Language-level Symmetry Reduction for Probabilistic Model Checking

Alastair F. Donaldson
Computing Laboratory
Oxford University
Oxford, UK
Email: alastair.donaldson@comlab.ox.ac.uk

Alice Miller
Department of Computing Science
University of Glasgow
Glasgow, UK
Email: alice@dcs.gla.ac.uk

David Parker
Computing Laboratory
Oxford University
Oxford, UK
Email: david.parker@comlab.ox.ac.uk

***Abstract*—Symmetry reduction is a technique for combating state-space explosion in model checking. The generic representatives approach to symmetry reduction uses a language-level translation of symmetric models to a reduced form, making it straightforward to combine with existing tools and implementations. These techniques have been proposed for both non-probabilistic and probabilistic model checking, but are currently difficult to apply to complex models due to prohibitive restrictions in the modelling language. We present a much richer language, which allows specification of probabilistic systems in a way that guarantees the applicability of the generic representatives technique, together with an extended translation algorithm, and demonstrate the effectiveness of our techniques on a large set of case studies.**

***Keywords*-Symmetry reduction; probabilistic model checking; counter abstraction**

I. Introduction

Probabilistic model checking provides a powerful set of techniques for formally verifying quantitative properties of systems that exhibit stochastic behaviour. As with any exhaustive approach to formal verification, when applying these methods to real-life systems it quickly becomes essential to use techniques that improve the efficiency of the process. For probabilistic model checking, the need for such techniques is particularly acute since it requires not only an exhaustive state-space exploration, but also a numerical solution phase to compute probabilities or other quantitative values. Possible approaches include generating a compact model representation, *e.g.* with state compaction or symbolic (BDD-based) data structures, and reducing the size of the model to be analysed, *e.g.* via abstraction, bisimulation minimisation, partial order or symmetry reduction.

In this paper, we consider *symmetry reduction* [1], [2], [3], [4], which exploits the presence of replication in a system. Symmetry is in fact prevalent in many of the application domains to which probabilistic model checking has already proved valuable, *e.g.* randomized distributed algorithms, communication protocols and biological systems.

Symmetry reduction via *generic representatives* [5], [6] combines the benefits of symmetry reduction and symbolic representation. A system specification is translated into a reduced form with a set of counters that represent the behaviour of processes *generically*, keeping track how many processes reside in each local state, but not of individual process identities. The semantics of the generic specification is isomorphic to the symmetry-reduced model associated with the original specification. In most cases, the new model is not only significantly smaller, but also has a more compact symbolic representation. This is particularly beneficial in the context of probabilistic model checking since, for many state-of-the-art implementations (including the PRISM tool, to which we apply our techniques), both model size (number of states) *and* the size of a symbolic representation are critical. Initial results applying symmetry reduction by generic representatives to probabilistic model checking indicate that this is indeed the case [7], [8].

Although effective for certain examples, the original generic representatives technique has limited application: it can only be applied to a single family of identical processes (or one with an additional singleton process [6]) where the local state of each process is modelled by a single variable.

In this paper we significantly extend the application of generic representatives. We expand the theory to specifications consisting of multiple families of identical processes, whose state can be represented by multiple local variables, and whose behaviour can be described by guarded commands involving the kind of complex expressions over local and global variables required to model realistic systems.

We present our results by introducing Symmetric Probabilistic Specification Language (SPSL), a modelling language for symmetric systems with shared-variable communication, designed to guarantee applicability of the generic representatives approach. We focus on a Markov decision process (MDP) semantics for SPSL. Since MDPs generalise Kripke structures, SPSL also represents a significant improvement to the applicability of generic representatives to non-probabilistic model checking.

We present an algorithm for translating SPSL specifications to generic form, such that the symmetry-reduced semantics can be analysed using the PRISM model checker [9]. We have implemented the translation in a tool, GRIP, and have applied it to a wide range of case studies. We provide a comprehensive experimental evaluation, illustrating that our techniques provide excellent improvements in efficiency, both in terms of the feasibility of verifying large

models and the time required for model construction and analysis. We also study the relative merits of a complementary symmetry reduction approach for PRISM [10].

II. Symmetry reduction for MDPs

We review some background material on Markov decision processes (MDPs), symmetry reduction and, in particular, generic representatives.

A. Markov decision processes

We use $Dist(S)$ to denote the set of discrete probability distributions over a set S.

Definition 1: A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, s_0, Steps)$, where S is a finite set of states, $s_0 \in S$ an initial state, and $Steps : S \rightarrow 2^{Dist(S)}$ a probabilistic transition function.

An MDP $\mathcal{M}$ represents a system which exhibits both nondeterministic and probabilistic behaviour (for example, the asynchronous parallel composition of a set of randomized processes). Intuitively, in each state $s \in S$, there is a nondeterministic choice between the elements of $Steps(s)$. The next state of the model to which a transition will occur is then chosen probabilistically according to the selected distribution $\mu \in Steps(s)$. A path through an MDP is obtained by resolving both probabilistic and non-deterministic choices. To do this one assumes that non-deterministic choices are resolved by an *adversary* which selects a choice based on the history of choices so far. A probability measure is defined that allows one to calculate the maximal and minimal probability for a set of paths.

B. Automorphisms and quotient MDPs

Definition 2: Let $\mathcal{M} = (S, s_0, Steps)$ and $\mathcal{M}' = (S', s'_0, Steps')$ be MDPs, and $\alpha : S \rightarrow S'$ a bijection. Suppose $\alpha(s_0) = s'_0$, and for $s \in S$, $\mu \in Steps(s)$ iff there exists $\mu' \in Steps'(\alpha(s))$ 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*.

Definition 3: An *automorphism* of $\mathcal{M}$ is an isomorphism from $\mathcal{M}$ to $\mathcal{M}$. The set of all automorphisms of $\mathcal{M}$ forms a group under composition of mappings, denoted $Aut(\mathcal{M})$.

Definition 4: Let $G \leq Aut(\mathcal{M})$. The *orbit relation* for G is the set $\theta = \{(s, \alpha(s)) : s \in S, \alpha \in G\} \subseteq S \times S$. For $s \in S$, the *orbit* of s under G is the set $[s]_G = \{t : (s,t) \in \theta\}$. If G is clear from the context, we write $[s]$ rather than $[s]_G$.

Definition 5: Suppose that we have a total ordering for S and let $\min[s]$ denote the smallest element of $[s]$ for any state s. For any $G \leq Aut(\mathcal{M})$, the *quotient MDP* for $\mathcal{M}$ *w.r.t.* G is the MDP $\overline{\mathcal{M}} = (\overline{S}, \overline{s_0}, \overline{Steps})$ where:

- $\overline{S} = \{\min[s] : s \in S\}$ and $\overline{s_0} = \min[s_0] = s_0$
- for each $\min[s] \in \overline{S}$ and $\mu \in Steps(\min[s])$, $\overline{Steps}(\min[s])$ contains a distribution $\overline{\mu} \in Dist(\overline{S})$ where, for $\min[t] \in \overline{S}$, $\overline{\mu}(\min[t]) = \sum_{x \in [t]} \mu(x)$.

We follow the widely used convention of choosing $\min[s]$ as a unique representative of $[s]$.

Let A be a set which we call *atoms*. In our application domain, an atom is a boolean expression over variables of a concurrent system. The logic $PCTL$ (probabilistic computation tree logic) [11], [12] is defined as a set of state and path formulae over a set of atoms A. If s is a state of MDP $\mathcal{M}$ and $a \in A$ is an atom, then $\mathcal{M}, s \models a$ if a evaluates to *true* at s. If $\bowtie \in \{<, \leq, >, \geq\}$, $p \in [0,1]$ and ψ is a path formula, then $\mathcal{M}, s \models P_{\bowtie p}[\psi]$ if the probability q of a path from s satisfying ψ is such that $q \bowtie p$ for all adversaries. If s_0 is the initial state of $\mathcal{M}$ and ϕ is a $PCTL$ formula, we write $\mathcal{M} \models \phi$ to denote that $\mathcal{M}, s_0 \models \phi$. For full details on $PCTL$, see [11], [12].

The following theorem, proved in [13], establishes a correspondence between $PCTL$ properties of isomorphic MDPs under an appropriate transformation of atoms.

Theorem 1: Let $\mathcal{M} = (S, s_0, Steps)$ and $\mathcal{M}' = (S', s'_0, Steps')$ be MDPs, A and A' sets of atoms, $\gamma : A \rightarrow A'$ a bijection, and δ an isomorphism from $\mathcal{M}$ to $\mathcal{M}'$ such that, for every $s \in S$ and $a \in A$, $\mathcal{M}, s \models a \Leftrightarrow \mathcal{M}', \delta(s) \models \gamma(a)$. Then for any $PCTL$ formula ϕ over A and $s \in S$,

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

where $\gamma(\phi)$ is the $PCTL$ formula over A' obtained by replacing every atom a occurring in ϕ with $\gamma(a)$.

For $G \leq Aut(\mathcal{M})$ and ϕ a $PCTL$ formula, we say that ϕ is *symmetric* with respect to G if, for every maximal propositional sub-formula f appearing in ϕ and for any state $s \in S$, we have $\mathcal{M}, s \models f \Rightarrow \bigwedge_{s' \in [s]} \mathcal{M}, s' \models f$ [2].

The following theorem, proved in [13], shows that if ϕ is a $PCTL$ formula which is symmetric with respect to a group of MDP automorphisms then we can check whether ϕ holds for $\mathcal{M}$ by considering the quotient MDP $\overline{\mathcal{M}}$ (which may be considerably smaller than $\mathcal{M}$).

Theorem 2: Let ϕ be a $PCTL$ formula which is symmetric *w.r.t.* $G \leq Aut(\mathcal{M})$. Let $\overline{\mathcal{M}}$ be the quotient MDP for $\mathcal{M}$ *w.r.t.* G. Then $\mathcal{M} \models \phi \Leftrightarrow \overline{\mathcal{M}} \models \phi$.

For any $s \in S$, the size of $[s]$ is bounded by $|G|$, so the theoretical minimum size of $\overline{S}$ is $|S|/|G|$. For highly symmetric systems with n components we may have $|G| = n!$, in which case exploiting symmetry potentially offers a considerable reduction in memory requirements.

C. Generic representatives

In the context of symbolic model checking, construction and representation of the orbit relation can be very costly. The *generic representatives* method of symmetry reduction

avoids construction of the orbit relation by applying symmetry at the language level via a source-to-source translation [5], [6], and thus requires no modification of existing model checking algorithms.

