Computing Laboratory

GAME-BASED PROBABILISTIC PREDICATE ABSTRACTION IN PRISM

M. Kattenbelt M. Kwiatkowska G. Norman D. Parker

CL-RR-08-01

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Abstract

Modelling and verification of systems such as communication, network and security protocols, which exhibit both probabilistic and non-deterministic behaviour, typically use Markov Decision Processes (MDPs). For large, complex systems, abstraction techniques are essential. This paper builds on a promising approach for abstraction of MDPs based on stochastic two-player games which provides distinct lower and upper bounds for minimum and maximum probabilistic reachability properties. Existing implementations work at the model level, limiting their scalability. In this paper, we develop language-level abstraction techniques that build game-based abstractions of MDPs directly from high-level descriptions in the PRISM modelling language, using predicate abstraction and SMT solvers. For efficiency, we develop a compositional framework for abstraction. We have applied our techniques to a range of case studies, successfully verifying models larger than was possible with existing implementations. We are also able to demonstrate the benefits of adopting a compositional approach.

1 Introduction

Verification of systems that exhibit both non-deterministic and probabilistic behaviour has proved to be very useful in domains such as communication and network protocols, security protocols, and randomised distributed algorithms. Markov Decision Processes (MDPs) are a natural model for such systems and several tools, such as PRISM [13] and LiQuor [4], implement efficient solution methods for these models. As in the field of non-probabilistic model checking, however, the state space explosion problem tends to limit the scalability of these approaches and techniques to counter this are an important area of research.

Of particular current interest are the development of *abstraction techniques* for the verification of MDPs [6, 8, 17, 21]. In this paper, we use the abstraction approach of [17], which is based on stochastic two-player games. The key idea is to separate the non-determinism that is introduced by the abstraction from the non-determinism present in the original MDP. This results in abstract models that provide distinct upper and lower bounds on minimum and maximum reachability probabilities. This is in contrast to alternative abstraction methods [6], where only an upper bound on the maximum probability and a lower bound on the minimum probability can be extracted. Besides being a more informative abstraction, these bounds also provide a measure of the quality of the abstraction. This information is potentially very useful when considering refinement.

A limitation of the existing implementation in [17] is that abstractions are performed at the model level, i.e. the full concrete model (MDP) is constructed and then reduced to the corresponding stochastic game. In this paper, we develop techniques to construct the abstraction directly from a high-level description of the MDP (in this case the modelling language of PRISM) using predicate abstraction [12, 1, 5], which has been very successful in the non-probabilistic setting.

Predicate abstraction for PRISM models was recently considered in [21], but using the abstraction technique of [6] which represents abstractions as MDPs. Applying predicate abstraction to the approach of [17] provides the additional benefits of the game-based approach but proves to be more involved. This is because the game-based abstraction preserves additional information which is non-trivial to extract from language-level descriptions of PRISM models.

We present a compositional variant of game-based abstraction of MDPs, explain how to apply it at the level of the PRISM modelling language, and describe an implementation of these techniques using SMT solvers and 'on-the-fly' abstraction. We illustrate its applicability on several examples, successfully analysing models larger than is possible with the implementation of [17] and improving performance on others. We also analyse the benefits of employing a compositional approach.

The remainder of this paper is structured as follows. Section 2 provides background material, including the PRISM modelling language and its semantics. In Section 3 we present a compositional variant of the game-based abstraction method of [17]. In Section 4, we give a game-based variant of PRISM called A–PRISM and describe a predicate abstraction procedure for PRISM models that results in A–PRISM models. Sections 5 and 6 describe our implementation and present experimental results from several case studies. We conclude with a discussion of related work and ideas for future development.

2 Background

We assume a set of typed variables V. A *valuation* of V is a function s mapping each variable in V to a value in its domain. We let $\mathsf{val}(V)$ denote the set of all valuations of V and, for any $s \in \mathsf{val}(V)$ and $V' \subseteq V$, let $s\lceil_{V'}$ denote the restriction of s to the domain V'. Furthermore, if $s_1 \in \mathsf{val}(V_1)$, $s_2 \in \mathsf{val}(V_2)$ and $V_1 \cap V_2 = \emptyset$, we let $s_1 \| s_2$ denote the valuation of $V_1 \cup V_2$ where $(s_1 \| s_2)\lceil_{V_1} = s_1$ and $(s_1 \| s_2)\lceil_{V_2} = s_2$. We will often refer to valuations as *states*. We also assume a finite set Act of *actions* and an additional 'silent' action $\tau \notin Act$.

A probability distribution over a finite set S is a function $\mu : S \to [0,1]$ such that $\sum_{s\in S} \mu(s) = 1$. Let $\mathsf{Dist}(S)$ denote the set of all distributions over S. For any $s \in S$, let η_s denote the point distribution at s. If $\mu_1 \in \mathsf{Dist}(\mathsf{val}(V_1))$, $\mu_2 \in \mathsf{Dist}(\mathsf{val}(V_2))$ and $V_1 \cap V_2 = \emptyset$, let $\mu_1 \| \mu_2$ denote the distribution over $\mathsf{val}(V_1 \cup V_2)$ such that $(\mu_1 \| \mu_2)(s) = \mu_1(s\lceil_{V_1}) \cdot \mu_2(s\lceil_{V_2})$ for all $s \in \mathsf{val}(V_1 \cup V_2)$.

Definition 1 *Let V, V' be sets of variables such that $V' \subseteq V$. A* transition *from V to V'' is a tuple $\langle s, \mathsf{step}\rangle$ where $s \in \mathsf{val}(V)$ and $\mathsf{step} \subseteq (Act \cup \{\tau\}) \times \mathsf{Dist}(\mathsf{val}(V'))$.*

A transition $\langle s, \mathsf{step}\rangle$ consists of a source state s and non-deterministic choice step between pairs comprising an action and a distribution over target states. We now define (standard CSP-style) parallel composition of transitions.

Definition 2 *Suppose $V_1, V_2 \subseteq V$ are disjoint sets of variables, $\langle s, \mathsf{step}_i\rangle$ is a transition from V to V_i for $i \in \{1,2\}$ and $A \subseteq Act$. Let $\langle s, \mathsf{step}_1\rangle \,|[A]|\, \langle s, \mathsf{step}_2\rangle$ denote the transition $\langle s, \mathsf{step}\rangle$ from V to $V_1 \cup V_2$ where $\langle a, \mu\rangle \in \mathsf{step}$ if and only if one of the following conditions holds:*

1. *$a \notin A$ and $\mu = \mu_1 \| \eta_{(s\lceil_{V_2})}$ for some $\langle a, \mu_1\rangle \in \mathsf{step}_1$;*

2. $a \notin A$ *and* $\mu = \eta_{(s\lceil_{V_1})} \,\|\, \mu_2$ *for some* $\langle a, \mu_2 \rangle \in \mathsf{step}_2$*;*

3. $a \in A$ *and* $\mu = \mu_1 \,\|\, \mu_2$ *for some* $\langle a, \mu_1 \rangle \in \mathsf{step}_1$ *and* $\langle a, \mu_2 \rangle \in \mathsf{step}_2$*.*

2.1 Controlled Markov Decision Processes

The techniques introduced in this paper are for Markov Decision Processes (MDPs). However, in order to adopt a compositional approach, we use a variant called Controlled Markov Decision Processes which represent components of an MDP. These are similar to the *probabilistic modules* of [7].

Definition 3 *A* Controlled Markov Decision Process *(CMDP) is a tuple* $\mathcal{C} = \langle V, V^{\mathsf{ctrl}}, V^{\mathsf{ext}}, Act, s^{\mathsf{init}}, Steps \rangle$ *where:*

- V *is a finite set of typed variables;*
- V^{ctrl} *and* V^{ext} *partition* V *into* controlled *and* external *variables;*
- Act *is a finite set of actions;*
- $s^{\mathsf{init}} \in \mathsf{val}(V^{\mathsf{ctrl}})$ *is the initial valuation;*
- $Steps : \mathsf{val}(V) \to \mathcal{P}((Act \cup \{\tau\}) \times \mathsf{Dist}(\mathsf{val}(V^{\mathsf{ctrl}})))$ *is the transition function.*

A CMDP specifies the initial values of its controlled variables and how these variables are updated. These updates depend on the values of both its controlled variables and the external variables, which are assumed to be under the control of other components in the system. Given a valuation of all variables $s \in \mathsf{val}(V)$ the set of action-distribution pairs $Steps(s)$ represents a non-deterministic choice between several behaviours. If the $\langle a, \mu \rangle$ is chosen, then the CMDP performs action a and then probabilistically selects a new valuation of its controlled variables according to μ. The transition function can equivalently be defined as the set $\{\langle s, Steps(s) \rangle \mid s \in \mathsf{val}(V)\}$ of transitions from V to V^{ctrl}.

We now describe the parallel composition of CMDPs. CMDPs can only be combined in parallel when they agree on the total set of variables and their control variables are disjoint. We call such CMDPs *composable*. Let $\mathcal{C}_i = \langle V, V_i^{\mathsf{ctrl}}, V_i^{\mathsf{ext}}, Act_i, s_i^{\mathsf{init}}, Steps_i \rangle$ for $i \in \{1, 2\}$.

