Symmetry Reduction for Probabilistic Model Checking Using Generic Representatives

Alastair F. Donaldson[⋆] and Alice Miller

Department of Computing Science
University of Glasgow
Glasgow, Scotland
{ally,alice}@dcs.gla.ac.uk

Abstract. Generic representatives have been proposed for the effective combination of symmetry reduction and symbolic representation with BDDs in non-probabilistic model checking. This approach involves the translation of a symmetric source program into a *reduced* program, in which counters are used to *generically* represent states of the original model. Symmetric properties of the original program can also be translated, and checked directly over the reduced program. We extend this approach to apply to probabilistic systems with Markov decision process or discrete time Markov chain semantics, represented as MTBDDs. We have implemented a prototype tool, GRIP, which converts a symmetric PRISM program and $PCTL$ property into reduced form. Model checking results for the original program can then be inferred by applying PRISM, unchanged, to the smaller model underlying the reduced program. We present encouraging experimental results for two case studies.

1 Introduction

Symmetry reduction techniques can be effective at combatting the state space explosion problem for model checking [4,5,9,12,13,22]. Replication in the structure of a distributed system can give rise to symmetries, or *automorphisms*, of a Kripke structure modelling the system: bijections of the states of the structure which preserve its transition relation. The most common type of Kripke structure automorphisms are *component symmetries* – permutations of the set of component identifiers which preserve the transition relation when applied to all states. A group of automorphisms gives rise to a partition of the states of a structure into equivalence classes, or *orbits*, and a *quotient* Kripke structure can be constructed by choosing a single representative from each orbit. If the group is non-trivial this structure is smaller than the original, and they are bisimilar. Thus temporal logic properties which are symmetrically invariant can be checked over the smaller quotient structure.

The obvious approach to combining symmetry reduction techniques with symbolic representation is to represent the quotient Kripke structure using a binary

[⋆] Supported by the Carnegie Trust for the Universities of Scotland.

decision diagram (BDD). However, this approach requires a BDD to be constructed for the *orbit relation* — for a symmetry group G this is the set of pairs of symmetrically equivalent states under G. It has been shown that for a large class of commonly occurring symmetry groups construction of such a BDD representation is intractable [6]. A promising approach which avoids computing the orbit relation for the case of full component symmetry uses *generic representatives* [12]. Here an equivalence class of states is represented by a state which *counts* the number of components in each local state, ignoring the specific component identities. This state is a *generic* representative. Using mutual exclusion as a canonical example, if the local state of a component belongs to $\{N, T, C\}$ (Non-critical, Trying and Critical), the global states for a 4-component model include (N, T, N, C), (C, T, N, N) and (N, C, N, T), which are symmetrically equivalent and are all represented generically by $(2N, 1T, 1C)$. The attraction of this approach is that symmetry can be exploited at the source code level: a fully symmetric source program and fully symmetric property can be translated into a *reduced* program and property. The Kripke structure for the reduced program is isomorphic to the quotient structure of the original under full symmetry, and the generic property is equivalent to the original. Thus the benefits of symmetry reduction can be obtained by applying standard symbolic model checking algorithms to the reduced program.

Recently there has been much interest in probabilistic model checking [1,18,25], and some work on extending symmetry reduction techniques to a probabilistic setting [8,20]. In this paper we extend the generic representatives approach to apply to symbolic model checking of probabilistic systems. We define a symmetric subset of the PRISM modelling language, SP, and show how SP programs with Markov decision process (MDP) or discrete time Markov chain (DTMC) semantics, together with symmetric PCTL properties, can be translated into a reduced form. As in the non-probabilistic case, the time complexity of this translation is polynomial in the size of the input program. We describe a software tool, GRIP, which automates the translation of an SP program to generic form, and illustrate our approach using two case studies.

2 Background

2.1 Symmetry Reduction for Non-probabilistic Model Checking

We use a simple model to represent the computation of a system comprised of n communicating components, interleaving concurrently [11]. Let $I = \{1, 2, \ldots, n\}$ be the set of component identifiers, and for some $k \geq 0$, let $L = \{0, 1, 2, \ldots, k\}$ denote the possible local states of the components. A Kripke structure is a pair $\mathcal{K} = (S, R)$, where $S \subseteq L^n$, is a non-empty set of states, and $R \subseteq S \times S$ is a total transition relation. The usual lexicographical ordering of vectors provides a total ordering on S. If $s = (l_1, l_2, \ldots, l_n) \in S$ then we use $s(i)$ to denote l_i, the local state of component i. Communication between components is via inspection of local states.

The set of all permutations of I forms a group under composition of mappings, denoted S_n (the symmetric group on n points). Let $\mathcal{K} = (S, R)$ be a Kripke structure, and let $\alpha \in S_n$. Then α acts on a state $s = (l_1, l_2, \ldots, l_n) \in S$ in the following way: $\alpha(s) = (l_{\alpha^{-1}(1)}, l_{\alpha^{-1}(2)}, \ldots, l_{\alpha^{-1}(n)})$. If $(\alpha(s), \alpha(t)) \in R \; \forall \; (s,t) \in R$, α is an *automorphism* of $\mathcal{K}$. The set of all automorphisms of $\mathcal{K}$ forms a group $Aut(\mathcal{K}) \leq S_n$ under composition of mappings.

A subgroup G of $Aut(\mathcal{K})$ induces an equivalence relation $\theta = \{(s, \alpha(s)) : s \in S, \alpha \in G\}$ on the states of $\mathcal{K}$. The equivalence class under θ of a state $s \in S$, denoted $[s]$, is called the *orbit* of s under the action of G, and θ is the *orbit relation.* The *smallest* element of $[s]$ under the total ordering described above is denoted $min[s]$. The quotient Kripke structure for $\mathcal{K}$ with respect to G is a pair $\overline{\mathcal{K}} = (\overline{S}, \overline{R})$ where $\overline{S} = \{min[s] : s \in S\}$, and $\overline{R} = \{(min[s], min[t]) : (s,t) \in R\}$. If G is non-trivial $\overline{\mathcal{K}}$ is a smaller structure than $\mathcal{K}$, but $\overline{\mathcal{K}}$ and $\mathcal{K}$ are equivalent in the sense that they satisfy the same set of temporal logic properties which are *invariant* under the group G. Thus by choosing a suitable symmetry group G, model checking can be performed over $\overline{\mathcal{K}}$ instead of $\mathcal{K}$, potentially resulting in considerable savings in memory and verification time.