The idea is best explained using an example. Consider a mutual exclusion algorithm for three identical processes, each with 3 local states *neutral* (N), *trying* (T) and *critical* (C). The global states $(N, N, T), (T, N, N)$ and (N, T, N) are symmetrically equivalent and have generic representative $(2N, 1T)$. A generic representative indicates *how many* processes are in each local state, without referring to individual processes.

The technique involves translating a fully symmetric specification $\mathcal{P}$, consisting of multiple instances of a single process type, into a generic form $\mathfrak{h}(\mathcal{P})$ where processes are replaced by *counters*. In $\mathfrak{h}(\mathcal{P})$ a counter variable is defined for each process local state indicating the number of processes currently in that state. Translation rules ensure that the model $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$ for $\mathfrak{h}(\mathcal{P})$ is isomorphic to the quotient model $\overline{\mathcal{M}(\mathcal{P})}$ for $\mathcal{P}$ [6]. Thus the state-space reduction potentially available by model checking $\overline{\mathcal{M}(\mathcal{P})}$ can be realised by instead model checking $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$. Theoretical and experimental results show that for systems consisting of many processes, where each process has a small local state-space, the BDD representation of $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$ can be significantly smaller than the BDD for $\mathcal{M}(\mathcal{P})$ or $\overline{\mathcal{M}(\mathcal{P})}$. The approach is extended to provide symmetry reduction for MDPs in [7].

As discussed in Section I, a major drawback to current generic representatives techniques is the fact that specifications must be expressed in an extremely restricted input language before translation to generic form can be performed. Here we significantly extend the approach to work with a richer input language supporting multiple families of symmetric processes. The state of each process is determined by the values of multiple local variables, and sophisticated process behaviour can be specified via arithmetic and boolean expressions over local and global variables.

III. Symmetric Probabilistic Specification Language

We define a simple but general language for specifying symmetric, probabilistic systems, which we call Symmetric Probabilistic Specification Language (SPSL). This is loosely based on Symmetric PRISM [7], a subset of the model specification language for PRISM [9]. The connection with PRISM is beneficial for several reasons. Firstly, other tools make direct use of the PRISM modelling language or accept inputs that can be generated by PRISM. Secondly, translations have been developed to the PRISM language from a variety of specification formalisms (including probabilistic variants of several process calculi).

A. Syntax

The syntax for SPSL is presented via a BNF-like grammar in Fig. 1. Note that $\bowtie$ is a binary operator in $\{+, -, *, /, **, <, \leq, >, \geq, =, \neq, \wedge, \vee\}$, where ** denotes exponentiation; $\bigcirc$ is an operator in $\{\sum, \prod, \bigwedge, \bigvee\}$; number denotes a numeric literal; name is the name of a variable, and local-name/global-name is the name of a local/global variable specifically.

An SPSL specification consists of an optional set of global variable declarations followed by a set of module type declarations, each representing a particular class of symmetric processes.

A *variable declaration* consists of a name, type and initial value. For simplicity, we use a basic type system in which variables are either booleans or finite integer ranges. The domain and initial value of a variable v are denoted by $dom(v)$ and $init(v)$, respectively.

A *module type* declaration consists of a name (M), an integer specifying the number of running instances of M (denoted $\#M$), a set of local variable declarations and a set of statements. The set of all module type names is denoted $\mathbf{M}$ and we refer to the running instances of $M \in \mathbf{M}$ as $M_1, M_2, \ldots, M_{\#M}$.

The set of global variables is denoted $global$ and the variables local to module type M are denoted $var(M)$. For simplicity, but without loss of generality, we assume that all variable names and module type names in a specification are distinct. For variable $v \in var(M)$, we write v_i for the corresponding local variable of module instance M_i. We use $var_t(M)$ to denote the t-th variable in the specification of M, and $var_t(M)_i$ the copy belonging to M_i.

The behaviour of instances of module type M is given by a set of statements defined with respect to a symbolic module instance M_i; the behaviour of a concrete instance M_z (for any $1 \leq z \leq \#M$) is derived from these statements by replacing each symbolic variable v_i with the corresponding concrete variable v_z. Statements are guarded commands: the *guard* is an expression determining whether a statement is executable in a given state; the effect of the statement is defined by a *stochastic update*, which takes the form $e^1{:}u^1{+}e^2{:}u^2{+}\ldots{+}e^k{:}u^k$ where the e^j are expressions (giving probability) and the u^j updates. Each update u^j is either a no-op (`skip`), or a set of concurrent updates (denoted by $\|$) to distinct variables that are either global or local to M.

Guards, probabilities and updates to variables in module type M are given by expressions derived from the production rule expr(M_i). This is defined in terms of two other rules: loc-expr(M), which permits expressions over variables from $var(M)$, and symm-expr, which defines *fully symmetric* expressions composed of globals, constants and combinations of a local expression instantiated over *all* modules of a given type. Thus expr(M_i) defines expressions which are *symmetric with respect to* M_i. These are composed of fully symmetric expressions, combinations of a local expression

specification	::=	global-variables$^?$ module$^+$
global-variables	::=	`globals` { var-decl$^+$ }
module	::=	`module` M[number]{ var-decl* statement$(M)^+$ }
var-decl	::=	name : type `init` constant
type	::=	[number..number] | `bool`
constant	::=	`true` | `false` | number
statement(M)	::=	expr$(M_i) \rightarrow$ st-update(M)
st-update(M)	::=	expr(M_i):update(M) + ... + expr(M_i):update(M)
update(M)	::=	`skip` | (name := expr(M_i)) $\|\| \ldots \|\|$ (name := expr(M_i))
symm-expr	::=	constant | global-name
	|	$\bigcirc_{1 \leq j \leq \#N}$loc-expr$(N)_j$ (for some module type N)
	|	symm-expr $\bowtie$ symm-expr
	|	$\neg$symm-expr | (symm-expr)
loc-expr(M)	::=	constant | local-name
	|	loc-expr$(M) \bowtie$ loc-expr(M)
	|	$\neg$loc-expr(M)
	|	(loc-expr(M))
expr(M_i)	::=	loc-expr$(M)_i$ | symm-expr
	|	$\bigcirc_{1 \leq j \neq i \leq \#M}$loc-expr$(M)_j$
	|	expr$(M_i) \bowtie$ expr(M_i)
	|	$\neg$expr(M_i) | (expr(M_i))

Figure 1. Syntax of Symmetric Probabilistic Specification Language (SPSL).

instantiated over all modules of type M *except* M_i and local expressions of the form loc-expr$(M)_i$.

The symmetric conjunctions and disjunctions of SPSL generalise simple symmetric boolean expressions provided by previous approaches [5], [6]. Additionally, SPSL allows symmetric summations and products, and allows expressions to operate on a mixture of local and global variables. This affords a great deal more flexibility when modelling.

B. MDP semantics

We now define semantics for an SPSL specification $\mathcal{P}$ in terms of an MDP $\mathcal{M}(\mathcal{P}) = (S, s_0, Steps)$. For brevity, we do not consider errors such as division-by-zero or probability distributions which do not sum to one; it is easy to extend the semantics to handle this via a designated error state.

The local state-space for a single module M_i of type M is $S(M_i) = \bigotimes_{v \in var(M)} dom(v)$ and the state-space for the global variables of $\mathcal{P}$ is $G = \bigotimes_{v \in global} dom(v)$. The global state-space of $\mathcal{M}(\mathcal{P})$ is:

$$S = G \times \Big(\bigotimes_{M \in \mathbf{M}} \Big(\bigotimes_{1 \leq i \leq \#M} S(M_i) \Big) \Big)$$

Let e be an expression of the form expr(M_i), and s a (local or global) state. Then $eval(s, e)$ is the result of evaluating the expression e at s. If e is a boolean expression, we use $s \models e$ as shorthand for $eval(s, e) = true$.

Let $s \in S$ and u an update of the form $(v^1 := e^1) \| (v^2 := e^2) \| \ldots \| (v^k := e^k)$, where the v^j are distinct variables. Then $exec(s, u)$ is the state identical to s except that the value of each variable v_j is equal to $eval(s, e^j)$. We also define $exec(s, \texttt{skip}) = s$.

We now define the probabilistic transition function $Steps$. Let $s \in S$, $M \in \mathbf{M}$ and $1 \leq z \leq \#M$. Let $e_z \rightarrow u_z$ be a concrete statement of module M_z. Suppose u_z has the form $e^1{:}u^1 + e^2{:}u^2 + \ldots + e^k{:}u^k$. If $s \models e_z$ then $\mu \in Steps(s)$, where $\mu : S \rightarrow [0, 1]$ is defined as follows: for $t \in S$, $\mu(t) = \sum_{1 \leq j \leq k, exec(s, u^j) = t} eval(s, e^j)$.

C. Symmetric PCTL

Let $\mathcal{P}$ be an SPSL specification, and A the set of all expressions of the form symm-expr defined over the variables of $\mathcal{P}$. Taking A to be our set of atoms, we refer to the set of $PCTL$ formulae over A as *symmetric PCTL* ($SPCTL$). The $SPCTL$ logic allows us to reason about properties of SPSL specifications which do not distinguish between individual modules within a given module type.

D. Running example

To illustrate the SPSL language and associated techniques, we introduce a running example: a simple SPSL specification, describing 3 *master* processes and a pool of 10 *worker* processes. A master probabilistically issues requests to the workers via a shared channel `m_to_w`, modelled abstractly as a counter, becoming inactive only when no worker processes remain active. A worker process continuously removes requests from the shared channel by decrementing the counter. When not processing work, a worker process may go to sleep with probability 0.1.

Fig. 2 shows the master/worker SPSL specification. The labels (1) to (7) are used for explanation in Section IV-C and should otherwise be ignored.

The following $SPCTL$ property states that the probability of reaching a state where all workers are asleep and there are outstanding requests on the shared channel is less than 0.01: $P_{<0.01}[\, \Diamond\, (\texttt{m\_to\_w} > 0 \ \wedge \ \bigwedge_{1 \leq j \leq 10} \texttt{awake}_j = 0)]$.