Definition 4 *The parallel composition of two composable CMDPs* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *is the CMDP* $\mathcal{C}_1 \,|[A]|\, \mathcal{C}_2 = \langle V, V^{\mathsf{ctrl}}, V^{\mathsf{ext}}, Act, s^{\mathsf{init}}, Steps \rangle$ *where:*

- $V^{\mathsf{ctrl}} = V_1^{\mathsf{ctrl}} \cup V_2^{\mathsf{ctrl}}$*;*
- $V^{\mathsf{ext}} = (V_1^{\mathsf{ext}} \cup V_2^{\mathsf{ext}}) \setminus (V_1^{\mathsf{ctrl}} \cup V_2^{\mathsf{ctrl}})$*;*
- $Act = Act_1 \cup Act_2$*;*
- $s^{\mathsf{init}} = s_1^{\mathsf{init}} \,\|\, s_2^{\mathsf{init}}$*;*
- *if* $s \in \mathsf{val}(V)$*, then* $\langle s, Steps(s) \rangle = \langle s, Steps_1(s) \rangle \,|[A]|\, \langle s, Steps_2(s) \rangle$*.*

We can also define *action renaming* and *action hiding* operations for CMDPs, but for brevity we will omit these from the presentation in this paper.

Example 2.1 *Consider the CMDP* $\mathcal{C}_{walk}$ *where* $V^{\mathsf{ctrl}}=\{val\}$*,* $V^{\mathsf{ext}}=\{close\}$*, val has domain* $\{0,\ldots,4\}$ *with initial value 2 and close has the type Boolean. For any valuation* $(v,c) \in \mathsf{val}(\{val, close\})$*,* $Steps_{\mathcal{C}_{walk}}(v,c) = \{\langle \tau, \frac{1}{2}\cdot\eta_{v-1} + \frac{1}{2}\cdot\eta_{v+1}\rangle, \langle read, \eta_v\rangle\}$ *if* $v \in \{1,2,3\}$ *and equals* $\{\langle read, \eta_v\rangle\}$ *otherwise. The CMDP models a random walk that can, at any time, do a read action. Let* $\mathcal{C}_{walk}$ *be composed with the CMDP* $\mathcal{C}_{obs}$ *in which* $V^{\mathsf{ctrl}}=\{close\}$*,* $V^{\mathsf{ext}}=\{val\}$*, close has initial value* `false`*, and, for* $(v,c) \in \mathsf{val}(\{val, close\})$*,* $Steps_{\mathcal{C}_{obs}}(v,c) = \{\langle read, \eta_{close}\rangle\}$ *if* $v{\neq}2$ *and equals* $\{\langle read, \eta_{\neg close}\rangle\}$ *otherwise. This models a CMDP which performs action read, and updates close depending on whether val is close to the ends of the walk or not. Graphical representations of these CMDPs are given in Figure 1(a).*

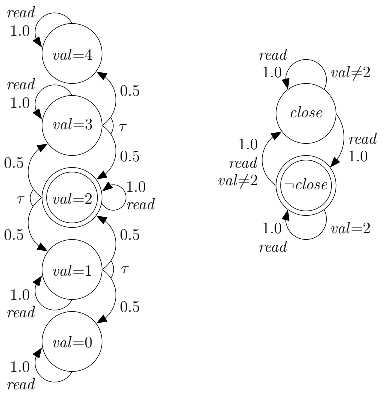
Note that a CMDP for which $V^{\mathsf{ext}} = \emptyset$ (for example the parallel composition of CMDPs whose controlled variables partition V) is simply an MDP. For an MDP, we are typically interested in quantitative aspects such as probabilistic reachability. An *adversary* of an MDP is a particular resolution of the non-determinism. Given an MDP, a valuation $s \in \mathsf{val}(V)$, a set of valuations $F \subseteq \mathsf{val}(V)$ and an adversary $\mathcal{A}$ we use $\mathbf{p}_s^{\mathcal{A}}(F)$ to denote the probability of reaching F from s under adversary $\mathcal{A}$, defined in the usual way [16]. We use $\mathbf{p}_s^{-}(F) = \inf_{\mathcal{A}} \mathbf{p}_s^{\mathcal{A}}(F)$ and $\mathbf{p}_s^{+}(F) = \sup_{\mathcal{A}} \mathbf{p}_s^{\mathcal{A}}(F)$ to denote the extremal probabilities of reaching F from s [2].

2.2 PRISM Models

We now describe the modelling language used by the PRISM [13] to describe MDPs. This language is based on guarded commands extended with probabilistic choices.

Definition 5 *A PRISM model is a tuple* $\mathbf{P} = \langle \mathsf{var}(\mathbf{P}), \mathsf{sys}, \{M_1,\ldots,M_m\}\rangle$ *consisting of a finite set of (Boolean or integer) variables* $\mathsf{var}(\mathbf{P})$*, a system definition* sys *and a finite set modules* $\{M_1,\ldots,M_m\}$*. The system definition* sys *is a process algebraic expression containing each of the m modules exactly once. Each module M consists of:*

- *a finite set of local variables* $\mathsf{var}(M) \subseteq \mathsf{var}(\mathbf{P})$ *such that:*
  - $\mathsf{var}(M)$ *are disjoint from the local variables of all other modules;*
  - *each variable* $var \in \mathsf{var}(M)$ *has the initial value* $\mathsf{init}(var)$*;*
  - $\mathsf{init}(M) \in \mathsf{val}(\mathsf{var}(M))$ *denotes the initial values of* $\mathsf{var}(M)$*;*
- *a finite set of commands* $\mathsf{com}(M)$ *where each command* $cmd \in \mathsf{com}(M)$ *includes:*
  - *a guard* $\mathsf{guard}(cmd)$ *which is a Boolean function over* $\mathsf{val}(\mathsf{var}(\mathbf{P}))$*;*
  - *an action* $\mathsf{act}(cmd)$*;*
  - *a finite set of updates* $\mathsf{updates}(cmd) = \{\langle\lambda_1, u_1\rangle,\ldots,\langle\lambda_n,u_n\rangle\}$ *such that* $\lambda_i \in (0,1]$*,* $\sum_{i=1}^{n}\lambda_i = 1$ *and* u_i *is a function from* $\mathsf{val}(\mathsf{var}(\mathbf{P}))$ *to* $\mathsf{val}(\mathsf{var}(M))$*.*



(a) CMDPs.

```
module walk

  val : [0..4] init 2;

  [] (0<val<4) → 0.5 : (val'=val−1) + 0.5 : (val'=val+1);
  [read] true → 1.0 : true;

endmodule

module obs

  close : bool init false;

  [read] (val≠2) → 1.0 : (close'=true);
  [read] (val=2) → 1.0 : (close'=false);

endmodule

system walk |[read]| obs endsystem
```

(b) PRISM syntax.

Figure 1: Simple example: a random walk and observer process (see Examples 2.1 and 2.2).

For each command cmd of a module M and valuation s of $\mathsf{var}(\mathbf{P})$, supposing $\mathsf{updates}(cmd)$ is $\{\langle\lambda_1, u_1\rangle, \ldots, \langle\lambda_n, u_n\rangle\}$, let $\mathsf{dist}(cmd, s)$ denote the distribution over $\mathsf{val}(\mathsf{var}(M))$ such that $\mathsf{dist}(cmd, s)(s') = \sum_{u_i(s)=s'} \lambda_i$ for all $s' \in \mathsf{val}(\mathsf{var}(M))$. Intuitively, $\mathsf{dist}(cmd, s)(s')$ is the probability that performing cmd in s updates the module's local variables to s'.

Definition 6 *The semantics of a module M of a PRISM model $\mathbf{P}$ is given by the CMDP $[\![M]\!] = \langle V, V^{\mathsf{ctrl}}, V^{\mathsf{ext}}, Act, s^{\mathsf{init}}, Steps\rangle$ where:*

- $V = \mathsf{var}(\mathbf{P})$, $V^{\mathsf{ctrl}} = \mathsf{var}(M)$ *and* $V^{\mathsf{ext}} = \mathsf{var}(\mathbf{P}) \setminus \mathsf{var}(M)$;
- $Act = \{\mathsf{act}(cmd) \mid cmd \in \mathsf{com}(M)\} \setminus \{\tau\}$;
- $s^{\mathsf{init}} = \mathsf{init}(M)$;
- *if* $s \in \mathsf{val}(\mathsf{var}(\mathbf{P}))$, *then* $\langle a, \mu\rangle \in Steps(s)$ *if and only if there exists* $cmd \in \mathsf{com}(M)$ *such that* $\mathsf{guard}(cmd)(s)$ *holds and* $\langle \mathsf{act}(cmd), \mathsf{dist}(cmd, s)\rangle = \langle a, \mu\rangle$.

The semantics $[\![\mathbf{P}]\!]$ of a PRISM model $\mathbf{P}$ is defined according to its system definition sys, using the semantics $[\![M]\!]$ of each individual module M, given above, and parallel composition of CMDPs (see Definition 4). We assume that $\mathsf{var}(\mathbf{P})$ is the disjoint union $\cup_{i=1}^{m} \mathsf{var}(M_i)$, and hence, for the parallel composition of all modules in a PRISM model, the set V^{ext} is empty. In other words, the semantics of a PRISM model is given by an MDP.

Example 2.2 *Figure 1(b) presents a PRISM model of the CMDP in Example 2.1.*

3 Abstraction of CMDPs

In this section we introduce abstractions of CMDPs, using the stochastic two-player game approach of [17] and predicates. A predicate φ is *over* variables V if all valuations of V uniquely determine the truth value of φ and we write $\varphi(s)$ to denote the value of φ for a valuation s of V. Given a set of predicates Φ, let $\mathsf{bool}(\Phi)$ be the set of Boolean variables indexed by the predicates in Φ, i.e. the set $\{b_\varphi \,|\, \varphi \in \Phi\}$. Furthermore, for abstraction of a particular CMDP using Φ, we will require that every predicate is either over *only* controlled variables or *only* external variables of this component. This partitions the predicates into Φ^{ctrl} and Φ^{ext}.