2.2 Symmetry Reduction and Symbolic Representation

Our definition of a quotient structure in the previous section involves the selection of the *smallest* state of an equivalence class as a representative. However, any representative function which maps all elements of a class on to the same unique representative could be used. Symmetry reduction techniques can in principle be combined with symbolic representation by constructing a BDD for such a representative function, and applying the function during fixpoint iterations. This is described in detail in [6], and summarised in [12]. Unfortunately, constructing the representative function requires a BDD for the orbit relation which, for many commonly occurring symmetry groups, has size exponential in $min(k+1, n)$ [6].

Several alternative approaches to combining symmetry reduction and symbolic model checking have been proposed, including the use of multiple representatives [5]; under-approximation [3]; dynamic symmetry reduction [13]; and generic representatives [11,12,14]. Summaries of these approaches can be found in [20,22]. In this paper, we restrict our attention to the generic representatives approach, which we now describe.

In order to avoid constructing a BDD for the orbit relation when exploiting full component symmetry, a fully symmetric source program can be translated into a reduced program, which can be checked using standard symbolic model checking algorithms and has a state space isomorphic to the symmetric quotient structure of the model underlying the original program. A program with n components and $k+1$ local states per component is translated into a program with $k+1$ *counter* variables, each with domain $I \cup \{0\}$ [11]. A state of this program indicates *how many* processes are in each local state of the original program, but does not refer to individual processes (see the mutual exclusion example in Section 1). This approach is extended [12] to include systems with global shared variables. The translation of a program into reduced form is polynomial in the length of the program and the approach compares well to those using unique or multiple representatives.

Details of the translation of a fully symmetric program into reduced form can be found in [12], and the approach is similar to the one we present for fully symmetric PRISM programs with MDP semantics in Section 5. The approach is limited as it only applies to fully symmetric systems and requires a somewhat restrictive input language. However, full component symmetry is the most common kind of symmetry in model checking problems, and promises the best state space reduction of any kind of symmetry.

3 Symmetry Reduction for Probabilistic Models

We now consider systems comprised of n *stochastic* components which interleave concurrently. Once again, communication between components is achieved by inspection of local states. Let I and L be as before. A Markov decision process (MDP) is a pair $\mathcal{M} = (S, \mathit{Steps})$, where $S \subseteq L^n$, and $\mathit{Steps} : S \to 2^{\mathit{Dist}(S)}$ maps each state s to a finite, non-empty set of probability distributions over S. A discrete time Markov chain (DTMC) is a pair $\mathcal{D} = (S, P)$ where S is as for an MDP, and $P : S \times S \to [0, 1]$ is a *transition probability matrix*. An MDP can model systems which exhibit both nondeterminism and probabilistic behaviour (e.g. nondeterministic scheduling of processes in a randomised distributed algorithm), whereas DTMCs can be used to model purely probabilistic systems.

For either type of model, a total ordering on states is provided as before by the usual lexicographic ordering on vectors, and the action of a permutation $\alpha \in S_n$ on states is the same as that described in Section 2.1. Recall that Kripke structure automorphisms preserve the transition relation of the structure. Automorphisms of probabilistic structures preserve the *probabilistic* transition relation. The following definitions are adapted from [20].

Definition 1. *Let $\mathcal{M} = (S, \mathit{Steps})$ and $\mathcal{M}' = (S', \mathit{Steps}')$ be MDPs, and let $\alpha : S \to S'$ be a bijection. Suppose that for all $s \in S$ and for all $\mu \in \mathit{Steps}(s)$, there exists $\mu' \in \mathit{Steps}(\alpha(s))$ such that, for all $t \in S$, $\mu(t) = \mu'(\alpha(t))$. Then α is an isomorphism from $\mathcal{M}$ to $\mathcal{M}'$, and $\mathcal{M}$ and $\mathcal{M}'$ are said to be isomorphic. If $\mathcal{M} = \mathcal{M}'$, α is an automorphism of $\mathcal{M}$.*

Definition 2. *Let $\mathcal{D} = (S, P)$ and $\mathcal{D}' = (S', P')$ be DTMCs, and let $\alpha : S \to S'$ be a bijection. Suppose that for all $s, t \in S$, $P(s, t) = P'(\alpha(s), \alpha(t))$. Then α is an isomorphism from $\mathcal{D}$ to $\mathcal{D}'$, and $\mathcal{D}$ and $\mathcal{D}'$ are said to be isomorphic. If $\mathcal{D} = \mathcal{D}'$, α is an automorphism of $\mathcal{D}$.*

In both cases, the set of all automorphisms forms a group under composition, denoted $\mathit{Aut}(\mathcal{D})$ or $\mathit{Aut}(\mathcal{M})$, and a subgroup of this group induces orbits on the state space as before. Taking the minimum element of each orbit as a representative, a quotient DTMC/MDP can be defined analogously to the non-probabilistic case [20].