E. SPSL Specifications are Symmetric

SPSL has been designed to guarantee full symmetry between instances of each module type. Formally, this can be stated as follows. For a module type M of $\mathcal{P}$, let $Sym(M)$ denote the group of permutations of the set $\{M_1, M_2, \ldots, M_{\#M}\}$. Elements of $Sym(M)$ act on a state s of $\mathcal{M}(\mathcal{P})$ in the obvious way. Suppose that $\mathbf{M} = \{M^1, M^2, \ldots, M^d\}$. Then $Sym(\mathbf{M}) = \{\alpha^1 \alpha^2 \ldots \alpha^d : \alpha^j \in Sym(M^j)\ (1 \leq j \leq d)\}$ is the largest group of module permutations that preserves module types. The fully symmetric

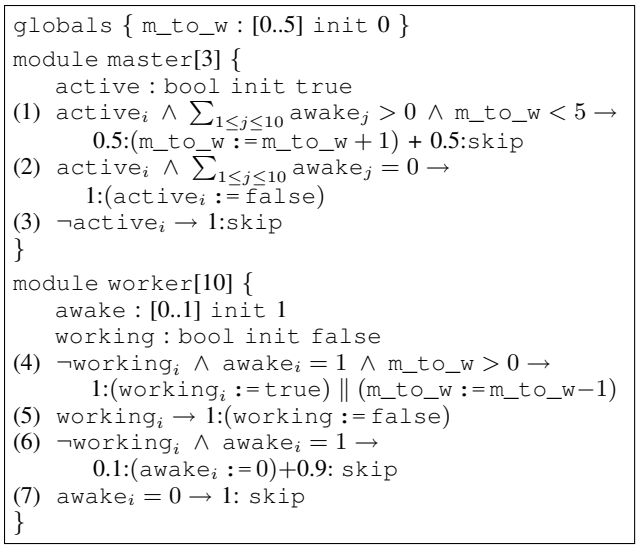

Figure 2. Example SPSL specification for the *master/worker* system described in Section III-D.

syntax of SPSL guarantees that every element of $Sym(\mathbf{M})$ is an automorphism of $\mathcal{M}(\mathcal{P})$, and any $SPCTL$ formula is symmetric *w.r.t.* $Sym(\mathbf{M})$. This, with Theorem 2, leads to the following result:

Theorem 3: Let $\overline{\mathcal{M}(\mathcal{P})}$ denote the quotient MDP for $\mathcal{M}(\mathcal{P})$ *w.r.t.* $Sym(\mathbf{M})$ and let ϕ be an $SPCTL$ formula. Then $\mathcal{M}(\mathcal{P}) \models \phi \Leftrightarrow \overline{\mathcal{M}(\mathcal{P})} \models \phi$.

Suppose the module types for $\mathcal{P}$ are $M^1, M^2, \ldots, M^d$ for some $d > 0$. It is easy to show that $|Sym(\mathbf{M})| = |\#M^1|! \times |\#M^2|! \times \cdots \times |\#M^d|!$. Since the extent of symmetry reduction depends on the size of the symmetry group (see Section II), there is potential for a large reduction factor with multiple instances of each module type.

IV. Translation to Generic Form

We now show how an SPSL specification $\mathcal{P}$ together with an $SPCTL$ property ϕ can be translated into a *generic specification* $\mathfrak{h}(\mathcal{P})$ and property $\mathfrak{h}(\phi)$, such that $\mathcal{M}(\mathcal{P}) \models \phi \Leftrightarrow \mathcal{M}(\mathfrak{h}(\mathcal{P})) \models \mathfrak{h}(\phi)$.

In this section, we impose a restriction on $\mathcal{P}$, requiring that in a an update $v_i := e$ to a *local* variable, expression e contains only local variables and constants.

A. The translation algorithm

Fig. 3 presents a set of syntax-directed translation rules, based on the SPSL grammar of Fig. 1, for translation of an SPSL specification $\mathcal{P}$ into a *generic* SPSL specification, $\mathfrak{h}(\mathcal{P})$. We call $\mathfrak{h}$ the *translation function*.

We introduce some additional notation. Let $S(M)$ denote the local state-space for some module type M (which is the same for any instance of M) and let $|S(M)| = t$. Using the natural lexicographic ordering over tuples in $S(M)$, we define a bijection $f_M : S(M) \to \{1, 2, \ldots, t\}$. For an expression e, $\mathsf{SAT}_M(e) = \{l \in S(M) : l \models e\}$ denotes the subset of local states of M in which e holds.

The first two rules in Fig. 3 state that the global variables of $\mathfrak{h}(\mathcal{P})$ are the same as for $\mathcal{P}$ and that, for every module type declaration M in $\mathcal{P}$, $\mathfrak{h}(\mathcal{P})$ contains a single instance of a module of type `generic_`M. This module has $|S(M)|$ *counter* variables with range $0, \ldots, \#M$. Each variable `count_`M`_`j records the number of instances of M residing in state $f_M^{-1}(j)$ (for $1 \leq j \leq t$). We initialise `count_`M`_`$f_M(init(M))$ to $\#M$ and all others to 0.

When translating statements for a module type declaration M, we must consider the fact that these statements comprise not only symmetric expressions, but also expressions referring to the local state of M (*i.e.* those from the grammar production $\mathsf{loc\text{-}expr}(M)$ in Fig. 1). For the purposes of translation, we assume that the guard of a statement is of the form $e_i \wedge \mathsf{expr}(M_i)$, where e has the form $\mathsf{loc\text{-}expr}(M)$. We then split the translation into cases, one for each $l \in \mathsf{SAT}_M(e)$. This loses no generality since, in the worst case, a guard of the general form $\mathsf{expr}(M_i)$ is trivially of the form given above, by taking e_i to be $true$. In this case $|\mathsf{SAT}_M(e)| = |S(M)|$, *i.e.* each local state of M is treated separately, meaning that the worst case complexity for our translation algorithm is $O(|\mathcal{P}| \times \max_{M \in \mathbf{M}} |S(M)|)$ where $|\mathcal{P}|$ is the number of statements in $\mathcal{P}$. In practice, guarded-command style descriptions of multi-process systems are very often written with guards of the form $e_i \wedge \mathsf{expr}(M_i)$ where e_i is reasonably strong, so the worst case complexity is rarely realised.

Given a statement x of the form $e_i \wedge \mathsf{expr}(M_i) \to \mathsf{stoch\text{-}update}(M)$, we generate, for each $l \in \mathsf{SAT}_M(e)$ a separate generic statement corresponding to the original statement x as follows. The e_i part of the guard is translated into the condition `count_`M`_`$f_M(l) > 0$. This condition asserts that *some* instance of M has local state l and thus the local part of the guard is satisfied for this module. The remainder of the guard, of the form $\mathsf{expr}(M_i)$, and the stochastic update $\mathsf{stoch\text{-}update}(M)$, are translated *in the context of* l using the rules $\mathfrak{h}(\mathsf{expr}(M_i), l)$ and $\mathfrak{h}(\mathsf{stoch\text{-}update}(M), l)$.

The most intricate part of the translation process is the translation of variable updates. Consider an update:

$$(v_i^1 := e_i^1) \| \ldots \| (v_i^t := e_i^t) \| (g^1 := d^1) \| \ldots \| (g^r := d^r)$$

where $v^j \in var(M)$, e^j (according to our restriction) has the form $\mathsf{loc\text{-}expr}(M)$ $(1 \leq j \leq t)$, $g^j \in global$ and d^j has the form $\mathsf{expr}(M_i)$ $(1 \leq j \leq r)$.

Let l' be the local state reached by executing all local variable updates (which can be statically computed). The update $\mathfrak{h}(u, l)$ (see Fig. 3) has the effect of: decrementing `count_`M`_`$f_M(l)$ (representing a process leaving state l), incrementing `count_`M`_`$f_M(l')$ (representing this process entering state l'), and updating the value of each global variable g^j according to the generic expression $\mathfrak{h}(d^k, l)$

$\mathcal{P}$	$\mathfrak{h}(\mathcal{P})$
global-variables module ... module	global-variables $\mathfrak{h}$(module) ... $\mathfrak{h}$(module)
module	$\mathfrak{h}$(module), with $m = init(M)$ and $t = \|S(M)\|$

`module` $M[k]$ {
var-decl*
statement(M) ... statement(M)
}
	`module generic_`$M[1]$ {
	`count_`M`_1 : [0..`k`] init 0`
	... `count_`M`_`$f_M(m)$ `: [0..`k`] init` k ...
	`count_`M`_`t `: [0..`k`] init 0`
	$\mathfrak{h}$(statement(M)) ... $\mathfrak{h}$(statement(M))
	}