3.1 Abstract Controlled Markov Decision Processes

In order to present a compositional variant of game-based MDP abstraction, we introduce Abstract Controlled Markov Decision Processes (ACMDPs) which are a variant of the class of stochastic two-player games used in [17].

Definition 7 *An* Abstract Controlled Markov Decision Process *(ACMDP) is a tuple* $\mathcal{A} = \langle \overline{V}, \overline{V^{\mathsf{ctrl}}}, \overline{V^{\mathsf{ext}}}, \overline{Act}, \overline{s^{\mathsf{init}}}, \overline{Steps} \rangle$ *where:*

- $\overline{V}$ *is a set of typed variables;*
- $\overline{V^{\mathsf{ctrl}}}$ *and* $\overline{V^{\mathsf{ext}}}$ *partition* $\overline{V}$ *into controlled and external variables;*
- $\overline{Act}$ *is a finite set of actions;*
- $\overline{s^{\mathsf{init}}} \in \mathsf{val}(\overline{V^{\mathsf{ctrl}}})$ *is the initial valuation;*
- $\overline{Steps} : \mathsf{val}(\overline{V}) \to \mathcal{P}(\mathcal{P}((\overline{Act} \cup \{\tau\}) \times \mathsf{Dist}(\mathsf{val}(\overline{V^{\mathsf{ctrl}}}))))$ *is the transition function.*

The crucial difference between CMDPs and ACMDPs is that the transition function now returns *sets of sets* of action-distribution pairs. This means ACMDPs capture two levels of non-determinism: the choice of a set of action-distribution pairs, and then the choice of an element in this set. This two-level non-determinism is equivalent to that of the stochastic two-player games used in [17], where first *player 1* makes a choice, then *player 2* does, followed by a *probabilistic* choice.

We now describe the parallel composition of ACMDPs. As for CMDPs, ACMDPs can only be combined when they agree on the total set of variables and their control variables are disjoint. We call such ACMDPs *composable.* Let $\mathcal{A}_i = \langle \overline{V}, \overline{V_i^{\mathsf{ctrl}}}, \overline{V_i^{\mathsf{ext}}}, \overline{Act}_i, \overline{s_i^{\mathsf{init}}}, \overline{Steps}_i \rangle$ for $i \in \{1, 2\}$.

Definition 8 *The parallel composition of two composable ACMDPs* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *is the ACMDP* $\mathcal{A}_1 \,|[A]|\, \mathcal{A}_2 = \langle \overline{V}, \overline{V^{\mathsf{ctrl}}}, \overline{V^{\mathsf{ext}}}, \overline{Act}, \overline{s^{\mathsf{init}}}, \overline{Steps} \rangle$ *where*

- $\overline{V^{\mathsf{ctrl}}} = \overline{V_1^{\mathsf{ctrl}}} \cup \overline{V_2^{\mathsf{ctrl}}}$*;*
- $\overline{V^{\mathsf{ext}}} = (\overline{V_1^{\mathsf{ext}}} \cup \overline{V_2^{\mathsf{ext}}}) \setminus (\overline{V_1^{\mathsf{ctrl}}} \cup \overline{V_2^{\mathsf{ctrl}}})$*;*

- $\overline{Act} = \overline{Act}_1 \cup \overline{Act}_2$*;*
- $\overline{s^{\mathsf{init}}} = \overline{s_1^{\mathsf{init}}} \,\|\, \overline{s_2^{\mathsf{init}}}$*;*
- *if* $\overline{s} \in \mathsf{val}(\overline{V})$*, then* $\overline{\mathsf{step}} \in \overline{Steps}\,(\overline{s})$ *if and only if* $\langle \overline{s}, \overline{\mathsf{step}} \rangle = \langle \overline{s}, \overline{\mathsf{step}}_1 \rangle \,|[A]|\, \langle \overline{s}, \overline{\mathsf{step}}_2 \rangle$ *for some* $\overline{\mathsf{step}}_1 \in \overline{Steps}_1(\overline{s})$ *and* $\overline{\mathsf{step}}_2 \in \overline{Steps}_2(\overline{s})$*.*

Like the relation between CMDPs and MDPs, an ACMDP for which $\overline{V^{\mathsf{ext}}} = \emptyset$ is equivalent to a stochastic two-player game from [17]. A *player 1 strategy* in such an ACMDP is a particular resolution of the first non-deterministic choice of transitions in the ACMDP, whereas a *player 2 strategy* resolves the second non-deterministic choice. Given a valuation $\overline{s} \in \mathsf{val}(\overline{V})$, a set of valuations $\overline{F} \subseteq \mathsf{val}(\overline{V})$ and strategy pair σ_1, σ_2, we use $\mathbf{p}_{\overline{s}}^{\sigma_1,\sigma_2}(\overline{F})$ to denote the probability of reaching $\overline{F}$ from $\overline{s}$ under the strategies σ_1, σ_2. Like for MDPs, we define extremal values as:

$$\mathbf{p}_{\overline{s}}^{--}(\overline{F}) = \inf_{\sigma_1} \inf_{\sigma_2} \mathbf{p}_s^{\sigma_1,\sigma_2}(\overline{F}) \qquad \mathbf{p}_{\overline{s}}^{+-}(\overline{F}) = \sup_{\sigma_1} \inf_{\sigma_2} \mathbf{p}_s^{\sigma_1,\sigma_2}(\overline{F})$$

$$\mathbf{p}_{\overline{s}}^{-+}(\overline{F}) = \inf_{\sigma_1} \sup_{\sigma_2} \mathbf{p}_s^{\sigma_1,\sigma_2}(\overline{F}) \qquad \mathbf{p}_{\overline{s}}^{++}(\overline{F}) = \sup_{\sigma_1} \sup_{\sigma_2} \mathbf{p}_s^{\sigma_1,\sigma_2}(\overline{F})$$

3.2 Predicate Abstraction for CMDPs

In this section we introduce a compositional and predicate-based extension of the abstraction procedure described in [17]. Like in non-probabilistic predicate abstraction [1], we will represent an abstract state using Boolean variables $\mathsf{bool}(\Phi)$ indexed by a set of predicates Φ. We will denote abstractions with respect to Φ by $\alpha(\,\cdot\,, \Phi)$, which we now define for states, distributions, transitions and then CMDPs.

Definition 9 *Given a set of variables V and predicates Φ over V, the abstractions of a valuation $s \in \mathsf{val}(V)$ and distribution $\mu \in \mathsf{Dist}(\mathsf{val}(V))$ with respect to Φ are defined as follows:*

- $\alpha(s, \Phi)$ *is the valuation of* $\mathsf{bool}(\Phi)$ *where* $\alpha(s, \Phi)(b_\varphi) = \varphi(s)$ *for all* $\varphi \in \Phi$*;*
- $\alpha(\mu, \Phi)$ *is the distribution over* $\mathsf{val}(\mathsf{bool}(\Phi))$ *where* $\alpha(\mu, \Phi)(\overline{s}) = \sum_{\alpha(s,\Phi)=\overline{s}} \mu(s)$ *for all* $\overline{s} \in \mathsf{val}(\mathsf{bool}(\Phi))$*.*

Definition 10 *Given a set of variables V, subset $V' \subseteq V$ and sets of predicates Φ and $\Phi' \subseteq \Phi$ over V and V', the abstraction of a transition $\langle s, \mathsf{step} \rangle$ from V to V' with respect to Φ, denoted $\alpha(\langle s, \mathsf{step} \rangle, \Phi)$, is given by the transition $\langle \alpha(s, \Phi), \{ \langle a, \alpha(\mu, \Phi') \rangle \mid \langle a, \mu \rangle \in \mathsf{step} \} \rangle$ from $\mathsf{bool}(\Phi)$ to $\mathsf{bool}(\Phi')$.*

We now define an abstraction function over CMDPs. For the remainder of Section 3, we fix a CMDP $\mathcal{C} = \langle V, V^{\mathsf{ctrl}}, V^{\mathsf{ext}}, Act, s^{\mathsf{init}}, Steps \rangle$ and set of predicates Φ over V.

Definition 11 *The abstraction of CMDP $\mathcal{C}$ with respect to the predicates Φ is the ACMDP $\alpha(\mathcal{C}, \Phi) = \langle \overline{V}, \overline{V^{\mathsf{ctrl}}}, \overline{V^{\mathsf{ext}}}, Act, \overline{s^{\mathsf{init}}}, \overline{Steps} \rangle$ where:*

- $\overline{V} = \mathsf{bool}(\Phi)$*;*

- $\overline{V^{\mathsf{ctrl}}} = \mathsf{bool}(\Phi^{\mathsf{ctrl}})$;
- $\overline{V^{\mathsf{ext}}} = \mathsf{bool}(\Phi^{\mathsf{ext}})$;
- $\overline{s^{\mathsf{init}}} = \alpha(s^{\mathsf{init}}, \Phi^{\mathsf{ctrl}})$;
- *if* $\overline{s} \in \mathsf{val}(\overline{V})$, *then* $\overline{\mathsf{step}} \in \overline{Steps}\,(\overline{s})$ *if and only if there exists* $s \in \mathsf{val}(V)$ *such that* $\alpha(\langle s, Steps\,(s)\rangle, \Phi) = \langle \overline{s}, \overline{\mathsf{step}}\rangle$.