The quotient DTMC $\overline{\mathcal{D}} = (\overline{S}, \overline{P})$ is defined by $\overline{S} = \{\mathit{min}[s] : s \in S\}$, and $\overline{P}(\mathit{min}[s], \mathit{min}[t]) = \sum_{x \in [t]} P(\mathit{min}[s], x)$. For a quotient MDP $\overline{\mathcal{M}} = (\overline{S}, \overline{\mathit{Steps}})$,

$\overline{S}$ is defined similarly and, if $min[s] \in \overline{S}$ then $\overline{\mu} \in \overline{Steps}(min[s])$ iff there is some $\mu \in Steps(min[s])$ such that, for all $min[t] \in \overline{S}$, $\overline{\mu}(min[t]) = \sum_{x \in [t]} \mu(x)$. It is easy to show using results on probabilistic bisimulation [21,26] that, as in the non-probabilistic case, the quotient models preserve the truth of temporal properties which are *invariant* under symmetry. This means that for each maximal propositional subformula f of a property ϕ, and for all $\alpha \in G$, $s \models f \Leftrightarrow \alpha(s) \models f$. To express properties of MDPs or DTMCs we use $PCTL$ (Probabilistic Computation Tree Logic) [17].

Theorem 1. *Let ϕ be a $PCTL$ property which is invariant under G. Then, for all $s \in S$,*

$$\mathcal{M}, s \models \phi \Leftrightarrow \overline{\mathcal{M}}, min[s] \models \phi.$$

Formulas in the sub-logic $SPCTL$, described in Section 4.1, are invariant under full symmetry.

The following theorem establishes a correspondence between properties of isomorphic MDPs under an appropriate transformation of atomic propositions. We omit the proof, which is straightforward using induction on the structure of $PCTL$ formulas.

Theorem 2. *Let $\mathcal{M} = (S, Steps)$ and $\mathcal{M}' = (S', Steps')$ be MDPs, F and F' sets of propositions over the local states of components of $\mathcal{M}$ and $\mathcal{M}'$ respectively, and $\gamma : F \to F'$ a bijection. For a $PCTL$ formula ϕ with maximal propositional subformulas taken from F, $\gamma(\phi)$ is the $PCTL$ formula with maximal propositional subformulas taken from F', obtained from ϕ by replacing every subformula f with $\gamma(f)$. Let δ be an isomorphism from $\mathcal{M}$ to $\mathcal{M}'$ such that, for all $s \in S$ and $f \in F$, $s \models f \Leftrightarrow \delta(s) \models \gamma(f)$. Then for any $PCTL$ formula ϕ over F and $s \in S$,*

$$\mathcal{M}, s \models \phi \Leftrightarrow \mathcal{M}', \delta(s) \models \gamma(\phi).$$

Analogous versions of Theorems 1 and 2 hold for DTMCs.

As with non-probabilistic symbolic model checking, construction of a quotient model as a multi-terminal BDD (MTBDD) is intractable for commonly occurring symmetry groups. We now define a subset of the PRISM modelling language for specification of fully symmetric MDP or DTMC models. In Section 5 we show how the generic representatives approach of [11,12] can be extended to exploit the symmetry inherent in these models.

4 Symmetric PRISM

We now define an input language for specifying fully symmetric programs. The language defined is a subset of the PRISM modelling language, which we call *Symmetric PRISM* (SP). An SP program consists of a module `process1`, and $n-1$ renamed copies of this module, denoted `process2`,..., `process`n. Each module has a single variable, `s`i, which has domain L and is initialised to 0. For ease of presentation, we sometimes refer to $\mathtt{s}_i$ rather than `s`i. Every statement of a module consists of a compound guard, followed by a probabilistic choice of

spec	::=	`nondeterministic` *main_module other_modules* |
		`probabilistic` *main_module other_modules*
main_module	::=	`module process1`
		`s1 : [0..`k`] init` l`;` *statements*
		`endmodule` $(l \in L)$
statements	::=	*statement* | *statement statements*
statement	::=	`[]` *local_guard* `&` *guard* `->` *stochastic_update*`;`
local_guard	::=	`s1=`j $(j \in L)$
stochastic_update	::=	λ_1`:(s1'=`j_1`) +` λ_2`:(s1'=`j_2`) + ... +` λ_v`:(s1'=`j_v`)`
		$(v > 0, \lambda_i \in [0,1], j_i \in L)$
other_modules	::=	`module process2 = process1 [s1=s2, s2=s1] endmodule`
		⋮
		`module process`n `= process1 [s1=s`n`, s`n`=s1] endmodule`

Fig. 1. Syntax of Symmetric PRISM

updates to $\mathtt{s}_i$ (a *stochastic* update). The compound guard is a conjunction of a *local guard*, which has the form $\mathtt{s}_i$=j, for some $j \in L$, and an optional guard over the variables $\mathtt{s}_1$,$\mathtt{s}_2$,...,$\mathtt{s}_n$.

The core grammar of SP is given in Figure 1, while the form of optional guards is presented in Table 1. The *generic form* column of Table 1 will be explained in Section 5. For conciseness, the quantifiers $\forall$ and $\exists$ are used to denote conjunctions and disjunctions over all (or all but one) components. For example, the guard (`s1=`j `&` `s2=`j `&`...`&` `s`n`=`j) is denoted by $\forall_i$ $\mathtt{s}_i$=j in the table. The last four forms of guard in Table 1, together with module renaming, allow conditions on just the state of *other* modules than that in which the guard appears. This extends the form of guards allowed in [11,12], and requires more complex rules for translation into generic form (see Section 5). In Figure 1, the keywords `nondeterministic` and `probabilistic` indicate that the underlying model is an MDP or DTMC respectively, and we say that $\mathcal{P}$ is a *nondeterministic* or *probabilistic* program, using $\mathcal{M}(\mathcal{P})/\mathcal{D}(\mathcal{P})$ to denote the model. Each statement in an SP program consists of a guard followed by a *stochastic update*. The stochastic update is a probabilistic choice over local updates of the form λ_i`:(s1'=`j_i`)`, where $\lambda_i \in [0,1]$ is the probability of this local update being chosen, and $\sum_{i=1}^{v} \lambda_i = 1$. Note that SP programs cannot include multiple local variables in modules, multiple module types, or communication by synchronisation.