statement(M), where e has form local-expr(M)	$\mathfrak{h}$(statement(M)), with $\mathsf{SAT}_M(e) = \{l^1, \ldots, l^z\}$
$e_i \wedge$ expr(M_i) $\rightarrow$ stoch-update(M)	`count_`M`_`$f_M(l^1) > 0 \wedge \mathfrak{h}$(expr($M_i$), l^1) $\rightarrow \mathfrak{h}$(stoch-update(M), l^1) ... `count_`M`_`$f_M(l^z) > 0 \wedge \mathfrak{h}$(expr($M_i$), l^z) $\rightarrow \mathfrak{h}$(stoch-update(M), l^z)
stoch-update(M)	$\mathfrak{h}$(stoch-update(M), l)
expr(M_i):update(M) + ... +expr(M_i):update(M)	$\mathfrak{h}$(expr(M_i), l):$\mathfrak{h}$(update(M), l) + ... +$\mathfrak{h}$(expr(M_i), l):$\mathfrak{h}$(update(M), l)
update(M), where $v^j \in var(M)$, $g^j \in global$ and e^j/d^j has form local-expr(M)/expr(M_i)	$\mathfrak{h}$(update(M), l), where $l' = l[v^1 := eval(l, e^1), \ldots, v^t := eval(l, e^t)]$
`skip` $(v_i^1 := e_i^1) \| \ldots \| (v_i^t := e_i^t)$ $\|(g^1 := d^1) \| \ldots \| (g^r := d^r)$	`skip` (`count_`M`_`$f_M(l)$ `:=count_`M`_`$f_M(l) - 1$) $\|$(`count_`M`_`$f_M(l')$ `:=count_`M`_`$f_M(l') + 1$) $\|(g^1 := \mathfrak{h}(d^1, l)) \| \ldots \| (g^r := \mathfrak{h}(d^r, 1))$
expr(M_i), where e has form local-expr(M)	$\mathfrak{h}$(expr(M_i), l)
e_i	$eval(l, e)$
symm-expr	$\mathfrak{h}$(symm-expr)
$\sum_{1 \le j \ne i \le \#M} e_j$	$\sum_{m \in S(M)} (eval(m, e) * $ `count_`M`_`$f_M(m)) - eval(l, e)$
$\prod_{1 \le j \ne i \le \#M} e_j$	$\prod_{m \in S(M)} (eval(m, e)$**`count_`$M$`_`$f_M(m)) / eval(l, e)$
$\bigwedge_{1 \le j \ne i \le \#M} e_j$	$\sum_{m \in \mathsf{SAT}_M(e)}$ `count_`M`_`$f_M(m) = \#M$ (if $l \models e$)
	$\sum_{m \in \mathsf{SAT}_M(e)}$ `count_`M`_`$f_M(m) = \#M - 1$ (if $l \not\models e$)
$\bigvee_{1 \le j \ne i \le \#M} e_j$	$\sum_{m \in \mathsf{SAT}_M(e)}$ `count_`M`_`$f_M(m) > 0$ (if $l \not\models e$)
	$\sum_{m \in \mathsf{SAT}_M(e)}$ `count_`M`_`$f_M(m) > 1$ (if $l \models e$)
expr(M_i) $\bowtie$ expr(M_i)	$\mathfrak{h}$(expr(M_i), l) $\bowtie \mathfrak{h}$(expr(M_i), l)
$\neg$expr(M_i)	$\neg \mathfrak{h}$(expr(M_i), l)
(expr(M_i))	($\mathfrak{h}$(expr(M_i), l))
symm-expr, where e has form local-expr(N)	$\mathfrak{h}$(symm-expr)
constant	constant
name (where name is a global variable)	name
$\sum_{1 \le j \le \#N} e_j$	$\sum_{l \in S(N)} (eval(l, e) *$ `count_`N`_`$f_N(l))$
$\prod_{1 \le j \le \#N} e_j$	$\prod_{l \in S(N)} (eval(l, e)$**`count_`$N$`_`$f_N(l))$
$\bigwedge_{1 \le j \le \#N} e_j$	$\sum_{l \in \mathsf{SAT}_N(e)}$ `count_`N`_`$f_N(l) = \#N$
$\bigvee_{1 \le j \le \#N} e_j$	$\sum_{l \in \mathsf{SAT}_N(e)}$ `count_`N`_`$f_N(l) > 0$
symm-expr $\bowtie$ symm-expr	$\mathfrak{h}$(symm-expr) $\bowtie \mathfrak{h}$(symm-expr)
$\neg$symm-expr	$\neg \mathfrak{h}$(symm-expr)
(symm-expr)	($\mathfrak{h}$(symm-expr))

Figure 3. Rules for translating an SPSL specification $\mathcal{P}$ to a generic form $\mathfrak{h}(\mathcal{P})$.

$(1 \leq j \leq r)$.

The remaining rules in Fig. 3 are concerned with translation of expressions of the forms $\mathsf{expr}(M_i)$ and symm-expr. The rationale for these rules is similar to the rationale presented in [6], generalised to our more expressive language, and extended with symmetric summations and products.

B. Model checking $\mathcal{M}(\mathcal{P})$ via $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$

Let ϕ be an $SPCTL$ property over $\mathcal{P}$. Recall from Section III-C that the atoms of ϕ are expressions of the form symm-expr over the variables of $\mathcal{P}$. We translate ϕ into a generic $SPCTL$ property $\mathfrak{h}(\phi)$ over $\mathfrak{h}(\mathcal{P})$ by replacing each atom e appearing in ϕ with $\mathfrak{h}(e)$, using the translation rules for symm-expr in Fig. 3.

Let $\mathcal{P}$ be an SPSL specification with associated MDP $\mathcal{M}(\mathcal{P}) = (S, s_0, Steps)$. Since $\mathfrak{h}(\mathcal{P})$ is itself an SPSL specification, the MDP $\mathcal{M}(\mathfrak{h}(\mathcal{P})) = (S', s'_0, Steps')$ associated with $\mathfrak{h}(\mathcal{P})$ can be derived directly as shown in Section III-B. However, it is useful to describe the states of $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$ with reference to $\mathcal{P}$. If $\mathbf{M}$ is the set of module types for $\mathcal{P}$, the state set S' associated with $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$ is:

$$S' = G \times \big(\bigotimes_{M \in \mathbf{M}} S(\texttt{generic\_}M) \big).$$

We define a function $\delta : S \to S'$ that maps equivalent states in S to a single state in S' – a *generic* representative.

For $M \in \mathbf{M}$, $s \in S$ and $l \in S(M)$, $\delta^l_M(s)$ denotes the number of instances of M with local state l at s. Thus δ^l_M maps S to $\{0, 1, \ldots, \#M\}$. Suppose that $S(M) = \{l_1 < l_2 < \cdots < l_t\}$ for some $t \geq 0$. Let $\delta_M(s) = (\delta^{l_1}_M(s), \delta^{l_2}_M(s), \ldots, \delta^{l_t}_M(s))$. The mapping $\delta_M : S \to S(\texttt{generic\_}M)$ transforms s into a tuple counting the number of instances of M residing in each local state of $S(M)$.

Suppose that the module types for $\mathcal{P}$ are, for some $d > 0$, $M^1, M^2, \ldots, M^d$, and let $s = (s_{global}, s_{local}) \in S$. Then we define $\delta(s) = (s_{global}, \delta_{M^1}(s), \delta_{M^2}(s), \ldots, \delta_{M^d}(s))$. We therefore have $\delta(s)_{global} = s_{global}$ and $\delta(s)_{local} = (\delta_{M^1}(s), \delta_{M^2}(s), \ldots, \delta_{M^d}(s))$.

We use the function δ to give a list of properties of our translation method (see Appendix A for proofs).

1) If e has the form symm-expr then $eval(s, e) = eval(\delta(s), \mathfrak{h}(e))$. If e is boolean then $s \models e \Leftrightarrow \delta(s) \models \mathfrak{h}(e)$.
2) If e has the form $\mathsf{expr}(M_i)$ and M_i has local state l at s then $eval(s, e) = eval(\delta(s), \mathfrak{h}(e, l))$. If e is boolean then $s \models e \Leftrightarrow \delta(s) \models \mathfrak{h}(e, l)$.
3) If u has the form $\mathsf{update}(M)$ and M_i has local state l at s then $exec(\delta(s), \mathfrak{h}(u, l)) = \delta(exec(s, u))$.

The above results allow us to prove that the reachable parts of $\overline{\mathcal{M}(\mathcal{P})}$ and $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$ are isomorphic, and that we can infer $SPCTL$ properties of $\mathcal{M}(\mathcal{P})$ by checking $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$. (See Appendix A for proofs.)

Without loss of generality, in the following theorems we consider MDPs to be restricted to reachable states.

Theorem 4: Let $\mathcal{P}$ be an SPSL specification. Let $\overline{\mathcal{M}(\mathcal{P})} = (\overline{S}, \overline{s_0}, \overline{Steps})$ be the quotient MDP for $\mathcal{M}(\mathcal{P}) = (S, s_0, Steps)$ *w.r.t.* the group $\underline{Sym}(\mathbf{M})$. Then δ, restricted to $\overline{S}$, is an isomorphism from $\overline{\mathcal{M}(\mathcal{P})}$ to $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$.

Theorem 1 now applies, in which γ is taken to be the translation function $\mathfrak{h}$, which is indeed a bijection over the atoms of $SPCTL$, *i.e.* expressions of the form symm-expr.

Theorem 5: $\mathcal{M}(\mathcal{P}) \models \phi \Leftrightarrow \mathcal{M}(\mathfrak{h}(\mathcal{P})) \models \mathfrak{h}(\phi)$.

Two extensions to our approach that have been implemented and deserve note are: the relaxation of the restriction to variable updates described above to allow updates to local variables with expressions involving globals; and the extension of the approach to several other model types (Kripke structions, Discrete Time Markov Chains and Continuous Time Markov Chains). We do not include details here for space reasons.

C. Running example

Fig. 4 shows the generic version of the running example of Fig. 2. Counter variables `count_master_`j and `count_worker_`j are abbreviated to `m_`j and `w_`j, respectively. There are two counter variables for the generic master process, since there are two possible master local states, (T) and (F) depending whether boolean local variable `active` is true or false. Similarly, there are four worker local states, $(0, F), (0, T), (1, F), (1, T)$, corresponding to the possible configurations of the `awake` and `working` local variables. For each statement labelled (1) to (7) in Fig. 2, Fig. 4 contains a label associated with corresponding statements in the reduced specification. We describe the translation process for statements (1) and (5).

Consider statement (1) of Fig. 2. The guard for this statement has the form $e_i \wedge d$ where e is `active` and d is $\sum_{1 \leq j \leq 10} \texttt{awake}_j > 0$. We statically compute $\mathsf{SAT}_{\texttt{master}}(\texttt{active}) = \{(T)\}$, where $(T) = l$, say. We have $f_{\texttt{master}}(l) = 2$, so we output a single generic statement with guard `m_2` $> 0 \wedge \mathfrak{h}(d, l)$. According to the translation rules of Fig. 3, $\mathfrak{h}(d, l)$ reduces to $\mathfrak{h}(d') > 0$, where d' is the sum of the expression `awake` over all worker modules. The translated expression $\mathfrak{h}(d')$ is a linear combination over the counter variables representing worker local states. For worker local state m, the linear combination includes the term $eval(m, \texttt{awake}) * \texttt{w\_}f_{\texttt{worker}}(m)$. This is the contribution to the original sum for every worker module in state m. Thus for $m = (0, F)$ we output the term $0 * \texttt{w\_1}$ because $eval(m, \texttt{awake}) = 0$ and $f_{\texttt{worker}}(m) = 1$. For $m = (1, T)$ we have $f_{\texttt{worker}}(m) = 4$ and $eval(m, \texttt{awake}) = 1$, resulting in the term $1 * \texttt{w\_4}$. The complete generic guard is shown in statement (1) of Fig. 4. (A constant folding optimization can be used to simplify this guard by removing

```
globals { m_to_w [0..5] init 0 }
module generic_master[1] {
    m_1 : [0..3] init 0    // num. masters in state (F)
    m_2 : [0..3] init 3    // num. masters in state (T)
(1) m_2 > 0 ∧ 0∗w_1+0∗w_2+1∗w_3+1∗w_4 > 0
    ∧ m_to_w<5 → 0.5:(m_to_w := m_to_w + 1) +
0.5:skip
(2) m_2 > 0 ∧ 0∗w_1+0∗w_2+1∗w_3+1∗w_4 = 0
    → 1:(m_2 := m_2 − 1) ‖ (m_1 := m_1 + 1)
(3) m_1 > 0 → 1:skip
}
module generic_worker[1] {
    w_1 : [0..10] init 0    // num. workers in state (0,F)
    w_2 : [0..10] init 0    // num. workers in state (0,T)
    w_3 : [0..10] init 10   // num. workers in state (1,F)
    w_4 : [0..10] init 0    // num. workers in state (1,T)
(4) w_3 > 0 ∧ m_to_w > 0 → 1:(w_3 := w_3 − 1)
      ‖ (w_4 := w_4 + 1) ‖ (m_to_w := m_to_w − 1)
(5) w_2 > 0 → 1:(w_2 := w_2 − 1) ‖ (w_1 := w_1 + 1)
    w_4 > 0 → 1:(w_4 := w_4 − 1) ‖ (w_3 := w_3 + 1)
(6) w_3 > 0 →
      0.1:(w_3 := w_3 − 1) ‖ (w_1 := w_1 + 1) +
0.9:skip
(7) w_1 > 0 → 1:skip
    w_2 > 0 → 1:skip
}
```