Example 3.1 *Consider the CMDP $\mathcal{C}_{walk}$ of Example 2.1 and set of predicates $\Phi = \{(val{=}0), (val{=}4), (close)\}$. Applying Definition 11, we obtain the ACMDP depicted in Figure 2(a) with $\overline{V^{\mathsf{ctrl}}} = \{b_{val=0}, b_{val=4}\}$ and $\overline{V^{\mathsf{ext}}} = \{b_{close}\}$. The states of the ACMDP are shown as rectangles and the initial state as a double rectangle. The first non-deterministic choice is represented by the circles within a state, and the second non-deterministic choice by outgoing distributions from a circle. Since the external variables have no influence, they are omitted.*

It is straightforward to show that applying Definition 11 to a CMDP for which $V^{\mathsf{ext}} = \emptyset$ yields an ACMDP (with $\overline{V^{\mathsf{ext}}} = \emptyset$) equivalent to the stochastic two-player game derived from the abstraction procedure described in [17]. Therefore the results of [17] carry over to this setting; in particular, by analysing the ACMDP $\alpha(\mathcal{C}, \Phi)$ we can obtain upper and lower bounds for both minimum and maximum reachability probabilities of the CMDP $\mathcal{C}$. More formally, given a CMDP $\mathcal{C} = \langle V, V^{\mathsf{ctrl}}, V^{\mathsf{ext}}, Act, s^{\mathsf{init}}, Steps\rangle$, valuation $s \in \mathsf{val}(V)$ and reachability objective $F \subseteq \mathsf{val}(V)$, letting $\overline{s} = \alpha(s, \Phi)$ and $\overline{F} = \{\alpha(s', \Phi) \mid s' \in F\}$ we have:

$$\mathbf{p}^{--}_{\overline{s}}(\overline{F}) \leq \mathbf{p}^{-}_{s}(F) \leq \mathbf{p}^{+-}_{\overline{s}}(\overline{F})$$
$$\mathbf{p}^{-+}_{\overline{s}}(\overline{F}) \leq \mathbf{p}^{+}_{s}(F) \leq \mathbf{p}^{++}_{\overline{s}}(\overline{F})$$

3.3 Compositional Abstraction of CMDPs

The abstraction of Definition 11 can be applied to any CMDP but, as the following example demonstrates, parallel composition and abstraction do not commute.

Example 3.2 *Consider again Example 2.1 (Figure 1(a)) and set of predicates $\Phi = \{(val{=}0), (val{=}4), (close)\}$. For the abstract valuation $\overline{s} = (\neg b_{val=0}, \neg b_{val=4}, b_{close})$ from Definition 11 it follows that $\{\langle \tau, \eta_{(\neg b_{val=0}, \neg b_{val=4})}\rangle, \langle read, \eta_{(\neg b_{val=0}, \neg b_{val=4})}\rangle\}$ and $\{\langle read, \eta_{b_{close}}\rangle\}$ are in $\overline{Steps}_{\alpha(\mathcal{C}_{obs}, \Phi)}(\overline{s})$. Therefore, by Definition 8, it follows that $\{\langle \tau, \eta_{\overline{s}}\rangle, \langle read, \eta_{\overline{s}}\rangle\}$ is in $\overline{Steps}_{\alpha(\mathcal{C}_{walk}, \Phi)\,|[read]|\,\alpha(\mathcal{C}_{obs}, \Phi)}(\overline{s})$. However, no valuation (v, c) abstracts to $\overline{s}$ and induces this transition in $\alpha(\mathcal{C}_{walk}\,|[read]|\,\mathcal{C}_{obs}, \Phi)$. More precisely, if the τ transition abstracts to a self-loop, then $v{=}2$, and if the read transition sets close to* `true`, *then $v{\neq}2$.*

As this example illustrates, a compositional abstraction may introduce spurious transitions, resulting in an over-approximation of the non-compositional abstraction and thus less precise lower and upper bounds for probabilistic reachability. Although such abstractions may still lead to useful results, we now introduce the notion of *abstraction* *preserving* CMDPs, for which compositional abstraction is precise (i.e. equivalent to the non-compositional abstraction).

Definition 12 *The CMDP $\mathcal{C}$ is called* abstraction preserving *with respect to the predicates Φ if for any $s, s' \in \mathsf{val}(V)$ such that $s\lceil_{V^{\mathsf{ctrl}}} = s'\lceil_{V^{\mathsf{ctrl}}}$ and $\alpha(s, \Phi^{\mathsf{ext}}) = \alpha(s', \Phi^{\mathsf{ext}})$, then $\alpha(\langle s, \mathit{Steps}(s)\rangle, \Phi) = \alpha(\langle s', \mathit{Steps}(s')\rangle, \Phi)$.*

Intuitively, this states that any valuations which agree on control variables and satisfy the same external predicates yield the same abstract transitions. As the following two results show, this property is both preserved under parallel composition and ensures a precise abstraction under parallel composition.

Proposition 3.3 *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be composable CMDPs and A be a set of actions. If $\mathcal{C}_1$ and $\mathcal{C}_2$ are abstraction preserving with respect to the predicates Φ, then their composition $\mathcal{C}_1 \,|[A]|\, \mathcal{C}_2$ is also abstraction preserving with respect to Φ.*

Proposition 3.4 *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be composable CMDPs and A be a set of actions. If $\mathcal{C}_1$ and $\mathcal{C}_2$ are abstraction preserving with respect to the predicates Φ, then:*

$$\alpha(\mathcal{C}_1, \Phi) \,|[A]|\, \alpha(\mathcal{C}_2, \Phi) = \alpha(\mathcal{C}_1 \,|[A]|\, \mathcal{C}_2, \Phi)\,.$$

From Proposition 3.3 and Proposition 3.4, we can infer that a compositional abstraction is precise if each individual component is abstraction preserving.

Example 3.5 *Consider the CMDP $\mathcal{C}_{obs}$ from Example 2.1 and set of predicates $\Phi = \{(val{=}0), (val{=}4), (close)\}$. This CMDP is not abstraction preserving with respect to Φ. For example, the valuations $s = (2, \mathtt{true})$ and $s' = (3, \mathtt{true})$ agree on the value of close and $\alpha(s, \Phi^{\mathsf{ext}}) = \alpha(s', \Phi^{\mathsf{ext}}) = (\neg b_{val=0}, \neg b_{val=4})$. However, $\alpha(\langle s, \mathit{Steps}_{\mathcal{C}_{obs}}(s)\rangle, \Phi) = \langle \alpha(s, \Phi), \{\langle read, \eta_{\neg b_{close}}\rangle\}\rangle$ while $\alpha(\langle s', \mathit{Steps}_{\mathcal{C}_{obs}}(s')\rangle, \Phi) = \langle \alpha(s', \Phi), \{\langle read, \eta_{b_{close}}\rangle\}\rangle$. If we extend Φ with the predicate $(val{=}2)$, then $\mathcal{C}_{obs}$ is abstraction preserving.*

4 Abstraction of PRISM Models

Suppose we wish to abstract a PRISM model. One possibility is to (compositionally or non-compositionally) apply the abstraction method of Section 3 to its CMDP semantics. In either case, the disadvantage of such a method is that the concrete CMDPs have to be constructed, limiting the applicability of the approach. In this section we define a language-level abstraction method to remedy the situation.

4.1 A–PRISM Models

For our language-level abstraction, we introduce the A–PRISM language, an extension of the PRISM language with an additional element of choice.

Definition 13 *An A–PRISM model is a tuple $\mathbf{A} = \langle \overline{\mathsf{var}}(\mathbf{A}), \overline{\mathsf{sys}}, \{\overline{M}_1, \ldots, \overline{M}_m\}\rangle$. The only difference between this and a PRISM model is the definition of the commands $\overline{\mathsf{com}}(\overline{M})$ for each module $\overline{M}$. Each command $\overline{cmd} \in \overline{\mathsf{com}}(\overline{M})$ includes:*

- *a guard* $\overline{\mathsf{guard}}(\overline{cmd})$ *which is a Boolean function over* $\mathsf{val}(\overline{\mathsf{var}}(\mathbf{A}))$*;*

- *a finite set of choices* $\overline{\mathsf{choices}}(\overline{cmd})$ *where each* $\overline{chc} \in \overline{\mathsf{choices}}(\overline{cmd})$ *consists of an action* $\overline{\mathsf{act}}(\overline{chc})$ *and a finite set of updates* $\overline{\mathsf{updates}}(\overline{chc}) = \{\langle \lambda_1, \overline{u}_1\rangle, \ldots, \langle \lambda_n, \overline{u}_n\rangle\}$ *such that* $\lambda_i \in (0,1]$, $\sum_{i=1}^{n} \lambda_i = 1$ *and* $\overline{u}_i$ *is a function from* $\overline{\mathsf{var}}(\mathbf{A})$ *to* $\overline{\mathsf{var}}(\overline{M})$*.*

For a choice $\overline{chc}$ of a command and valuation $\overline{s} \in \mathsf{val}(\overline{\mathsf{var}}(\mathbf{A}))$, supposing $\overline{\mathsf{updates}}(\overline{chc}) = \{\langle \lambda_1, \overline{u}_1\rangle, \ldots, \langle \lambda_n, \overline{u}_n\rangle\}$, let $\overline{\mathsf{dist}}(\overline{chc}, \overline{s})$ denote the distribution over $\overline{\mathsf{var}}(\overline{M})$ such that $\overline{\mathsf{dist}}(\overline{chc}, \overline{s})(\overline{s}') = \sum_{\overline{u}_i(\overline{s})=\overline{s}'} \lambda_i$ for all $\overline{s}' \in \mathsf{val}(\overline{\mathsf{var}}(\overline{M}))$.