We illustrate the syntax of SP by modelling a minimum space shared memory leader election protocol [7]. The protocol is carried out by a set of n processors, each with a single-writer multi-reader binary register (the *leader* register). Eventually, all of these registers apart from one will be set to zero (the *election condition* will be satisfied). The process for whom the associated register is non zero is chosen as the leader. At each stage of the protocol, if the election condition is not satisfied then for each processor P_i such that P_i has associated register value 1, or P_i has register value 0 and *every* other processor also has register value 0, P_i updates its register to 0 or 1 with equal probability.

Table 1. Forms of guard, with their generic versions, where the associated SP statement has local guard `s1=`i (for some $i \in L$)

guard ::=	generic form $\widehat{\mathit{guard}}$	
(guard)	$(\widehat{\mathit{guard}})$	
$!\,\mathit{guard}$	$!\,\widehat{\mathit{guard}}$	
guard_1 `&` guard_2	$\widehat{\mathit{guard}_1}$ `&` $\widehat{\mathit{guard}_2}$	
guard_1 `|` guard_2	$\widehat{\mathit{guard}_1}$ `|` $\widehat{\mathit{guard}_2}$	
$\forall_i\ \mathtt{s}_i\mathtt{=}j$	$\mathtt{no\_}j\mathtt{=}n$	
$\forall_i\ \mathtt{s}_i\mathtt{!=}j$	$\mathtt{no\_}j\mathtt{=0}$	
$\exists_i\ \mathtt{s}_i\mathtt{=}j$	$\mathtt{no\_}j\ \mathtt{>}\ \mathtt{0}$	
$\exists_i\ \mathtt{s}_i\mathtt{!=}j$	$\mathtt{no\_}j\ \mathtt{<}\ n$	
$\exists_i\ (\mathtt{s}_i\mathtt{=}j\ \mathtt{\&}\ (\forall_{k\neq i}\ \mathtt{s}_k\mathtt{!=}j))$	$\mathtt{no\_}j\mathtt{=1}$	
	$j = i$	$j \neq i$
$\forall_{i>1}\ \mathtt{s}_i\mathtt{=}j$	$\mathtt{no\_}j\mathtt{=}n$	$\mathtt{no\_}j\ \mathtt{=}\ n-1$
$\forall_{i>1}\ \mathtt{s}_i\mathtt{!=}j$	$\mathtt{no\_}j\mathtt{=1}$	$\mathtt{no\_}j\mathtt{=0}$
$\exists_{i>1}\ \mathtt{s}_i\mathtt{=}j$	$\mathtt{no\_}j\ \mathtt{>1}$	$\mathtt{no\_}j\ \mathtt{>0}$
$\exists_{i>1}\ \mathtt{s}_i\mathtt{!=}j$	$\mathtt{no\_}j\ \mathtt{<}\ n$	$\mathtt{no\_}j\ \mathtt{<}\ n-1$

To model all possible initial configurations, in our specification processors start in default state 2, from which they move to state 0 or 1 nondeterministically. The protocol begins once all processors have state 0 or 1. Below we give the SP specification for a system of 3 processors.

```
nondeterministic
module process1
  s1 : [0..2] init 2;
  [] s1=2 -> 1:(s1'=0);
  [] s1=2 -> 1:(s1'=1);
  [] s1=0 & (s1!=2 & s2!=2 & s3!=2) & (s1=0 & s2=0 & s3=0) ->
                               0.5:(s1'=0) + 0.5:(s1'=1);
  [] s1=0 & (s1!=2 & s2!=2 & s3!=2) & (s2=1 | s3=1) -> 1:(s1'=0);
  [] s1=1 & (s1!=2 & s2!=2 & s3!=2) & (s2=1 | s3=1) ->
                               0.5:(s1'=0) + 0.5:(s1'=1);
  [] s1=1 & (s1!=2 & s2!=2 & s3!=2) & (s2=0 & s3=0) -> 1:(s1'=1);
endmodule

module process2 = process1 [ s1 = s2, s2 = s1 ] endmodule
module process3 = process1 [ s1 = s3, s3 = s1 ] endmodule
```

We now show that if $\mathcal{M}(\mathcal{P})$ is the MDP associated with a nondeterministic SP program $\mathcal{P}$ then any permutation of components is an automorphism of $\mathcal{M}(\mathcal{P})$ when lifted to states. Note that if $t \in S$ then $t(i)$ is the value of variable $\mathtt{s}_i$. The proof of the following lemma (which applies to both MDPs and DTMCs) is straightforward, using structural induction on the form of guards given in Table 1.

Lemma 1. *Let σ be a statement in module* `process`i *of an SP program $\mathcal{P}$ ($1 \leq i \leq n$), let t be a state in the associated MDP or DTMC and let $\alpha \in S_n$. If σ' is the corresponding statement in module* `process`$\alpha(i)$ *then σ is executable in $t \Leftrightarrow \sigma'$ is executable in $\alpha(t)$.*

Theorem 3. *Let $\mathcal{P}$ be a nondeterministic SP program. Then $Aut(\mathcal{M}(\mathcal{P})) = S_n$.*

Proof. By definition, $Aut(\mathcal{M}(\mathcal{P})) \subseteq S_n$. Let $\alpha \in S_n$ and $\mu \in Steps(t)$ for some $t \in S$. By the definition of an MDP automorphism (Definition 1), we must show that $\alpha(t) \in S$, and there exists $\mu' \in Steps(\alpha(t))$ such that $\mu(u) = \mu'(\alpha(u))$ for all $u \in S$. Suppose first that $t = t_0$, the initial state of $\mathcal{M}(\mathcal{P})$. Since each variable $\mathtt{s}_i$ is initialised to l for some $0 \leq l \leq k$, $t_0 = (l, l, \ldots, l)$, so clearly $\alpha(t_0) = t_0$.