Figure 4. Generic SPSL specification corresponding to the *master/worker* specification of Fig. 2.

redundant multiplications by zero and one.) The stochastic update associated with the reduced statement is the same as for the original, since the update does not modify local state. Note that the probability distribution is passed through unchanged.

The guard for statement (5) in Fig. 2, for a worker module, can be re-written as $\texttt{working}_i \ \wedge \ \texttt{true}$, which has the form $\mathsf{local\text{-}expr}(\texttt{worker})_i \wedge \mathsf{expr}(\texttt{worker}_i)$. We compute $\mathsf{SAT}_{\texttt{worker}}(\texttt{working}) = \{(0,T),(1,T)\}$ and generate a generic statement for each of these local states. Considering the second local state, $(1,T) = l$ say, we have $f_{\texttt{worker}}(l) = 4$, so the generic guard is $\texttt{w\_4} > 0 \ \wedge \ \texttt{true}$, or just $\texttt{w\_4} > 0$. The stochastic update associated with the statement consists of one local update, $\texttt{working}_i := 0$. Given that the worker module executing this update is in local state $l = (1,T)$, after the update the worker has local state $m = (1,F)$. Since $f_{\texttt{worker}}(l) = 4$ and $f_{\texttt{worker}}(m) = 3$, the generic update involves decrementing $\texttt{w\_4}$ and incrementing $\texttt{w\_3}$.

Recall the $SPCTL$ property for the master/worker specification presented at the end of Section III-D. We convert this property to generic form by applying the translation function $\mathfrak{h}$ to each subformula of the form $\mathsf{symm\text{-}expr}$. Subformula $\texttt{m\_to\_w} > 0$ is left unchanged by $\mathfrak{h}$. Continuing to use the abbreviations $\texttt{m\_}j$ and $\texttt{w\_}j$, we have:

$$\begin{aligned}
&\mathfrak{h}(\textstyle\bigwedge_{1\le j\le 10} \texttt{awake}_j = 0)\\
&= (\textstyle\sum_{l\in \mathsf{SAT}_{\texttt{worker}}(\texttt{awake=0})} \texttt{w_}f_{\texttt{worker}}(l) = 10)\\
&= (\textstyle\sum_{l\in\{(0,F),(0,T)\}} \texttt{w_}f_{\texttt{worker}}(l) = 10)\\
&= (\texttt{w_}f_{\texttt{worker}}((0,F)) + \texttt{w_}f_{\texttt{worker}}((0,T)) = 10)\\
&= (\texttt{w_1} + \texttt{w_2} = 10)
\end{aligned}$$

Thus the generic $SPCTL$ formula is:

$$P_{<0.01}[\ \Diamond\ (\texttt{m\_to\_w} > 0 \ \wedge \ \texttt{w\_1} + \texttt{w\_2} = 10)].$$

V. Experimental Results

We have implemented our techniques in a tool, GRIP (Generic Representatives in PRISM). It accepts a PRISM specification conforming to a syntax analogous to SPSL, and outputs a generic specification for model checking with PRISM. GRIP supports MDP, DTMC and CTMC specifications, with associated symmetric temporal properties.

We apply GRIP to seven case studies: Aspnes & Herlihy's randomised consensus protocol (*consensus*); a randomised Byzantine agreement protocol (*byzantine*); the randomised mutual exclusion protocols of Pnueli & Zuck (*mutex*) and Rabin (*rabin*); a simplified model of the Fibroblast Growth Factor signalling pathway (*fgf*); a peer-to-peer protocol based on BitTorrent (*peer2peer*); and Dolev *et al.*'s minimum space shared memory leader election protocol (*leader*). All models, properties and references, as well as binary and source code versions of GRIP, can be found at [14].

We compare the performance of GRIP with PRISM and PRISM-symm [10], a symmetry reduction tool, implemented in PRISM, that constructs a symbolic (MTBDD) representation for the unreduced model and then performs symmetry reduction at the level of the MTBDD data structure. Experiments were performed on a 2.40 GHz PC with 2 GB RAM, running PRISM version 3.2 under Linux.

A. Results

Experimental results are shown in Fig. 5. For each family of specifications, $\#M$ denotes the number of symmetric modules in a particular specification and $|S(M)|$ the number of local states of each symmetric module (all of our examples consist of multiple instances of a single module type).

Columns 3–7 give the sizes of the unreduced and reduced models (produced by either GRIP or PRISM-symm) and the model storage requirements. The latter is given in terms of the number of nodes in the MTBDD used by PRISM to represent the model (one node uses 20 bytes of storage).

Columns 8-13 give the time required to build each model and perform probabilistic model checking. For the model building process, all three tools start from the same high-level model specification. For PRISM, this specification is translated into an MTBDD representation of the corresponding probabilistic model, the set of all reachable states is computed, and all unreachable states are removed. PRISM-symm performs the same model construction process, and then applies symmetry reduction to the MTBDD representation. GRIP first applies the language-level symmetry reduction

process described in this paper then passes the resulting generic model specification to PRISM for construction. The models constructed by each tool are model checked using PRISM, applying the fastest available technique of the tool.

B. Discussion

As is clearly illustrated by the table in Fig. 5, the reductions in state-space provided by symmetry reduction are substantial. This has a significant effect on the efficiency or feasibility of applying probabilistic model checking. This is because, in many cases, when numerical computations are performed on the model, storage proportional to the number of states is required (even when using compact, symbolic techniques to store the model itself). For some examples symmetry reduction permits verification where it was previously impossible due to insufficient memory or prohibitively long solution times.

Another important factor for the efficiency of model checking is the size of the MTBDD representing the model. In 4 out of 7 case studies, GRIP produces a smaller MTBDD than both PRISM and PRISM-symm. In other cases both GRIP and PRISM-symm produce a larger MTBDD than PRISM, despite the much smaller state-space. This is caused by the loss of regularity in the probabilistic model, once symmetric states have been collapsed.

From Fig. 5, we see that in terms of model construction time GRIP also performs better than the other tools on 4 out of 7 examples. In two cases GRIP performs poorly – this is due to the size of the MTBDDs produced for these models.

A comparison of GRIP and PRISM-symm in terms of verification time is also interesting. For three case studies, model checking is much faster on the GRIP-reduced models, despite the fact the symmetric quotient model is the same. For two of these (*consensus* and *byzantine*) the cause is a side-effect of the PRISM-symm reduction process whereby duplicated probability distributions can appear in the quotient MDP. This has no effect on the correctness of verification, but can slow down the process. On the third case study (*rabin*), the superior performance of GRIP is due to the fact that it results in a much smaller MTBDD.

On another example (*peer2peer*), model checking of the GRIP-reduced model is slower than for PRISM-symm. Here, the large number of variables in the GRIP model results in an MTBDD which is expensive to manipulate. Typically, GRIP is faster for systems comprising a large number of simple modules, whereas PRISM-symm will perform better on a small number of more complex modules.

VI. Related Work

Symmetry reduction techniques for model checking were originally proposed in [1], [2], [3] and have been studied extensively since (see [4] for a survey). The use of generic representatives in model checking was proposed in [5] and extended in [6], [15]. The approach was extended to probabilistic model checking [7] via the definition of *Symmetric PRISM*, a precursor to SPSL. A tool paper describing a preliminary version of the GRIP tool appeared as [8].

Another approach to symmetry reduction for probabilistic model checking is given in [10], which performs a model-level reduction built into the symbolic implementation of the PRISM tool. This approach is based on dynamic symmetry reduction for non-probabilistic model checking [16], [17]. Language-level exploitation of symmetry has also been explored for several other probabilistic modelling formalisms, for example, stochastic process algebrae [18], stochastic Petri nets [19] and stochastic activity networks [20].

VII. Conclusions and Future Work

We have introduced SPSL, a simple but general language for specifying symmetric, probabilistic systems. We have provided an algorithm, based on the generic representatives approach to symmetry reduction, which translates an SPSL specification into an equivalent generic form. This can then be used to model check symmetric properties of the original model, with considerable increase in efficiency thanks to the reduced size of the generic model. SPSL is a much richer language than those provided by existing generic representatives techniques, allowing for example multiple families of symmetric processes, multiple local variables per process and complex expressions over these variables. We have implemented our techniques in a tool, GRIP, which targets the PRISM modelling language, and have presented very positive results for a large set of case studies.

We plan to extend this work in a number of directions. Firstly, some useful features of the PRISM modelling language, notably synchronisation between modules, have not yet been incorporated into SPSL. We hope to extend our generic representatives approach to include these. Secondly, the fact that these techniques are based on language-level translations raises a number of interesting issues concerning, for example, the provision to the user of probabilistic counterexamples or optimal adversaries. Finally, we plan to investigate optimising symbolic implementations of probabilistic model checking for generic representatives, for example using zero-suppressed variants of binary decision diagrams.

Acknowledgement

Alastair F. Donaldson, David Parker and Alice Miller are supported by EPSRC projects EP/G051100, EP/D07956X and EP/E032354 respectively.