Definition 14 *The semantics of a module* $\overline{M}$ *of A–PRISM model* $\mathbf{A}$ *is given by the ACMDP* $[\![\overline{M}]\!] = \langle \overline{V}, \overline{V}^{\mathsf{ctrl}}, \overline{V}^{\mathsf{ext}}, \overline{Act}, \overline{s}^{\mathsf{init}}, \overline{Steps}\rangle$*, where:*

- $\overline{V} = \overline{\mathsf{var}}(\mathbf{A})$*,* $\overline{V}^{\mathsf{ctrl}} = \overline{\mathsf{var}}(\overline{M})$ *and* $\overline{V}^{\mathsf{ext}} = \overline{\mathsf{var}}(\mathbf{A}) \setminus \overline{\mathsf{var}}(\overline{M})$*;*

- $\overline{Act} = \{\overline{\mathsf{act}}(\overline{chc}) \mid \overline{cmd} \in \overline{\mathsf{com}}(\overline{M}),\ \overline{chc} \in \overline{\mathsf{choices}}(\overline{cmd})\} \setminus \{\tau\}$*;*

- $\overline{s^{\mathsf{init}}} = \overline{\mathsf{init}}(\overline{M})$*;*

- *if* $\overline{s} \in \mathsf{val}(\overline{\mathsf{var}}(\mathbf{A}))$*, then* $\overline{\mathsf{step}} \in \overline{Steps}(\overline{s})$ *if and only if there exists a command* $\overline{cmd} \in \overline{\mathsf{com}}(M)$ *such that* $\overline{\mathsf{step}} = \{\langle \overline{\mathsf{act}}(\overline{chc}), \overline{\mathsf{dist}}(\overline{chc}, \overline{s})\rangle \mid \overline{chc} \in \overline{\mathsf{choices}}(\overline{cmd})\}$ *and* $\overline{\mathsf{guard}}(\overline{cmd})(\overline{s})$ *holds.*

The semantics $[\![\mathbf{A}]\!]$ of an A–PRISM model $\mathbf{A}$ is defined according to its system definition $\overline{\mathsf{sys}}$, using the semantics $[\![\overline{M}]\!]$ of each individual module $\overline{M}$, given above, and parallel composition of ACMDPs (see Definition 8).

The 'first' non-deterministic choices of $[\![\overline{M}]\!]$ are caused by overlaps between the guards of commands, whereas the 'second' non-deterministic choices are induced by the choices within commands.

Example 4.1 *Figure 2 shows an A–PRISM module and its ACMDP semantics.*

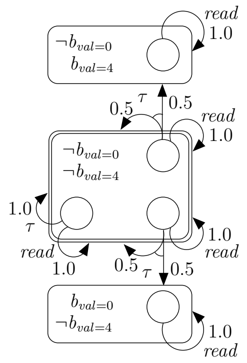
4.2 Language-level Abstraction of PRISM Models

In this section we introduce a language-level abstraction method for PRISM. We assume a fixed PRISM model $\mathbf{P} = \langle \mathsf{var}(\mathbf{P}), \mathsf{sys}, \{M_1, \ldots, M_m\}\rangle$ and a set of predicates Φ which is partitioned into subsets $\Phi^{M_1}, \ldots, \Phi^{M_m}$ over the local variables of the modules $M_1, \ldots, M_m$. The abstraction of $\mathbf{P}$ is defined as the A–PRISM model:

$$\beta(\mathbf{P}, \Phi) = \langle \mathsf{bool}(\Phi), \beta(\mathsf{sys}), \{\beta(M_1, \Phi), \ldots, \beta(M_m, \Phi)\}\rangle$$

where the system definition $\beta(\mathsf{sys})$ is a syntactic copy of sys and each module M is replaced by the language-level abstraction $\beta(M, \Phi)$, defined below.

Definition 15 *The language-level abstraction of a module* M *of* $\mathbf{P}$ *is the A–PRISM module* $\beta(M, \Phi)$ *where:*



(a) ACMDP.

```
module walk

  b_val0  : bool init false;
  b_val4  : bool init false;

  (!b_val0&!b_val4) [read] → 1.0 : true;
                    [] → 1.0 : true;
  (!b_val0&!b_val4) [read] → 1.0 : true;
                    [] → 0.5 : (b_val0′=true) + 0.5 : true;
  (!b_val0&!b_val4) [read] → 1.0 : true;
                    [] → 0.5 : true + 0.5 : (b_val4′=true);
  (b_val0&!b_val4)  [read] → 1.0 : true;
  (!b_val0&b_val4)  [read] → 1.0 : true;

endmodule
```

(b) A–PRISM syntax.

Figure 2: Abstraction of the random walk component of Figure 1 (see Example 3.1).

- *the local variables* $\overline{\mathsf{var}}(\beta(M,\Phi))$ *are* $\mathsf{bool}(\Phi^M)$*;*
- *the initial value* $\overline{\mathsf{init}}(b_\varphi)$ *equals* $\varphi(\mathsf{init}(M))$ *for all* $b_\varphi \in \mathsf{bool}(\Phi^M)$*;*
- *the set of commands* $\overline{\mathsf{com}}(\beta(M,\Phi))$ *equals* $\{\overline{cmd_s} \mid s \in \mathsf{val}(\mathsf{var}(\mathbf{P}))\}$ *where*
  - $\overline{\mathsf{guard}}(\overline{cmd_s}) = \bigwedge_{\varphi\in\Phi} (b_\varphi = \varphi(s))$,
  - $\overline{\mathsf{choices}}(\overline{cmd_s}) = \{\overline{cmd} \mid cmd \in \mathsf{com}(M), \mathsf{guard}(cmd)(s)\}$ *with*
    - $\overline{\mathsf{act}}(\overline{cmd}) = \mathsf{act}(cmd)$
    - *if* $\mathsf{updates}(cmd) = \{\langle\lambda_1,u_1\rangle,\ldots,\langle\lambda_n,u_n\rangle\}$ *and* $\overline{u}$ *is the constant function that returns* $\alpha(u(s),\Phi^M)$*, then* $\overline{\mathsf{updates}}(\overline{cmd}) = \{\langle\sum_{\overline{u_j}=\overline{u_i}}\lambda_j,\overline{u_i}\rangle \mid 1 \le i \le n\}$*.*

The results below illustrate that using the language-level method we obtain the same abstraction as with the model-level abstraction of Section 3.2.

Proposition 4.2 *If* M *is a module of* $\mathbf{P}$*, then* $[\![\beta(M,\Phi)]\!] = \alpha([\![M]\!],\Phi)$*.*

Combining this with the results of the previous sections we have the following.

Theorem 4.3 *If* $[\![M]\!]$ *is abstract preserving with respect to* Φ *for each module* M *of* $\mathbf{P}$*, then* $[\![\beta(\mathbf{P},\Phi)]\!] = \alpha([\![\mathbf{P}]\!],\Phi)$*.*

Figure 4.2 presents an overview of the correspondence between the model-level (α) and language-level (β) abstractions.

The remaining question is how to check abstraction preservation at the language level. We now outline a simple check which guarantees this. It is always possible to rewrite a single PRISM command into commands with disjoint guards such that the updates only contain local variables. Now, if we have a PRISM module containing only commands of this form, then to check its semantics is abstraction preserving it is sufficient to show that if $s,s' \in \mathsf{val}(\mathsf{var}(\mathbf{P}))$ such that $s\lceil_{\mathsf{var}(M)} = s'\lceil_{\mathsf{var}(M)}$ and $\alpha(s,\Phi) = \alpha(s',\Phi)$, then $\mathsf{guard}(cmd)(s) = \mathsf{guard}(cmd)(s')$ for all commands cmd.

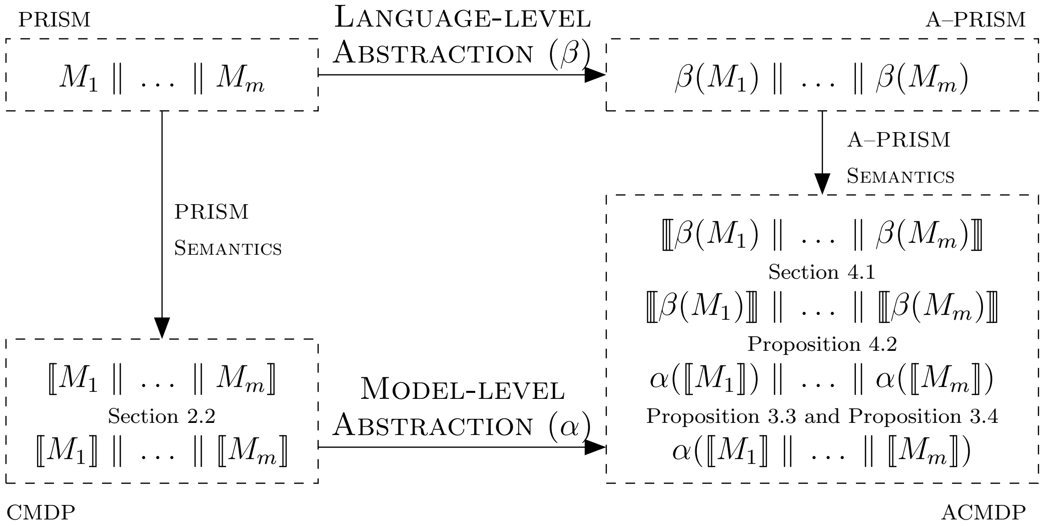


Figure 3: Relation between model-level and language-level abstraction functions.