Now let t be arbitrary in S, and suppose that $\alpha(t) \in S$. The distribution μ arises from the stochastic update of a statement, σ say, in module `process`i, for some $i \in I$, in which the value of $\mathtt{s}_i$ is updated. Module `process`$\alpha(i)$ is a renaming of `process`i where $\mathtt{s}_i$ and $\mathtt{s}_{\alpha(i)}$ are interchanged, so `process`$\alpha(i)$ has a corresponding statement σ' in which $\mathtt{s}_{\alpha(i)}$ is updated. By Lemma 1, σ' is executable in $\alpha(t)$.

Let $\mu' \in Steps(\alpha(t))$ be the probability distribution associated with σ'. Since σ only updates $\mathtt{s}_i$, for any $u \in S$, if $t(j) \neq u(j)$ for some $i \neq j$ then $\mu(u) = 0$. As $\alpha(t)(j) \neq \alpha(u)(j)$, by a similar argument $\mu'(\alpha(u)) = 0$. Now suppose that $t(j) = u(j)$ for all $j \neq i$. Suppose $u(i) = k \in L$. Then $\mu(u)$ is the probability of updating $\mathtt{s}_i$ to k in σ, and $\mu'(u)$ is the probability of updating $\mathtt{s}_{\alpha(i)}$ to k in σ'. Thus $\mu(u) = \mu'(\alpha(u))$.

Finally, for any $t \in S$, we *must* have $\alpha(t) \in S$, since if $t_0, t_1, \ldots, t$ is a path from the initial state to t, by the above argument there is a corresponding path $t_0 = \alpha(t_0), \alpha(t_1), \ldots, \alpha(t)$ from the initial state to $\alpha(t)$.

The proof of the analogous result for probabilistic SP programs is similar.

Theorem 4. *Let $\mathcal{P}$ be a probabilistic SP program. Then $Aut(\mathcal{D}(\mathcal{P})) = S_n$.*

4.1 Symmetric PCTL

The temporal logic $PCTL$ (probabilistic computation tree logic), presented in detail in [25], can be used to specify properties of MDP and DTMC models which can be verified using the PRISM model checker.

We define $SPCTL$, a subset of $PCTL$ for reasoning about SP programs. A *symmetric guard* is a guard of the form described in Table 1 which *does not* include sub-expressions of the last four forms in the table. A symmetric guard g has the property that, if $\alpha \in S_n$, the guard $\alpha(g)$ obtained by replacing $\mathtt{s}_i$ with $\mathtt{s}_\alpha(i)$ is identical to g (modulo order of operands to commutative operators). A $PCTL$ property ϕ is in $SPCTL$ iff every maximal propositional subformula of ϕ is a symmetric guard. Formulas of $SPCTL$ are, by construction, invariant under full symmetry.

For the leader election example, the property

$$P_{\geq 1}\left[\, true\; U\, \texttt{(((s1=1 \& s2!=1 \& s3!=1) | (s2=1 \& s1!=1 \& s3!=1) |} \right. \quad (1)$$
$$\left. \texttt{(s3=1 \& s1!=1 \& s2!=1)) \& (s1!=2 \& s2!=2 \& s3!=2))} \,\right]$$

is in *SPCTL*, and asserts that with probability 1 a leader will eventually be elected.

5 Symmetry Reduction by Counter Abstraction

We now show how a nondeterministic SP program $\mathcal{P}$ can be translated into a reduced program $\widehat{\mathcal{P}}$ and an *SPCTL* formula ϕ into a reduced formula $\widehat{\phi}$. The translation process is polynomial in the size of $\mathcal{P}$. We then show that $\overline{\mathcal{M}(\mathcal{P})}$ and $\mathcal{M}(\widehat{\mathcal{P}})$ are isomorphic, and apply Theorem 2 to show that *SPCTL* properties of $\overline{\mathcal{M}(\mathcal{P})}$ can be inferred by checking reduced properties of $\mathcal{M}(\widehat{\mathcal{P}})$. As $\mathcal{M}(\mathcal{P})$ and $\overline{\mathcal{M}(\mathcal{P})}$ are bisimilar (Theorem 1), *PCTL* properties of the *original* model can be inferred in this way. As $\overline{\mathcal{M}(\mathcal{P})}$ and $\mathcal{M}(\widehat{\mathcal{P}})$ are isomorphic, the state space reduction associated with model checking over $\mathcal{M}(\widehat{\mathcal{P}})$ is the same as that gained by building a quotient structure.

5.1 Translating SP into Generic Form

Let $\mathcal{P}$ be an SP program. The corresponding reduced program $\widehat{\mathcal{P}}$ consists of a single module, `generic_process`. Recall that each module `process`i of $\mathcal{P}$ has a local variable $\mathtt{s}_i$ with domain $L = \{0, 1, \ldots, k\}$, for some $k \geq 0$. The module `generic_process` has k+1 local variables, `no_0`, `no_1`,. . . ,`no_`k, each with domain $I \cup \{0\}$, where `no_`j indicates the number of components of the original system which are in local state $j \in L$. For any $j \in L$, `no_`j is initialised to n if j is the initial value of the variable `s1` in the original program, and to 0 otherwise.