References

[1] E. Clarke, S. Jha, R. Enders, and T. Filkorn, "Exploiting symmetry in temporal logic model checking." *Formal Methods in System Design*, vol. 9, no. 1/2, pp. 77–104, 1996.

[2] E. Emerson and A. Sistla, "Symmetry and model checking." *Formal Methods in System Design*, vol. 9, no. 1/2, pp. 105–131, 1996.

Case study ($\|S(M)\|$)	$\#M$	Model size (states)		Model storage (MTBDD nodes)			Build time (sec.)			Model check time (sec.)		
		Full model	Symm. reduced	PRISM	PRISM -symm	GRIP	PRISM	PRISM -symm	GRIP	PRISM	PRISM -symm	GRIP
	8	6.1+e7	46,482	15,681	14,078	12,008	1.18	1.48	2.90	20,013	12.5	7.57
consensus	12	1.2e+11	339,729	50,431	45,352	27,766	5.55	6.34	5.89	>24h	183.0	99.8
(6)	16	2.1e+14	1.5e+6	116,133	104,902	60,107	15.3	21.4	14.4	>24h	1,380	734.3
	8	6.4e+8	298,993	713,143	167,587	167,372	19.1	21.8	15.6	92.0	8.53	7.88
byzantine	12	1.0e+13	8.0e+6	4.3e+6	937,484	646,455	132.9	160.5	33.6	mem-out	91.1	57.3
(10)	16	1.9e+16	1.1e+8	1.3e+7	3.0e+6	1.9e+6	831.0	1,013	113.5	mem-out	429.6	190.5
	4	201,828	11,130	65,624	96,559	53,171	1.19	2.52	14.6	0.11	0.18	0.11
rabin	6	1.3e+8	356,592	206,213	408,291	185,943	6.22	14.3	42.2	0.46	0.60	0.45
(17)	8	4.5e+10	4.1+e6	381,184	796,324	430,901	17.5	41.2	87.3	0.91	1.74	1.13
	5	4.6e+6	63,756	224,843	440,445	601,627	10.5	18.2	17.2	1009.0	11.5	13.2
fgf	6	9.6e+7	283,360	522,063	1.0e+6	1.2e+6	39.9	64.4	39.6	>24h	69.2	72.6
(19)	7	2.0e+9	1.1e+6	1.1e+6	2.2e+6	2.3e+6	110.2	180.9	94.2	mem-out	342.1	354.2
	5	3.4e+7	376,992	26,266	101,630	157,476	0.10	1.06	3.93	1055.1	9.10	24.8
peer2peer	6	1.1e+9	2.3e+6	40,591	189,704	247,122	0.12	3.07	4.77	mem-out	72.8	174.8
(32)	7	3.4e+10	1.3e+7	54,916	306,123	355,721	0.40	4.61	5.40	mem-out	472.4	1,049
	60	4.2e+28	1,891	44,780	32,989	1,602	3.10	30.4	1.94	72.2	3.83	0.07
leader	100	5.2e+47	5,151	122,660	90,989	2,858	11.7	259.1	2.04	997.0	20.0	0.19
(3)	140	6.3e+66	10,011	238,940	177,789	4,618	32.5	1,081	2.24	6,241	60.5	0.42
	30	1.0e+31	7.3e+8	258,937	135,428	97,977	235.8	257.4	99.5	146.8	14.9	10.2
mutex	36	1.2e+37	3.1e+9	373,159	192,839	137,758	521.0	558.9	458.6	431.8	43.7	29.1
(16)	42	1.4e+43	1.1e+10	508,117	260,294	166,237	1,049	1,052	2,218	635.5	40.5	32.2

Figure 5. Experimental results for GRIP, PRISM and PRISM-symm.

[3] C. Ip and D. Dill, "Better verification through symmetry." *Formal Methods in System Design*, vol. 9, no. 1/2, pp. 41–75, 1996.

[4] A. Miller, A. Donaldson, and M. Calder, "Symmetry in temporal logic model checking," *ACM Computing Surveys*, vol. 38, no. 3, p. 8, 2006.

[5] E. Emerson and R. Trefler, "From asymmetry to full symmetry: New techniques for symmetry reduction in model checking." in *Proc. CHARME'99*, ser. LNCS, vol. 1703. Springer, 1999, pp. 142–156.

[6] E. Emerson and T. Wahl, "On combining symmetry reduction and symbolic representation for efficient model checking." in *Proc. CHARME'03*, ser. LNCS, vol. 2860. Springer, 2003, pp. 216–230.

[7] A. Donaldson and A. Miller, "Symmetry reduction for probabilistic model checking using generic representatives." in *Proc. ATVA'06*, ser. LNCS, vol. 4218. Springer, 2006, pp. 9–23.

[8] A. Donaldson, A. Miller, and D. Parker, "GRIP: Generic representatives in PRISM," in *Proc. QEST '07*. IEEE Computer Society, 2007, pp. 115–116.

[9] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems." in *Proc. TACAS'06*, ser. LNCS, vol. 3920. Springer, 2006, pp. 441–444.

[10] M. Kwiatkowska, G. Norman, and D. Parker, "Symmetry reduction for probabilistic model checking." in *Proc. CAV'06*, ser. LNCS, vol. 4144. Springer, 2006, pp. 234–248.

[11] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[12] A. Bianco and L. de Alfaro, "Model checking of probabilistic and nondeterministic systems," in *Proc. FSTTCS'95*, ser. LNCS, vol. 1026. Springer, 1995, pp. 499–513.

[13] A. Miller and A. Donaldson, "Property preservation in quotient structures." Department of Computing Science, University of Glasgow, Technical Report TR-2008-270, 2008.

[14] *GRIP Website*, University of Glasgow/PRISM group, 2008, www.prismmodelchecker.org/grip/.

[15] E. Emerson and T. Wahl, "Efficient reduction techniques for systems with many components." *Electr. Notes Theor. Comput. Sci.*, vol. 130, pp. 379–399, 2005.

[16] E. Emerson and T. Wahl, "Dynamic symmetry reduction." in *Proc. TACAS'05*, ser. LNCS, vol. 3440. Springer, 2005, pp. 382–396.

[17] T. Wahl, N. Blanc, and E. Emerson, "SVISS: Symbolic verification of symmetric systems," in *Proc. TACAS'08*, ser. LNCS, vol. 4963. Springer, 2008, pp. 459–462.

[18] S. Gilmore, J. Hillston, and M. Ribaudo, "An efficient algorithm for aggregating PEPA models," *IEEE Trans. Software Eng.*, vol. 27, no. 5, pp. 449–464, 2001.

[19] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, "Stochastic well-formed coloured nets for symmetric modeling applications," *IEEE Trans. Computers*, vol. 42, no. 11, 1993.

[20] J. Meyer and W. Sanders, "Reduced base model construction methods for stochastic activity networks," *IEEE J. Selected Areas in Comm.*, vol. 9, no. 1, pp. 25–36, 1991.

APPENDIX

In this appendix, we provide proofs of properties 1–3 stated on page 7, and of Theorems 4 and 5.

Throughout the proofs we use the following notation. Let $M \in \mathbf{M}$, $\alpha \in Sym(\mathbf{M})$, $1 \leq i \leq \#M$, and let s be a state. Then $\alpha(i)$ denotes the unique j for which $\alpha(M_i) = M_j$; $s(i)$ denotes the local state of module M_i at s.

We begin by proving properties 1 – 3 of our translation method, stated on page 7.

Lemma 1: (First part of property 1, page 7.) If e has the form symm-expr then $eval(s,e) = eval(\delta(s), \mathfrak{h}(e))$.

Proof We consider the translation rules given in Fig. 3 for the various forms for symm-expr. If e is a constant, the result is trivial. If e has the form name then e is a global variable name, g. Since the translation process preserves the global variables of $\mathcal{P}$ in $\mathfrak{h}(\mathcal{P})$, we have $eval(s,e) = eval(s,g) = eval(\delta(s),g) = eval(\delta(s), \mathfrak{h}(e))$.

Suppose e has the form $\sum_{1\leq j\leq \#N} e'_j$, where e' has the form local-expr(N). This is the sum of the expression e' evaluated with respect to the local state of every instance of N. We have:

$$\begin{aligned}
&eval(s,e)\\
&\quad = eval(s, \textstyle\sum_{1\leq j\leq \#N} e'_j)\\
&\quad = \textstyle\sum_{l\in S(N)} \sum_{\substack{1\leq j\leq \#N\\ s(j)=l}} eval(l,e')\\
&\quad = \textstyle\sum_{l\in S(N)} \big(eval(l,e') \times \delta^l_N(s)\big)\\
&\quad = eval(\delta(s), \textstyle\sum_{l\in S(N)} (eval(l,e') * \texttt{count_}N\texttt{_}f_N(l)))\\
&\quad = eval(\delta(s), \mathfrak{h}(e)).
\end{aligned}$$

The case where $\sum$ is replaced by $\prod$ follows similarly.

Suppose e has the form $\bigwedge_{1\leq j\leq \#N} e'_j$, where e' has the form local-expr(N). Then we have:

$$\begin{aligned}
&s \models e\\
&\quad \Leftrightarrow s \models \textstyle\bigwedge_{1\leq j\leq \#N} e'_j\\
&\quad \Leftrightarrow \forall_{1\leq j\leq \#N}\ s(j) \models e'\\
&\quad \Leftrightarrow \forall_{1\leq j\leq \#N}\ s(j) \in \mathsf{SAT}_N(e')\\
&\quad \Leftrightarrow \textstyle\sum_{l\in \mathsf{SAT}_N(e')} \delta^l_N(s) = \#N \text{ (by definition of } \delta^l_N)\\
&\quad \Leftrightarrow \delta(s) \models \big(\textstyle\sum_{l\in \mathsf{SAT}_N(e')} \texttt{count_}N\texttt{_}f_N(l) = \#N\big)\\
&\quad \Leftrightarrow \delta(s) \models \mathfrak{h}(e,l).
\end{aligned}$$

It follows that $eval(s,e) = eval(s, \mathfrak{h}(e,l))$ in this case, since for a boolean expression e, $eval(s,e) = eval(\delta(s), \mathfrak{h}(e))$ iff $(s \models e \Leftrightarrow \delta(s) \models \mathfrak{h}(e))$.