5 Implementation

We have built prototype tools both for our language-level abstraction (translation from PRISM to A–PRISM) and model checking A–PRISM models. The model checker is a relatively simple extension of PRISM's MTBDD model checking engine for MDPs. In the remainder of this section we will discuss the abstraction tool.

5.1 Abstraction with SMT Solvers

The key step in the translation of a (concrete) PRISM model to an (abstract) A–PRISM model is the construction of abstract commands, as described in Definition 15. For this, the implementation uses ALL-SAT procedures over the theories of integer arithmetic and fixed-size bit-vectors through the SMT solver Yices [10]. This is based on the principles described in [5, 19] for predicate abstraction of non-probabilistic systems.

In Definition 15 each abstract command is induced by a concrete valuation and the concrete commands enabled for this valuation. However, considering each concrete valuation individually is clearly inefficient. Our implementation therefore employs an approach which detects multiple valuations inducing identical commands. The basic idea is to instead enumerate what we call *overlaps*, which are combinations of commands that can be simultaneously enabled.

Formally, an overlap of module M from a PRISM model $\mathbf{P}$ is a set of commands $\mathcal{O} \subseteq \mathsf{com}(M)$ for which there exists $s \in \mathsf{val}(\mathsf{var}(\mathbf{P}))$ such that $cmd \in \mathcal{O}$ if and only if $\mathsf{guard}(cmd)(s)$. Given a module M, we first find all overlaps of M with an ALL-SAT procedure[1]. Then, for a given overlap $\mathcal{O}$, we find the corresponding abstract commands, again with an ALL-SAT procedure. To optimise this approach, we remove unnecessary

[1]The algorithm to check if a PRISM module is abstraction preserving can be implemented similarly.

predicates both from the guards and updates of abstract commands. For example, we do not include any predicates in an abstract update if the corresponding concrete updates do not influence their values.

5.2 'On-the-Fly' Abstraction

During prototyping, our implementation would often find a large number of overlaps, making the ALL-SAT procedures infeasible. However, further investigation revealed that the majority of these overlaps were induced by unreachable concrete valuations. Therefore, the prototype was extended with an 'on-the-fly' abstraction method to overcome this problem. Like in explicit-state model checking, this is achieved by keeping a stack of reachable abstract valuations of $\mathsf{bool}(\Phi^M)$. Initially, this stack only contains the element $\alpha(s^{\mathsf{init}}, \Phi^M)$. The method takes individual abstract valuations off the stack, constructs the abstract commands for this valuation and adds any new abstract valuations that are the target of this command to the stack. Note that, since the tool now constructs abstract commands for each abstract state $\overline{s}$ separately, only commands that are enabled for some valuation s such that $\alpha(s, \Phi^M) = \overline{s}$ need be considered when searching for overlaps of these commands.

Although this 'on-the-fly' abstraction method does perform reachability over abstract states, it is important to stress that, unlike [17], it does not require the construction of the reachable concrete state space or take into account whether concrete states are reachable.

6 Experimental Results

We have tested the performance of our implementation on three case studies:[2]

- An extension of the sliding window protocol of [20] where channels lose messages probabilistically instead of non-deterministically and a notion of timeout is included. We fix the window size of the sender (2) and receiver (1), buffer size of the channels (2) and sequence numbers (modulo 4) while varying the number of data frames (D) in the source. We analyse '*the maximum probability of sending D data frames without a timeout*' using an abstraction that removes the values of the data frames. In the compositional approach, we abstract the sender and data channel separately from the receiver and acknowledgement channel.

- IPv4 Zeroconf protocol [3], as described in [17], parameterised by the number of configured hosts (N) and with 64 IP addresses. We encode the abstraction of [17] into predicates and consider '*the minimum probability that the host eventually secures an IP address*'. In the compositional approach, the configuring host is abstracted separately from the channel and configured hosts.

- Israeli and Jalfon's self-stabilisation protocol [15] for a ring with N processes. We encode the abstraction of [9] into predicates and analyse '*the minimum probability*

[2]Files for the case studies are available from `http://www.prismmodelchecker.org/files/qapl08/`.

		Concrete Model			Abstract Models					
					Non-compositional		Compositional			
		Num. comm.	Num. states	Check time	Abstr. time	Num. Comm.	Abstr. time	Num. Comm.	Num. states	Check time
Sliding Win. (D)	8	19	189,952	30.2	96.7	540	220	3,260	742	0.21
	10	19	987,136	153	126	706	336	4,870	964	0.43
	12	19	–	–	155	872	473	6,545	1,186	0.69
	14	19	–	–	200	1,038	630	8,225	1,408	1.23
	16	19	–	–	237	1,204	819	9,905	1,630	1.68
	18	19	–	–	285	1,370	962	11,585	1,852	2.47
	20	19	–	–	334	1,536	1,201	13,265	2,074	3.45
Zeroconf (N)	4	89	50,377	206	1,110	2,349	106	362	1,325	104
	5	109	113,217	355	1,480	2,523	183	396	1,421	134
	6	129	282,185	678	2,480	2,695	262	431	1,517	161
	7	149	426,529	952	3,630	2,762	434	444	1,549	175
	8	169	838,905	1,400	6,370	2,804	785	453	1,581	209
Israeli & Jalfon (N)	8	8	255	0.01	13.4	28	n/a	n/a	22	<0.01
	10	10	1,023	0.03	95.2	68	n/a	n/a	42	<0.01
	12	12	4,095	0.08	727	168	n/a	n/a	77	<0.01
	14	14	16,383	0.22	4,210	415	n/a	n/a	135	<0.01
	16	16	65,535	0.75	28,000	1,025	n/a	n/a	231	<0.01
	18	18	262,143	2.27	136,000	2,505	n/a	n/a	385	<0.01
	20	20	1,048,575	8.51	1,090,000	6,056	n/a	n/a	627	0.02

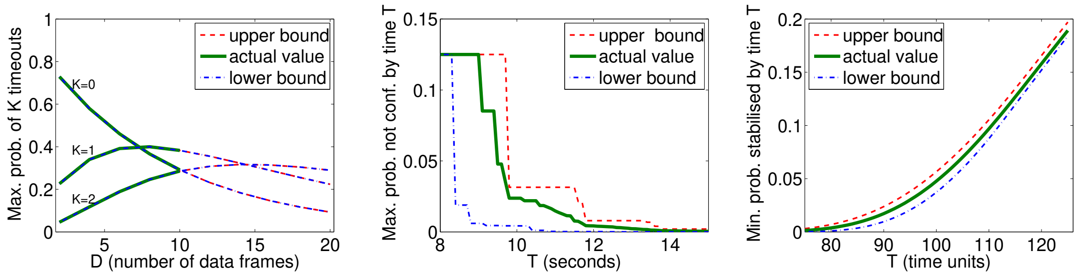
Figure 4: Experimental results of compositional and non-compositional language-level abstraction.

that the ring eventually stabilises'. Since each predicate refers to variables from all components of the system, this model cannot be abstracted compositionally.

Figure 4 presents a summary of the performance for each case study. For the full concrete model, we give the number of states, the number of PRISM commands and the time required to perform model checking in PRISM (to ensure a fair comparison, we use PRISM's MTBDD engine). For the abstract model, we show the number of states, the time to perform model checking with our prototype and, for each of the two abstraction approaches (non-compositional and compositional), the number of A–PRISM commands and the time required for abstraction. All times are in seconds and experiments were run on an AMD Athlon 4600+ with 2GB RAM.

Figure 5 gives quantitative results obtained from the models: lower and upper bounds from the abstract model and, where possible, exact answers from the concrete model. This, together with Figure 4, confirms that the game-based abstraction works well, in all cases providing tight lower and upper bounds from relatively small abstract models.

A key observation from the results is that we successfully managed to analyse models (for the sliding window protocol) larger than is possible with the model-level implementation of game-based abstraction from [17]. Furthermore, for the larger Zeroconf models, building and checking the abstraction is more efficient than checking the full model. In all



(a) Sliding window protocol. (b) Zeroconf (N=8). (c) Israeli and Jalfon (N=20).

Figure 5: Quantitative results obtained using language-level game-based abstraction.

cases, the use of 'on-the-fly' abstraction is essential to make the abstraction process feasible. This is because the number of potential overlaps, if reachability is not considered, is prohibitively large. The worst performance is observed for the Israeli & Jalfon's protocol. For this model, the generation of abstract commands described in Section 5.1 needs to consider every concrete state in the model (the worst possible scenario), resulting in a large number of calls to the SMT solver and thus a very slow abstraction time.

As regards a comparison of the compositional and non-compositional approaches to abstraction, we observe varied results. For Zeroconf models, the compositional abstraction significantly outperforms the non-compositional one, both in terms of abstraction time and A–PRISM model size, but for the sliding window protocol the reverse is true. In fact, this is due to the general suitability of the models to a compositional analysis. For the sliding window protocol, each component makes no assumptions about the content (and ordering) of incoming messages and thus, when considered in isolation, its (concrete or abstract) state space is much larger. This makes the compositional approach perform poorly. What is very encouraging, however, is that for models which can be decomposed without such a blow-up (such as Zeroconf) our composition approach can exploit this and performs much better.