For nondeterministic programs, the `generic_process` module has one statement corresponding to each statement of `process1` in $\mathcal{P}$. Suppose a statement of $\mathcal{P}$ has the following form:

$$\texttt{[] s\_1=}j\texttt{ \& }\mathit{guard}\texttt{ -> }\lambda_1\texttt{:(s1'=}j_1\texttt{) + }\lambda_2\texttt{:(s1'=}j_2\texttt{) + ... + } \lambda_v\texttt{:(s1'=}j_v\texttt{);} \quad (2)$$

where *guard* is a guard of the form specified in Table 1. Then the corresponding statement of `generic_process` is as follows:

$$\begin{aligned}\texttt{[] no\_}j\texttt{>0 \& }\widehat{\mathit{guard}}\texttt{ -> } & \lambda_1\texttt{:(no\_}j\texttt{'=no\_}j\texttt{-1)\&(no\_}j_1\texttt{'=no\_}j_1\texttt{+1)} \\ & \texttt{+ }\lambda_2\texttt{:(no\_}j\texttt{'=no\_}j\texttt{-1)\&(no\_}j_2\texttt{'=no\_}j_2\texttt{+1)} \\ & \vdots \\ & \texttt{+ }\lambda_v\texttt{:(no\_}j\texttt{'=no\_}j\texttt{-1)\&(no\_}j_v\texttt{'=no\_}j_v\texttt{+1);}\end{aligned} \quad (3)$$

with the exception that if one of the j_i in the original stochastic update equals j, the corresponding component of the generic update is `(no_`j_i`'=no_`j_i`)` (otherwise the update would be `(no_`j_i`'=no_`j_i`-1)&(no_`j_i`'=no_`j_i`+1)`, which intuitively should have the same effect, but is not legal in PRISM). The guard $\widehat{guard}$ in $\widehat{\mathcal{P}}$ is the *generic form* of *guard*. Details of the translation of guards are given in the *generic form* column of Table 1. Note that, for the last four forms of guard in the table, the translation to generic form depends on the local guard associated with the statement.

Translation of statements is less straightforward for probabilistic programs, due to the absence of nondeterminism. Let *update* denote the right hand side of `->` in (3). The probability of some module executing their copy of statement (2) in a given state is proportional to the number of modules for which this statement is executable. Thus, in the reduced program, statement (2) is translated to n statements as follows:

$$\begin{array}{r}
\texttt{[] no\_}j\texttt{>0 \& } \widehat{guard} \texttt{ -> } update \\
\texttt{[] no\_}j\texttt{>1 \& } \widehat{guard} \texttt{ -> } update \\
\vdots \\
\texttt{[] no\_}j\texttt{>} n-1 \texttt{ \& } \widehat{guard} \texttt{ -> } update
\end{array}$$

Thus if d modules can execute a statement equivalent to (2) in a given state of the *original* model ($0 \leq d \leq n$), exactly d of the statements above will be executable in the corresponding state of the *reduced* model.

Recall that the states of $\mathcal{M}(\mathcal{P})/\mathcal{D}(\mathcal{P})$ are a subset of L^n. Clearly for the MDP/DTMC $\mathcal{M}(\widehat{\mathcal{P}}) = (\widehat{S}, \widehat{Steps})/\mathcal{D}(\widehat{\mathcal{P}}) = (\widehat{S}, \widehat{P})$, we have $\widehat{S} \subseteq I^{k+1}$.

Below we give the generic form of the leader election example introduced in Section 4.

```
nondeterministic module generic_process
  no_0 : [0..3] init 0;
  no_1 : [0..3] init 0;
  no_2 : [0..3] init 3;
  [] no_2>0 -> 1:(no_2'=no_2-1)&(no_0'=no_0+1);
  [] no_2>0 -> 1:(no_2'=no_2-1)&(no_1'=no_1+1);
  [] no_0>0 & no_2=0 & no_0=3 -> 0.5:(no_0'=no_0) +
                                 0.5:(no_0'=no_0-1)&(no_1'=no_1+1);
  [] no_0>0 & no_2=0 & no_1>0 -> 1:(no_0'=no_0);
  [] no_1>0 & no_2=0 & no_1>1 -> 0.5:(no_1'=no_1-1)&(no_0'=no_0+1)
                                 + 0.5:(no_1'=no_1);
  [] no_1>0 & no_2=0 & no_0=2 -> 1:(no_1'=no_1);
endmodule
```

5.2 Translation of *SPCTL* Properties

Since the states of $\mathcal{M}(\mathcal{P})/\mathcal{D}(\mathcal{P})$ relate to the local states of components, whereas those of $\mathcal{M}(\widehat{\mathcal{P}})/\mathcal{D}(\widehat{\mathcal{P}})$ relate to *how many* components are in each local state, it is necessary to convert an *SPCTL* formula ϕ into a *reduced form*.

Let ϕ be an *SPCTL* formula. Recall from Section 4.1 that the maximal propositional subformulas of ϕ are symmetric guards. The reduced form of ϕ, denoted

$\widehat{\phi}$, is identical to ϕ except that every maximal propositional formula g occurring in ϕ is replaced with $\widehat{g}$, using the translation rules of Table 1.

The generic form of the *election* property (property (1) in Section 4.1) is

$$P_{\geq 1}\;[\;\mathit{true}\,U\,(\texttt{no\_1=1 \& no\_2=0})\;].$$

Note that this concise property is independent of the number of processors participating in the protocol, whereas in the unreduced program, a variant of (1) is required for every protocol configuration.

5.3 Relationship Between $\overline{\mathcal{M}(\mathcal{P})}$ and $\mathcal{M}(\widehat{\mathcal{P}})$

Recall that $\overline{S} = \{min[s] : s \in S\}$. Since components of $\mathcal{P}$ are fully interchangeable, $\overline{S} = \{s \in S : i < j \Rightarrow s(i) \leq s(j)\}$. Then a state $s \in \overline{S}$ has the form

$$s = (\underbrace{0,0,\ldots,0}_{m_0},\underbrace{1,1,\ldots,1}_{m_1},\ldots,\underbrace{k,k,\ldots,k}_{m_k}),$$

where m_i denotes the number of entries equal to i, and $\sum_{i=0}^{k} m_i = n$. With $s \in \overline{S}$ as above, define a mapping $\delta : \overline{S} \to \widehat{S}$ by $\delta(s) = (m_0, m_1, \ldots, m_k)$.

Lemma 2. *The mapping δ is an isomorphism from $\overline{\mathcal{M}(\mathcal{P})}$ to $\mathcal{M}(\widehat{\mathcal{P}})$.*

Let SG be the set of all symmetric guards. The translation rules of Table 1 define a bijection $\widehat{\ } : SG \to SG'$, where SG' is the set of reduced forms of symmetric guards.