The case where $\bigwedge$ is replaced by $\bigvee$ follows similarly, and the cases where e has the form symm-expr $\bowtie$ symm-expr, $\neg$symm-expr and (symm-expr) are proved using straightforward induction. The result follows. □

Corollary 1: (Second part of property 1, page 7.) With the conditions of Lemma 1, if e is boolean then $s \models e \Leftrightarrow \delta(s) \models \mathfrak{h}(e)$.

Lemma 2: (First part of property 2, page 7.) If e has the form expr(M_i) and M_i has local state l at s then $eval(s,e) = eval(\delta(s), \mathfrak{h}(e,l))$.

Proof We consider the translation rules given in Fig. 3 for the various forms for expr(M_i). Suppose e has the form e'_i, where e' has the form local-expr. We have $eval(s,e) = eval(s,e'_i) = eval(l,e') = eval(\delta(s), \mathfrak{h}(e'_i,l))$ (see Fig. 3) $= eval(\delta(s), \mathfrak{h}(e,l))$. If e has the form e', where e' has the form symm-expr, then $eval(s,e) = eval(s,e') = eval(\delta(s), \mathfrak{h}(e'))$ (by Lemma 1) $= eval(\delta(s), \mathfrak{h}(e,l))$.

Suppose e has the form $\prod_{1\leq j\neq i\leq \#M} e'_j$, where e' has the form local-expr$M$. Then we have:

$$\begin{aligned}
&eval(s,e)\\
&\quad = eval(s, \textstyle\prod_{1\leq j\neq i\leq \#M} e'_j)\\
&\quad = eval(s, (\textstyle\prod_{1\leq j\leq \#M} e'_j)/e'_i)\\
&\quad = eval(s, \textstyle\prod_{1\leq j\leq \#M} e'_j)/eval(s,e'_i).
\end{aligned}$$

Now $\prod_{1\leq j\leq \#M} e'_j$ has the form symm-expr and e'_i has the form local-exprM_i. Using Lemma 1 combined with the above argument and the translation rules for symm-expr in Fig. 3, we get:

$$\begin{aligned}
&eval(s, \textstyle\prod_{1\leq j\leq \#M} e'_j)/eval(s,e'_i)\\
&\quad = eval(\delta(s), \mathfrak{h}(\textstyle\prod_{1\leq j\leq \#M} e'_j))/eval(l,e')\\
&\quad = eval(\delta(s), \textstyle\prod_{m\in S(M)} (eval(m,e')^{**}\texttt{count_}M\texttt{_}f_M(m)))\\
&\qquad\quad /eval(l,e')\\
&\quad = eval(\delta(s), \textstyle\prod_{m\in S(M)} (eval(m,e')^{**}\texttt{count_}M\texttt{_}f_M(m))\\
&\qquad\quad /eval(l,e'))\\
&\quad = eval(\delta(s), \mathfrak{h}(e,l)).
\end{aligned}$$

The case where $\prod$ is replaced by $\sum$ follows similarly.

Suppose e has the form $\bigvee_{1\leq j\neq i\leq \#M} e'_j$. The translation rule of Fig. 3 for this case has two alternatives depending on whether the input local state l satisfies e'. Suppose first that $l \not\models e'$. Then we have $eval(s,e'_i) = F$, so:

$$\begin{aligned}
&eval(s,e)\\
&\quad = eval(s,e) \vee F\\
&\quad = eval(s,e) \vee eval(s,e'_i)\\
&\quad = eval(s, e \vee e'_i)\\
&\quad = eval(s, (\textstyle\bigvee_{1\leq j\neq i\leq \#M} e'_j) \vee e'_i)\\
&\quad = eval(s, \textstyle\bigvee_{1\leq j\leq \#M} e'_j)\\
&\quad = eval(\delta(s), \mathfrak{h}(\textstyle\bigvee_{1\leq j\leq \#M} e'_j)) \text{ (by Lemma 1)}\\
&\quad = eval(\delta(s), \textstyle\sum_{m\in \mathsf{SAT}_M(e')} \texttt{count_}M\texttt{_}f_M(m) > 0)\\
&\quad = eval(\delta(s), \mathfrak{h}(e,l)
\end{aligned}$$

Suppose instead that $l \models e'$. We have:

$$\begin{aligned}
&s \models e\\
&\quad \Leftrightarrow s \models \textstyle\bigvee_{1\leq j\neq i\leq \#M} e'_j\\
&\quad \Leftrightarrow \exists_{1\leq j\neq i\leq \#M}\ s(j) \models e'\\
&\quad \Leftrightarrow \exists_{1\leq j\neq i\leq \#M}\ s(j) \in \mathsf{SAT}_N(e')\\
&\quad \Leftrightarrow \big(\textstyle\sum_{m\in \mathsf{SAT}_M(e')\setminus\{l\}} \delta^m_M(s)\big) + (\delta^l_M(s) - 1) > 0\\
&\qquad\quad \text{(by definition of } \delta^m_M \text{, and because } s(i) = l)\\
&\quad \Leftrightarrow \textstyle\sum_{m\in \mathsf{SAT}_M(e')} \delta^m_M(s) - 1 > 0\\
&\quad \Leftrightarrow \textstyle\sum_{m\in \mathsf{SAT}_M(e')} \delta^m_M(s) > 1\\
&\quad \Leftrightarrow \delta(s) \models \big(\textstyle\sum_{m\in \mathsf{SAT}_M(e')} \texttt{count_}M\texttt{_}f_M(m) > 1\big)\\
&\quad \Leftrightarrow \delta(s) \models \mathfrak{h}(e,l).
\end{aligned}$$

We have shown that $s \models e \Leftrightarrow \delta(s) \models \mathfrak{h}(e,l)$. It follows that $eval(s,e) = eval(\delta(s), \mathfrak{h}(e,l))$.

The case where $\bigvee$ is replaced by $\bigwedge$ follows similarly, and the cases where e has the form $\mathsf{expr}(M_i) \bowtie \mathsf{expr}(M_i)$, $\neg\mathsf{expr}(M_i)$ and $(\mathsf{expr}(M_i))$ are proved using straightforward induction. The result follows. □

Corollary 2: (Second part of property 2, page 7.) With the conditions of Lemma 2, if e is boolean then $s \models e \Leftrightarrow \delta(s) \models \mathfrak{h}(e,l)$.

Lemma 3: (Property 3, page 7.) If u has the form $\mathsf{update}(M)$ and M_i has local state l at s then $exec(\delta(s), \mathfrak{h}(u,l)) = \delta(exec(s,u))$.

Proof If u is `skip` then $exec(s,u) = s$ and $exec(\delta(s), \mathfrak{h}(u,l)) = \delta(s)$, and the result follows trivially.

Otherwise, assume without loss of generality that u has the form: $(v_i^1 := e_i^1) \| \ldots \| (v_i^t := e_i^t) \| (g^1 := d^1) \| \ldots \| (g^r := d^r)$, where $v^j \in var(M)$, e^j has the form $\mathsf{local\text{-}expr}(M)$ $(1 \leq j \leq t)$, $g^j \in global$ and d^j has the form $\mathsf{expr}(M_i)$ $(1 \leq j \leq r)$. Assume further that $r = |global|$, i.e. u assigns to *every* global variable in the specification (we can always express u in this form by adding dummy assignments of the form $g := g$ if necessary).

Let $t = exec(s,u)$ and $r = exec(\delta(s), \mathfrak{h}(u,l))$. We must prove that $r = \delta(t)$. We prove this by showing that $r_{global} = \delta(t)_{global}$ and $r_{local} = \delta(t)_{local}$.

For $1 \leq j \leq r$ we have $t(g^j) = eval(s, d^j) = a$ say. Since, by definition, δ has no effect on global variables, $\delta(t)(g^j) = a$ also. According to the translation rule for updates, $\mathfrak{h}(u,l)$ contains the assignment $g^j := \mathfrak{h}(d^j, l)$, so $r(g^j) = eval(\delta(s), \mathfrak{h}(d^j,l)) = eval(s, d^j)$ (by Lemma 2) $= a$. Therefore $\delta(t)(g^j) = r(g^j)$. Since u assigns to every global variable in the specification, it follows that $r_{global} = \delta(t)_{global}$.

Suppose the module types for $\mathcal{P}$ are $N^1, N^2, \ldots, N^k$ $(k > 0)$. By definition of δ we have

$$\delta(s)_{local} = (\delta_{N^1}(s), \delta_{N^2}(s), \ldots, \delta_{N^k}(s))$$

and

$$\delta(t)_{local} = (\delta_{N^1}(t), \delta_{N^2}(t), \ldots, \delta_{N^k}(t)).$$

Suppose $r_{local} = (x^1, x^2, \ldots, x^k)$. We must show that $x^j = \delta_{N^j}(t)$ $(1 \leq j \leq k)$.

Let $1 \leq j \leq k$ be such that $N^j \neq M$. Since u does not affect the local state of any module of type N^j we find that $\delta_{N^j}(t) = \delta_{N^j}(s)$. Additionally, the translated update $\mathfrak{h}(u,l)$ does not modify any counter variable `count_`N^j`_`w $(1 \leq w \leq |S(N^j)|)$, so $x^j = \delta_{N^j}(s)$. It follows that $x^j = \delta_{N^j}(t)$.

To complete the proof we must show that $x^j = \delta_M(t)$ when $1 \leq j \leq k$ is the unique j such that $N^j = M$. We know that M_i has local state l at s. Let $l' = l[v^1 := eval(l, e^1), \ldots, v^t := eval(l, e^t)]$ as in Fig. 3. Thus M_i has local state l' at t. The result follows trivially if $l = l'$, for in this case u does not affect the local state of any module of type M.

Suppose instead that $l \neq l'$. Let $S(M) = \{l_1 < l_2 < \cdots < l_z\}$ for some $z > 0$, as in Section IV-B. We have $\delta_M(s) = (\delta_M^{l_1}(s), \delta_M^{l_2}(s), \ldots, \delta_M^{l_z}(s))$ and $\delta_M(t) = (\delta_M^{l_1}(t), \delta_M^{l_2}(t), \ldots, \delta_M^{l_z}(t))$. Let $x_j = (m_1, m_2, \ldots, m_z)$ so that m_w is the value of `count_`M`_`w at r $(1 \leq w \leq z)$. We must show that $m_w = \delta_M^{l_w}(t)$ $(1 \leq w \leq z)$.