7 Related Work

Practical approaches for abstracting MDPs are presented in [6, 17], the former using MDPs themselves as abstract models and the latter using stochastic two-player games. In [6] the tool RAPTURE is presented which performs successive abstractions and refinements for checking bounds on reachability probabilities. In [17], a prototype implementation is used to construct abstract models from the corresponding MDPs and partition of the state space and compute upper and lower bounds on reachability probabilities.

Predicate abstraction techniques [12] are prevalent in non-probabilistic verification. In the probabilistic case, the only other work we are aware of is [21] which introduces the PASS tool for language-level abstraction of PRISM models using the abstract approach of [6]. Like to our approach, PASS employs an SMT solver in the abstraction procedure. A key difference, however, is our use of stochastic two-player games rather than MDPs.

While this will in general provide a more useful abstraction, it is also more difficult to apply to predicate abstraction. In [21] each command of a PRISM module can be abstracted separately. Here, as described in Section 5, we must consider overlaps between commands in order to distinguish between the two types of non-determinism. To improve efficiency, we also adopt a compositional approach to abstraction and use 'on-the-fly' techniques.

Also relevant is the 'magnifying-lens abstraction' (MLA) approach of [8], which computes lower and upper bounds for PCTL formulae on MDPs. This is done by partitioning the state space into regions and analysing each region separately. It is still necessary to build the full MDP, however. Finally, approaches have also been proposed for abstracting discrete-time Markov chains [11, 14], using interval-based extensions of Markov chains, but no implementations or results were presented.

8 Conclusions

We have introduced a method to obtain stochastic two-player game abstractions of MDPs, directly from high-level model descriptions in the PRISM language. Our approach is based on a compositional reformulation of the abstraction techniques from [17] and the use of predicate abstraction. Although a compositional abstraction is potentially an over-approximation (compared to the non-compositional version), we provide conditions which guarantee a precise abstraction. We have developed an implementation of our techniques based on the SMT solver Yices and present experimental results from a range of case studies, illustrating how our work can generate game-based abstractions for larger models than was previously possible. We also highlight the benefits of adopting a compositional approach.

In the future, we hope to improve the performance of our tool chain using symbolic decision procedures [18]. We also plan to integrate this with ongoing work to develop an abstraction-refinement loop for MDP verification. Finally, we also intend to extend the current method to imperative programming languages.

References

[1] T. Ball, T. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. *SIGPLAN Notices*, 36(5):203–213, 2001.

[2] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. S. Thiagarajan, editor, *Proc. 15th Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTC'95)*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[3] S. Cheshire, B. Adoba, and E. Gutterman. Dynamic configuration of IPv4 link local addresses. Available from http://www.ietf.org/rfc/rfc3927.txt.

[4] F. Ciesinski and C. Baier. LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *Proc. 3rd Int. Conf. Quantitative Evaluation of Systems (QEST'06)*, pages 131–132. IEEE CS, 2006.

[5] E. Clarke, D. Kroening, N. Sharygina, and K. Yorav. Predicate abstraction of ANSI-C programs using SAT. *Formal Methods in System Design*, 25:105–127, 2004.

[6] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In L. de Alfaro and S. Gilmore, editors, *Proc. 1st Joint Int. Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'01)*, volume 2165 of *LNCS*, pages 39–56. Springer, 2001.

[7] L. de Alfaro, T. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In K. Larsen and M. Nielsen, editors, *Proc. 12th Int. Conf. Concurrency Theory (CONCUR'01)*, volume 2154 of *LNCS*, pages 351–365. Springer, 2001.

[8] L. de Alfaro and P. Roy. Magnifying-lens abstraction for Markov decision processes. In W. Damm and H. Hermanns, editors, *Proc. 19th Int. Conf. Computer Aided Verification (CAV'07)*, volume 4590 of *LNCS*, pages 325–338. Springer, 2007.

[9] M. Duflot, L. Fribourg, and C. Picaronny. Randomized finite-state distributed algorithms as Markov chains. In J. Welch, editor, *Proc. 15th Int. Conf. Distributed Computing (DISC'01)*, volume 2180 of *LNCS*, pages 240–254. Springer, 2001.

[10] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In T. Ball and R. Jones, editors, *Proc. 18th Int. Conf. Computer Aided Verification (CAV'06)*, volume 4114 of *LNCS*, pages 81–94. Springer, 2006.

[11] H. Fecher, M. Leucker, and V. Wolf. Don't know in probabilistic systems. In A. Valmari, editor, *Proc. 13th Int. SPIN Workshop (SPIN'06)*, volume 3925 of *LNCS*, pages 71–88. Springer, 2006.

[12] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *Proc. 9th Int. Conf. Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

[13] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[14] M. Huth. On finite-state approximants for probabilistic computation tree logic. *Theoretical Computer Science*, 346(1):113–134, 2005.

[15] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proc. 9th ACM Symp. Principles of Distributed Computing (PODC'90)*, pages 119–131. ACM, 1990.

[16] J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Denumerable Markov Chains*. Springer-Verlag, 2 edition, 1976.

[17] M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for Markov decision processes. In *Proc. 3rd Int. Conf. Quantitative Evaluation of Systems (QEST'06)*, pages 157–166. IEEE CS, 2006.

[18] S. Lahiri, T. Ball, and B. Cook. Predicate abstraction via symbolic decision procedures. In K. Etessami and S. Rajamani, editors, *Proc. 17th Int. Conf. on Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*, pages 24–38. Springer, 2005.

[19] S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras. SMT techniques for fast predicate abstraction. In T. Ball and R. B. Jones, editors, *Proc. 18th Int. Conf. Computer Aided Verification (CAV'06)*, volume 4144 of *LNCS*, pages 424–437. Springer, 2006.

[20] K. Stahl, K. Baukus, Y. Lakhnech, and M. Steffen. Divide, abstract, and model-check. In D. Dams, R. Gerth, S. Leue, and M. Massink, editors, *Proc. 5th and 6th Int. SPIN Workshops (SPIN'99)*, volume 1680 of *LNCS*, pages 57–76. Springer, 1999.

[21] B. Wachter, L. Zhang, and H. Hermanns. Probabilistic model checking modulo theories. In *Proc. 4th Int. Conf. Quantitative Evaluation of Systems (QEST'07)*, pages 119–128. IEEE CS, 2007.

Appendices

A Proofs of Section 3.3

In this section we give the proofs of Proposition 3.3 and Proposition 3.4 which we first recall.

Proposition 3.3 *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be composable CMDPs and A be a set of actions. If $\mathcal{C}_1$ and $\mathcal{C}_2$ are abstraction preserving with respect to the predicates Φ, then their composition $\mathcal{C}_1 \,|[A]|\, \mathcal{C}_2$ is also abstraction preserving with respect to Φ.*

Proposition 3.4 *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be composable CMDPs and A be a set of actions. If $\mathcal{C}_1$ and $\mathcal{C}_2$ are abstraction preserving with respect to the predicates Φ, then*

$$\alpha(\mathcal{C}_1, \Phi) \,|[A]|\, \alpha(\mathcal{C}_2, \Phi) = \alpha(\mathcal{C}_1 \,|[A]|\, \mathcal{C}_2, \Phi)\,.$$

Before we give the proof, we require a number of preliminary lemmas. In the following, we assume that $V_1 \subseteq V$ and $V_2 \subseteq V$ are disjoint sets of variables, and $\Phi_1 \subseteq \Phi$ and $\Phi_2 \subseteq \Phi$ are set of predicates over V_1 and V_2.

Lemma A.1 *If s_1 and s_2 are valuations of V_1 and V_2, then*

$$\alpha(s_1, \Phi_1) \,\|\, \alpha(s_2, \Phi_2) = \alpha(s_1 \,\|\, s_2, \Phi_1 \cup \Phi_2)\,.$$

Proof. The result follows from showing that for any predicate $\varphi \in \Phi_1 \cup \Phi_2$:

$$\big(\alpha(s_1, \Phi_1) \,\|\, \alpha(s_2, \Phi_2)\big)(b_\varphi) = \alpha(s_1 \,\|\, s_2, \Phi_1 \cup \Phi_2)(b_\varphi).$$

Therefore, consider any $\varphi \in \Phi_1 \cup \Phi_2$, if $\varphi \in \Phi_1$, since by the hypothesis $V_1 \cap V_2 = \emptyset$ it follows by definition of $\|$ on valuations that:

$$\begin{aligned}
\big(\alpha(s_1, \Phi_1) \,\|\, \alpha(s_2, \Phi_2)\big)(b_\varphi) \;&=\; \alpha(s_1, \Phi_1)(b_\varphi) && \\
=\; \varphi(s_1) & && \text{by Definition 9} \\
=\; \varphi(s_1 \,\|\, s_2) & && \text{by definition of } \| \text{ on valuations} \\
=\; \alpha(s_1 \,\|\, s_2, \Phi_1 \cup \Phi_2)(b_\varphi) & && \text{by Definition 9.}
\end{aligned}$$

An analogous result holds if $\varphi \in \Phi_2$ and, since these are the only cases to consider, this completes the proof. $\square$

Lemma A.2 *If μ_1 and μ_2 are distributions over valuations of V_1 and V_2, then*

$$\alpha(\mu_1, \Phi_1) \,\|\, \alpha(\mu_2, \Phi_2) = \alpha(\mu_1 \,\|\, \mu_1, \Phi_1 \cup \Phi_2)\,.$$