Lemma 3. *Let $g \in SG$ be a symmetric guard. Then, for all $s \in \overline{S}$, $s \models g \Leftrightarrow \delta(s) \models \widehat{g}$.*

Proof. Suppose $g =$`(s1=`d` & s2=`d` & ... & s`n`=`d`)` for some $d \in L$. Then $\widehat{g} =$ `no_`d`=`n. Clearly g only holds at the state $t = (d, d, \ldots, d)$, and $\widehat{g}$ only holds at the state

$$\delta(t) = (\underbrace{0,0,\ldots,0}_{d}, n, \underbrace{0,\ldots,0}_{k-(d+1)}).$$

The other base cases are similar, and the result follows by structural induction on the form of symmetric guards.

We can now apply Theorem 2 to deduce:

Theorem 5. *For any $SPCTL$ property ϕ and $s \in \overline{s}$,*

$$\overline{\mathcal{M}}, s \models \phi \Leftrightarrow \mathcal{M}(\widehat{\mathcal{P}}), \delta(s) \models \widehat{\phi}.$$

Further, combining Theorem 1 and Theorem 5 we get

Corollary 1. *For any $SPCTL$ property ϕ and $s \in \overline{S}$,*

$$\mathcal{M}, s \models \phi \Leftrightarrow \overline{\mathcal{M}}, min[s] \models \phi \Leftrightarrow \mathcal{M}(\widehat{\mathcal{P}}), \delta(min[s]) \models \widehat{\phi}.$$

It is thus possible to infer $SPCTL$ properties of $\mathcal{M}(\mathcal{P})$ by checking corresponding reduced properties of $\mathcal{M}(\widehat{\mathcal{P}})$. Analogous results to those in this section hold for probabilistic programs.

6 Experimental Results for Case Studies

We have implemented GRIP (Generic Representatives In PRISM), a Java tool which takes an SP program as input, and outputs the corresponding reduced version. A parser for SP was generated using SableCC [15]. In this section we present experimental results for two case studies – the leader election protocol from [7] which we have used as a running example within this paper, and a probabilistic mutual exclusion protocol adapted from a case study supplied with the PRISM distribution [23], and analysed in [24].

Proving property (1) of Section 4.1 for the leader election example requires the imposition of fairness constraints. It is well known for non-probabilistic model checking that fairness and symmetry reductions cannot be directly combined since the path of an individual process cannot be traced in the quotient structure [10]. Thus for this example we use PRISM to prove that the *maximum* probability of a leader being elected is 1, using the original specification and its generic form. Expressing the mutual exclusion protocol in SP required some straightforward syntactic modifications to the original PRISM code. The property here is that the maximum probability of the critical section becoming clear once occupied approaches 1.

Table 2. Experimental results for various configurations of the leader election (leader) and mutual exclusion (mutex) protocols

	original				reduced			
system	**states**	**nodes**	**build**	**check**	**states**	**nodes**	**build**	**check**
leader 20	3.5×10^{9}	5300	0.4	1	231	1144	0.1	0.04
leader 40	1.2×10^{19}	20240	2	26	861	2563	0.2	0.2
leader 60	4.2×10^{28}	44780	4	109	1891	3735	0.4	0.2
leader 80	1.5×10^{38}	78920	10	669	3321	5706	0.7	0.5
leader 100	5.2×10^{47}	122660	19	2754	5151	7054	1	0.7
leader 120	1.8×10^{57}	176888	30	o/m	7381	8378	1	1
leader 140	6.3×10^{66}	238940	53	o/m	10011	11133	2	2
mutex 4	26600	3591	0.7	0.2	1691	4069	1	0.2
mutex 12	4.9×10^{12}	40687	22	14	892542	25670	5	2
mutex 20	7.1×10^{20}	114647	137	86	3.3×10^{7}	59202	18	7
mutex 28	9.4×10^{28}	225471	552	499	4.2×10^{8}	90381	64	20
mutex 36	1.2×10^{37}	373159	14,003	3262	3.1×10^{9}	138006	322	44
mutex 44	-	-	o/t	-	1.6×10^{10}	175990	604	112
mutex 52	-	-	o/t	-	6.3×10^{10}	214045	1805	162

Table 2 shows, for various configurations of each case study, the number of states (**states**) and MTBDD nodes (**nodes**) for each model. Time taken (in seconds) for model building (**build**) and checking the associated SPCTL property (**check**) are given for each case. Cases where PRISM did not complete model building within 24 hours, and where our memory limit (500 Mb) was exceeded, are denoted by o/t and o/m respectively. All experiments were performed using a PC with a 2.4 GHz Intel Pentium 4 processor, running PRISM version 3.0.beta1 under Red Hat Linux.

Symmetry reduction with generic representatives works particularly well for the leader election example, with significant reductions in both state space and MTBDD sizes, and much shorter times for model building and checking. This is expected, as the approach has been shown to work well in the non-probabilistic case when there are a small number of local states. Here there are 3 local states, and the number of reachable states is reduced from 3^n to $\frac{1}{2}(n+1)(n+2)$ – the theoretical maximum factor of reduction.

Results for the mutual exclusion case study show a saving in MTBDD nodes for larger configurations, but the original model for 4 processes actually requires *fewer* nodes than the generic version (there are 16 process local states so the generic program always uses 16 variables, whereas a configuration with 4 processes only uses 4 variables). It is unsurprising that the benefit of symmetry reduction is not as striking here as there are more local states. Nevertheless, larger configurations exhibit an encouraging reduction in time for both model building and checking, and GRIP enabled us to verify configurations which were previously intractable.

GRIP, together with PERL scripts to generate SP programs and SPCTL properties for both case studies, can be downloaded from our website [16].