If $l_w = l$ then $\delta_M^{l_w}(t) = \delta_M^l(t) = \delta_M^l(s) - 1 = \delta_M^{l_w}(s) - 1$. This is because the update u causes exactly one module (namely module M_i) to leave local state l, and does not cause any module to enter local state l. On the other hand, executing the update $\mathfrak{h}(u,l)$ at state $\delta(s)$ decrements the value of `count_`M`_`$f_M(l)$ by 1, so $eval(r, \texttt{count\_}M\texttt{\_}f_M(l)) = eval(\delta(s), \texttt{count\_}M\texttt{\_}f_M(l)) - 1$, i.e. $m_{f_M(l)} = \delta_M^l(s) - 1$. We have $f_M(l) = w$ since $l = l_w$, so we get $m_w = \delta_M^{l_w}(s) - 1 = \delta_M^{l_w}(t)$ as required.

If $l_w = l'$ then $\delta_M^{l_w}(t) = \delta_M^{l_w}(s) + 1$ since update u causes M_i to enter local state l', and causes no other module to enter or leave l'. On the other hand, executing $\mathfrak{h}(u,l)$ at $\delta(s)$ increments the value of `count_`M`_`$f_M(l')$ by 1. By an argument analogous to the case where $l_w = l$ we get $m_w = \delta_M^{l_w}(s) + 1 = \delta_M^{l_w}(t)$.

Finally, if $l_w \neq l$ and $l_w \neq l'$ then the update u causes no module to enter or leave state l_w, so $\delta_M^{l_w}(t) = \delta_M^{l_w}(s)$. Also, the update $\mathfrak{h}(u,l)$ does not modify the variable `count_`M`_`w, so $m_w = \delta_M^{l_w}(s) = \delta_M^{l_w}(t)$ as required. The result follows. □

Before proving Theorem 4 we prove two helper lemmas:

Lemma 4: $\mathcal{M}(\mathcal{P}) = (S, s_0, Steps)$ be the MDP associated with an SPSL specification $\mathcal{P}$. Let $s \in S$ and $\mu \in Steps(S)$. Then, for any $t_1, t_2 \in S$, if $\mu(t_1)$, $\mu(t_2) > 0$ and $t_1 \in [t_2]$ then $t_1 = t_2$.

Proof The distribution μ corresponds to the execution of a stochastic update $e^1{:}u^1 + e^2{:}u^2 \ldots e^w{:}u^w$ by module M_i at s. Let $s^j = exec(s, u^j)$ $(1 \leq j \leq w)$. Since each u^j updates only global variables and variables local to M_j, the s^j are identical except in the components which refer to global variables and local state of M_i. It follows that, for any $\alpha \in Sym(\mathbf{M})$, either $\alpha(s^j) = s^j$ or $\alpha(s^j) \neq s^k$ for any $1 \leq k \leq j$. Suppose we have $t_1 \neq t_2$ such that $\mu(t_1) > 0$, $\mu(t_2) > 0$ and $t_1 \in [t_2]$. By definition of μ (see Section III-B), we must have $t_1 = s^j$ and $t_2 = s^k$ for some $1 \leq j, k \leq w$. Since $t_1 \in [t_2]$, $s^j \in [s^k]$, i.e. $\alpha(s^j) = s^k$ for some $\alpha \in Sym(\mathbf{M})$. This is a contradiction since $s^j \neq s^k$. The result follows. □

Lemma 5: Let $s_1, s_2 \in S$. Then $\delta(s_1) = \delta(s_2) \Leftrightarrow s_1 \in [s_2]$.

Proof We have $\delta(s_1) = \delta(s_2)$ iff, for all $M \in \mathbf{M}$ and $l \in S(M)$, $\delta_M^l(s_1) = \delta_M^l(s_2)$. In other words, $\delta(s_1) = \delta(s_2)$ iff, for all $M \in \mathbf{M}$ and $l \in S(M)$, the number of instances of M with local state l at s_1 is equal to the number of instances of M with local state l at s_2. This is true iff s_1 and

s_2 are identical up to re-arrangement of module instances, i.e. there exists $\alpha \in Sym(\mathbf{M})$ such that $s_2 = \alpha(s_1)$, i.e. $s_1 \in [s_2]$. $\square$

Proof of Theorem 4, page 7. Let $\mathcal{M}(\mathfrak{h}(\mathcal{P})) = (S', s'_0, Steps')$. We first show that δ is a bijection between $\overline{S}$ and S'. Let $s_1, s_2 \in \overline{S}$ and suppose that $\delta(s_1) = \delta(s_2)$. By Lemma 5, $s_1 \in [s_2]$. Since s_1 and s_2 are both states of $\overline{\mathcal{M}(\mathcal{P})}$, from Definition 5 we have $s_1 = \min[s_2] = s_2$. Thus δ is injective. Observe that for any $z \in S'$ we can easily pick a state $s \in S$ such that $z = \delta(s)$. We then have $z = \delta(s) = \delta(\alpha(s))$ for any $\alpha \in Sym(\mathbf{M})$ (by Lemma 5), and so $z = \delta(\min[s])$. Since $\min[s] \in \overline{S}$ it follows that δ is surjective and thus a bijection.

Clearly $s'_0 = \delta(s_0)$. Let $\overline{s}$ be a reachable state in $\overline{S}$ and $\overline{\mu} \in \overline{Steps}(\overline{s})$. From Definition 5, there exists $\mu \in Steps(\overline{s})$ such that, for any $\overline{t} \in \overline{S}$, $\overline{\mu}(\overline{t}) = \sum_{z \in [\overline{t}]} \mu(z)$.

There must be a module M_i with a statement z such that z is executable at $\overline{s}$ and executing z at $\overline{s}$ gives rise to the distribution μ. Statement z has the form $e_i \wedge d \rightarrow e^1{:}u^1 + e^2{:}u^2 \ldots e^w{:}u^w$, where e has the form $\mathsf{local\text{-}expr}(M)$, d and e^j have the form $\mathsf{expr}(M_i)$ and u^j has the form $\mathsf{update}(M)$ $(1 \leq j \leq w)$.

Suppose M_i has local state $l \in S(M)$ at s. Since z is executable we must have $l \in \mathsf{SAT}_M(e)$, and therefore module $\texttt{generic\_}M$ has a statement $\mathfrak{h}(z, l)$. According to the translation rules of Fig. 3, the guard for $\mathfrak{h}(z, l)$ is: $\texttt{count\_}M\texttt{\_}f_M(l) > 0 \wedge \mathfrak{h}(d, l)$. Now $\delta(\overline{s}) \models \texttt{count\_}M\texttt{\_}f_M(l) > 0$ since $eval(\delta(\overline{s}), \texttt{count\_}M\texttt{\_}f_M(l)) = \delta^l_M(\overline{s})$, which is greater than zero as M_i is in local state l. Also, since $\overline{s} \models d$ we have $\delta(\overline{s}) \models \mathfrak{h}(d, l)$ by Corollary 2. Thus $\mathfrak{h}(z, l)$ is executable at $\delta(\overline{s})$.

Let $\mu' \in Steps'(\delta(\overline{s}))$ be the probability distribution which results from the execution of $\mathfrak{h}(z, l)$ at $\delta(\overline{s})$. We show that, for any $\overline{t} \in \overline{S}$, $\mu'(\delta(\overline{t})) = \overline{\mu}(\overline{t})$.

For $1 \leq j \leq t$, let λ^j denote $eval(\overline{s}, e^j)$ and let ν^j denote $eval(\delta(\overline{s}), \mathfrak{h}(e^j, l))$. Also let $s^j = exec(\overline{s}, u^j)$ and $r^j = exec(\delta(\overline{s}), \mathfrak{h}(u^j, l))$. By Lemma 2, $\lambda^j = \nu^j$, and by Lemma 3, $r^j = \delta(s^j)$ $(1 \leq j \leq w)$. By Lemma 5, we have $r^j = r^k$ iff $s^j \in [s^k]$. Combining this with Lemma 4 we have $r^j = r^k$ iff $s^j = s^k$.

Let $\overline{t} \in \overline{S}$. Then $\overline{\mu}(\overline{t}) = \sum_{t \in [\overline{t}]} \mu(x)$. By Lemma 4, $\mu(t) > 0$ for at most one $t \in [\overline{t}]$, so $\overline{\mu}(\overline{t}) = \mu(t)$ for some $t \in [\overline{t}]$. We have:

$$\mu(t) = \sum_{\substack{1 \leq j \leq w \\ s^j = t}} \lambda^j = \sum_{\substack{1 \leq j \leq w \\ r^j = \delta(t)}} \nu^j = \mu'(\delta(t)).$$

We have shown that $\overline{\mu}(\overline{t}) = \mu'(\delta(t))$. By Lemma 5, $\delta(t) = \delta(\overline{t})$, so $\overline{\mu}(\overline{t}) = \mu'(\delta(\overline{t}))$ as required.

The above argument also shows that state $\overline{s} \in \overline{S}$ is reachable in $\overline{\mathcal{M}(\mathcal{P})}$ iff $\delta(\overline{s}) \in S'$ is reachable in $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$: any chain of probability distributions from s_0 to $\overline{s}$ in $\overline{\mathcal{M}(\mathcal{P})}$ is matched by an equivalent chain of distributions from s'_0 to $\delta(\overline{s})$ in $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$. Thus, restricting δ to reachable states, the result follows. $\square$

Proof of Theorem 5, page 7 Recall from Section III-C that in the definition of $SPCTL$, the set of atoms A is taken to be the set of all expressions of the form $\mathsf{symm\text{-}expr}$.

We know from Corollary 1 that, for any $a \in A$ and $s \in S$, $s \models a \Leftrightarrow \delta(s) \models \mathfrak{h}(\phi)$, and by restriction this is also true for any $\overline{s} \in \overline{S}$.

We also know from Theorem 4 that δ is an isomorphism between $\overline{\mathcal{M}(\mathcal{P})}$ and $\mathcal{M}(\mathfrak{h}(\mathcal{P}))$ (restricted to reachable states). Therefore, by Theorem 1, $\overline{\mathcal{M}(\mathcal{P})} \models \phi \Leftrightarrow \mathcal{M}(\mathfrak{h}(\mathcal{P})) \models \mathfrak{h}(\phi)$.

Combining this with Theorem 2 we get $\mathcal{M}(\mathcal{P}) \models \phi \Leftrightarrow \mathcal{M}(\mathfrak{h}(\mathcal{P})) \models \mathfrak{h}(\phi)$ as required. $\square$