmetry in Temporal Logic Model Checking. Formal Methods Syst. Des. **9**(1/2), 77–104 (1996). https://doi.org/10.1007/BF00625969
6. Cranen, S., Luttik, B., Willemse, T.A.C.: Evidence for Fixpoint Logic. In: CSL. pp. 78–93. LIPIcs, Schloss Dagstuhl (2015). https://doi.org/10.4230/LIPICS.CSL.2015.78
7. Donaldson, A.F.: Automatic Techniques for Detecting and Exploiting Symmetry in Model Checking. Ph.D. thesis, University of Glasgow, UK (2007)
8. The GAP Group: GAP – Groups, Algorithms, and Programming, Version 4.15.1 (2025), https://www.gap-system.org
9. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. Int. J. Softw. Tools Technol. Transf. **15**(2), 89–107 (2013). https://doi.org/10.1007/S10009-012-0244-Z
10. Groote, J.F., Willemse, T.A.C.: Parameterised boolean equation systems. Theor. Comput. Sci. **343**(3), 332–369 (2005). https://doi.org/10.1016/J.TCS.2005.06.016
11. Hendriks, M., Behrmann, G., Larsen, K.G., Niebert, P., Vaandrager, F.W.: Adding Symmetry Reduction to Uppaal. In: FORMATS. pp. 46–59. LNCS, Springer (2003). https://doi.org/10.1007/978-3-540-40903-8_5

12. Ip, C.N., Dill, D.L.: Better Verification Through Symmetry. Formal Methods Syst. Des. **9**(1/2), 41–75 (1996). https://doi.org/10.1007/BF00625968
13. Jackson, D., Jha, S., Damon, C.: Isomorph-Free Model Enumeration: A New Method for Checking Relational Specifications. ACM Trans. Program. Lang. Syst. **20**(2), 302–343 (1998). https://doi.org/10.1145/276393.276396
14. Kant, G., van de Pol, J.: Efficient Instantiation of Parameterised Boolean Equation Systems to Parity Games. In: GRAPHITE. pp. 50–65. EPTCS (2012). https://doi.org/10.4204/EPTCS.99.7
15. Keiren, J.J.A., Wesselink, W., Willemse, T.A.C.: Liveness Analysis for Parameterised Boolean Equation Systems. In: ATVA. pp. 219–234. LNCS, Springer (2014). https://doi.org/10.1007/978-3-319-11936-6_16
16. Kwiatkowska, M.Z., Norman, G., Parker, D.: Symmetry Reduction for Probabilistic Model Checking. In: CAV. pp. 234–248. LNCS, Springer (2006). https://doi.org/10.1007/11817963_23
17. Markey, N., Vester, S.: Symmetry Reduction in Infinite Games with Finite Branching. In: ATVA. pp. 281–296. LNCS, Springer (2014). https://doi.org/10.1007/978-3-319-11936-6_21
18. Neele, T.: (Re)moving Quantifiers to Simplify Parameterised Boolean Equation Systems. In: ARQNL@IJCAR. pp. 64–80. CEUR Workshop Proceedings, CEUR-WS.org (2022)
19. Neele, T., Willemse, T.A.C., Groote, J.F.: Finding compact proofs for infinite-data parameterised Boolean equation systems. Sci. Comput. Program. **188**, 102389 (2020). https://doi.org/10.1016/J.SCICO.2019.102389
20. Neele, T., Willemse, T.A.C., Wesselink, W., Valmari, A.: Partial-order reduction for parity games and parameterised Boolean equation systems. Int. J. Softw. Tools Technol. Transf. **24**(5), 735–756 (2022). https://doi.org/10.1007/S10009-022-00672-0
21. Orzan, S., Wesselink, W., Willemse, T.A.C.: Static Analysis Techniques for Parameterised Boolean Equation Systems. In: TACAS. pp. 230–245. LNCS, Springer (2009). https://doi.org/10.1007/978-3-642-00768-2_22
22. Peterson, G.L.: Myths About the Mutual Exclusion Problem. Inf. Process. Lett. **12**(3), 115–116 (1981). https://doi.org/10.1016/0020-0190(81)90106-X
23. van de Pol, J., Timmer, M.: State Space Reduction of Linear Processes Using Control Flow Reconstruction. In: ATVA. pp. 54–68. LNCS, Springer (2009). https://doi.org/10.1007/978-3-642-04761-9_5
24. Rotman, J.: An Introduction to the Theory of Groups, Graduate Texts in Mathematics, vol. 148. Springer (2012). https://doi.org/10.1007/978-1-4612-4176-8
25. Sistla, A.P.: Employing symmetry reductions in model checking. Comput. Lang. Syst. Struct. **30**(3-4), 99–137 (2004). https://doi.org/10.1016/J.CL.2004.02.002
26. Sistla, A.P., Godefroid, P.: Symmetry and Reduced Symmetry in Model Checking. In: CAV. pp. 91–103. LNCS, Springer (2001). https://doi.org/10.1007/3-540-44585-4_9
27. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. **s2-42**(1), 230–265 (1937). https://doi.org/10.1112/PLMS/S2-42.1.230
28. Volk, M., Junges, S., Katoen, J.: Fast Dynamic Fault Tree Analysis by Model Checking Techniques. IEEE Trans. Ind. Informatics **14**(1), 370–379 (2018). https://doi.org/10.1109/TII.2017.2710316

29. Wahl, T., Blanc, N., Emerson, E.A.: SVISS: Symbolic Verification of Symmetric Systems. In: TACAS. pp. 459–462. LNCS, Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_34
30. Zielonka, W.: Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. Theor. Comput. Sci. **200**(1-2), 135–183 (1998). https://doi.org/10.1016/S0304-3975(98)00009-7