7 Related Work

Generic representatives and fully symmetric program transformations were first proposed in [11]. This approach is extended to programs which include global variables [12] and optimised using techniques from compiler optimisation (static reachability analysis and dead variable elimination) in [14], where a prototype generic model checker, *UTOOL* is described. Preliminary results on extending these ideas to probabilistic model checking were presented in [8].

Another approach to combining symmetry reduction with symbolic representation is proposed in [13], where representative states are determined dynamically during fixpoint iterations. This approach has some advantages over using generic representatives (including fewer restrictions on the form of of input programs), but requires significant modifications to a symbolic model checking algorithm. A related approach has been used for symmetry reduction in the PRISM model checker [20]. Here the state space explosion problem is partially avoided: construction of an MTBDD for the full model is still required, but probabilistic temporal properties can be checked over a quotient structure. This approach is useful as, in many cases, it is possible to represent a very large model as an MTBDD, but not to check properties of this model. The problem of combining symmetry reduction with fairness assumptions is discussed in [10], where an automata theoretic approach applicable to explicit state model checking is presented. To our knowledge, the problem of combining symmetry, fairness and symbolic representation has not been investigated.

Other methods for combining symmetry reduction with non-probabilistic symbolic model checking are given in [3,6]. Numerous approaches for exploiting symmetry in non-probabilistic explicit state model checking have been proposed (see [22] for a recent survey), but the application of these techniques to probabilistic explicit state model checking [1] has not been investigated.

8 Conclusions and Future Work

We have shown that an approach to symmetry reduction for non-probabilistic symbolic model checking, based on generic representatives, can be applied in the probabilistic setting. Our techniques are applicable to symmetric PRISM programs with MDP or DTMC semantics, and the translation of an SP program to its reduced form is implemented by the GRIP tool. Experimental results for two protocol case studies – minimum space leader election and mutual exclusion – show that the technique can be effective.

Future work includes extending the approach to allow model checking of CSL properties over continuous time Markov chain models, and using techniques proposed in [14] to allow a less restrictive input language for symmetric programs. It will also be useful to carry out an experimental comparison with alternative symmetry reduction techniques for PRISM [20] based on dynamic symmetry reduction [13].

Acknowledgements. The authors would like to thank Douglas Graham for many useful discussions relating to this work, and Dave Parker for providing an advance copy of [20].

References

1. C. Baier, F. Ciesinski and M. Größer. ProbMela and verification of Markov decision processes. *SIGMETRICS Performance Evaluation Review*, 32(4): 22-27, 2005.
2. C. Baier, M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11:125–155, 1998.
3. S. Barner and O. Grumberg. Combining symmetry reduction and under-approximation for symbolic model checking. *Formal Methods in System Design*, 27(1–2):29–66, 2005.
4. D. Bosnacki, D. Dams, and L. Holenderski. Symmetric spin. *International Journal on Software Tools for Technology Transfer*, 4(1):65–80, 2002.
5. E.M. Clarke, E.A. Emerson, S. Jha, and A.P. Sistla. Symmetry reductions in model checking. In *CAV'98*, LNCS 1427, pages 147–158. Springer, 1998.
6. E.M. Clarke, R. Enders, T. Filkhorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1–2):77–104, 1996.
7. S. Dolev, A. Israeli and S. Moran. Analysing expected time by scheduler-luck games. *IEEE Transactions on Software Engineering*, 21(5):429–439, 1995.
8. A.F. Donaldson and A. Miller Symmetry reduction for probabilistic systems. In *Proc. 12th Workshop on Automated Reasoning*, pages 17–18, 2005.

9. A.F. Donaldson and A. Miller Exact and approximate strategies for symmetry reduction in model checking. In *FM'06*, LNCS 4085, pages 541–556. Springer, 2006.
10. E.A. Emerson and A.P. Sistla. Utilizing symmetry when model-checking under fairness assumptions: an automata-theoretic approach. *ACM Transactions on Programming Languages and Systems*, 19(4):617–638, 1997.
11. E.A. Emerson and R.J. Trefler. From asymmetry to full symmetry: new techniques for symmetry reduction in model checking. In *CHARME'99*, LNCS 1703, pages 142–156. Springer, 1999.
12. E.A. Emerson and T. Wahl. On combining symmetry reduction and symbolic representation for efficient model checking. In *CHARME'03*, LNCS 2860, pages 216–230. Springer, 2003.
13. E.A. Emerson and T. Wahl. Dynamic symmetry reduction. In *TACAS'05*, LNCS 3440, pages 382–396. Springer, 2005.
14. E.A. Emerson and T. Wahl. Efficient reduction techniques for systems with many components. *Electronic Notes in Theoretical Computer Science*, 130:379–399, 2005.
15. E. Gagnon and L. J. Hendren. SableCC, an object-oriented compiler framework. In *TOOLS'98*, pages 140–154. IEEE Computer Society Press, 1998.
16. GRIP website. http://www.dcs.gla.ac.uk/people/personal/ally/grip/.
17. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(4):512–535, 1994.
18. A. Hinton, M. Kwiatkowska, G. Norman and D. Parker. PRISM: a tool for automatic verification of probabilistic systems. In *TACAS'06*, LNCS 3920, pages 441–444. Springer, 2006.
19. C.N. Ip and D.L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1/2): 41–75, 1996.
20. M. Kwiatkowska, G. Norman and D. Parker. Symmetry reduction for probabilistic model checking. To appear in *CAV'06*, LNCS. Springer, 2006.
21. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94: 1–28, 1991.
22. A. Miller, A. Donaldson and M. Calder. Symmetry in temporal logic model checking. To appear in *Computing Surveys*, 2006.
23. PRISM website. http://www.cs.bham.ac.uk/∼dxp/prism/.
24. A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.
25. J.J.M.M. Rutten, M. Kwiatkowska, G. Norman and D. Parker. Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems. *CRM Monograph Series* 23. American Mathematical Society 2004.
26. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.