Proof. We prove the lemma by showing that for any $\overline{s} \in \mathsf{val}(\mathsf{bool}(\Phi_1 \cup \Phi_2))$:

$$\big(\alpha(\mu_1, \Phi_1) \,\|\, \alpha(\mu_2, \Phi_2)\big)(\overline{s}) = \alpha(\mu_1 \,\|\, \mu_1, \Phi_1 \cup \Phi_2)(\overline{s})\,.$$

Therefore consider any $\overline{s} \in \mathsf{val}(\mathsf{bool}(\Phi_1 \cup \Phi_2))$, by the hypothesis Φ_1 and Φ_2 are predicates over V_1 and V_2 and $V_1 \cap V_2 = \emptyset$, and hence we can write $\overline{s}$ as $\overline{s}_1 \parallel \overline{s}_2$ where $\overline{s}_1 \in \mathsf{val}(\mathsf{bool}(\Phi_1))$ and $\overline{s}_2 \in \mathsf{val}(\mathsf{bool}(\Phi_2))$. By definition of $\parallel$ on distributions it follows that:

$$\begin{aligned}
&\big(\alpha(\mu_1,\Phi_1) \parallel \alpha(\mu_2,\Phi_2)\big)(\overline{s}) = \alpha(\mu_1,\Phi_1)(\overline{s}_1) \cdot \alpha(\mu_2,\Phi_2)(\overline{s}_2) \\
&= \left(\sum_{\alpha(s_1,\Phi_1)=\overline{s}_1} \mu_1(s_1)\right) \cdot \left(\sum_{\alpha(s_2,\Phi_2)=\overline{s}_2} \mu_2(s_2)\right) && \text{by Definition 9} \\
&= \sum_{\alpha(s_1,\Phi_1)=\overline{s}_1} \; \sum_{\alpha(s_2,\Phi_2)=\overline{s}_2} \mu_1(s_1) \cdot \mu_2(s_2) && \text{rearranging} \\
&= \sum_{\alpha(s_1,\Phi_1) \parallel \alpha(s_2,\Phi_2)=\overline{s}_1 \parallel \overline{s}_2} (\mu_1 \parallel \mu_2)(s_1 \parallel s_2) && \text{by definition of } \parallel \text{ on distributions} \\
&= \sum_{\alpha(s_1 \parallel s_2,\Phi_1 \cup \Phi_2)=\overline{s}_1 \parallel \overline{s}_2} (\mu_1 \parallel \mu_2)(s_1 \parallel s_2) && \text{by Lemma A.1} \\
&= \alpha(\mu_1 \parallel \mu_1, \Phi_1 \cup \Phi_2)(\overline{s}_1 \parallel \overline{s}_2) && \text{by Definition 9} \\
&= \alpha(\mu_1 \parallel \mu_1, \Phi_1 \cup \Phi_2)(\overline{s}) && \text{by construction of } \overline{s}_1 \parallel \overline{s}_2.
\end{aligned}$$

Hence, since $\overline{s} \in \mathsf{val}(\mathsf{bool}(\Phi_1 \cup \Phi_2))$ was arbitrary, the lemma holds. $\square$

Lemma A.3 *If $\langle s, \mathsf{step}_1\rangle$ and $\langle s, \mathsf{step}_2\rangle$ are transitions from V to V_1 and V_2 respectively, then for any set of actions A:*

$$\alpha(\langle s, \mathsf{step}_1\rangle, \Phi) \,|[A]|\, \alpha(\langle s, \mathsf{step}_2\rangle, \Phi) = \alpha\big(\langle s, \mathsf{step}_1\rangle \,|[A]|\, \langle s, \mathsf{step}_2\rangle, \Phi\big)\,.$$

Proof. Let $\langle \overline{s}_{\alpha(\parallel)}, \overline{\mathsf{step}}_{\alpha(\parallel)}\rangle = \alpha(\langle s, \mathsf{step}_1\rangle, \Phi) \,|[A]|\, \alpha(\langle s, \mathsf{step}_2\rangle, \Phi)$ and $\langle \overline{s}_{\alpha\parallel\alpha}, \overline{\mathsf{step}}_{\alpha\parallel\alpha}\rangle = \alpha(\langle s, \mathsf{step}_1\rangle \,|[A]|\, \langle s, \mathsf{step}_2\rangle, \Phi)$. Using Lemma A.1 it follows that $\overline{s}_{\alpha\parallel\alpha} = \overline{s}_{\alpha(\parallel)} = \alpha(s,\Phi)$, and hence is remains to show that $\overline{\mathsf{step}}_{\alpha\parallel\alpha} = \overline{\mathsf{step}}_{\alpha(\parallel)}$. Therefore consider any $\langle act, \mu\rangle \in \overline{\mathsf{step}}_{\alpha\parallel\alpha}$ by the hypothesis, Definition 4 and Definition 11 one of the following cases must hold:

- $act \notin A$ and $\mu = \alpha(\mu_1 \parallel \eta_{s\lceil_{V_2}}, \Phi)$ and $\langle act, \mu_1\rangle \in \mathsf{step}_1$;
- $act \notin A$ and $\mu = \alpha(\eta_{s\lceil_{V_1}} \parallel \mu_2, \Phi)$ and $\langle act, \mu_2\rangle \in \mathsf{step}_2$;
- $act \in A$ and $\mu = \alpha(\mu_1 \parallel \mu_2, \Phi)$ and $\langle act, \mu_i\rangle \in \mathsf{step}_i$ for $i \in \{1,2\}$.

On the other hand, for any $\langle act, \mu\rangle \in \overline{\mathsf{step}}_{\alpha(\parallel)}$, by the hypothesis, Definition 8 and Definition 11 one of the following cases must hold:

- $act \notin A$ and $\mu = \alpha(\mu_1, \Phi_1) \parallel \eta_{\alpha(s,\Phi)\lceil_{\mathsf{bool}(V_2)}}$ and $\langle act, \mu_1\rangle \in \mathsf{step}_1$;
- $act \notin A$ and $\mu = \eta_{\alpha(s,\Phi)\lceil_{\mathsf{bool}(V_1)}} \parallel \alpha(\mu_2, \Phi_2)$ and $\langle act, \mu_2\rangle \in \mathsf{step}_2$;
- $act \in A$ and $\mu = \alpha(\mu_1, \Phi_1) \parallel \alpha(\mu_2, \Phi_2)$ and $\langle act, \mu_i\rangle \in \mathsf{step}_i$ for $i \in \{1,2\}$.

Now, since $\eta_{\alpha(s,\Phi)\lceil_{\mathsf{bool}(V_i)}} = \alpha(\eta_{s\lceil_{V_i}}, \Phi)$ for $i \in \{1,2\}$, it follows from Lemma A.2 that the cases are equivalent, and hence $\overline{\mathsf{step}}_{\alpha\|\alpha} = \overline{\mathsf{step}}_{\alpha(\|)}$ as required. □

We are now in a position to present the proofs of Proposition 3.3 and Proposition 3.4.

Proof of Proposition 3.3. Let $\mathcal{C}_i = \langle V, V_i^{\mathsf{ctrl}}, V_i^{\mathsf{ext}}, Act_i, s^{\mathsf{init}}, Steps_i\rangle$ for $i \in \{1,2\}$ and $\mathcal{C}_1 \,|[A]|\, \mathcal{C}_2 = \langle V, V^{\mathsf{ctrl}}, V^{\mathsf{ext}}, Act, s^{\mathsf{init}}, Steps\rangle$. Consider any $s, s' \in \mathsf{val}(V)$ such that $s\lceil_{V^{\mathsf{ctrl}}} = s'\lceil_{V^{\mathsf{ctrl}}}$ and $\alpha(s, \Phi^{\mathsf{ext}}) = \alpha(s', \Phi^{\mathsf{ext}})$. Now by Definition 4 we have $\langle s, Steps\,(s)\rangle = \langle s, Steps_1(s)\rangle \,|[A]|\, \langle s, Steps_2(s)\rangle$ and using Lemma A.3 it follows that:

$$\alpha(\langle s, Steps\,(s)\rangle, \Phi) = \alpha(\langle s, Steps_1(s)\rangle, \Phi) \,|[A]|\, \alpha(\langle s, Steps_2(s)\rangle, \Phi_2)\,.$$

Similarly, we have:

$$\alpha(\langle s', Steps\,(s')\rangle, \Phi) = \alpha(\langle s', Steps_1(s')\rangle, \Phi) \,|[A]|\, \alpha(\langle s', Steps_2(s')\rangle, \Phi)\,.$$

By the hypothesis $\mathcal{C}_1$ and $\mathcal{C}_2$ are abstraction preserving, therefore by construction of s and s' we have $\alpha(\langle s, Steps_i(s)\rangle, \Phi) = \alpha(\langle s', Steps_i(s')\rangle, \Phi)$ for $i \in \{1,2\}$. Combining these results gives $\alpha(\langle s, Steps\,(s)\rangle, \Phi) = \alpha(\langle s', Steps\,(s')\rangle, \Phi)$, and hence $\mathcal{C}_1 \,|[A]|\, \mathcal{C}_2$ is abstraction preserving as required. □

Proof of Proposition 3.4. Let $\mathcal{C}_i = \langle V, V_i^{\mathsf{ctrl}}, V_i^{\mathsf{ext}}, Act_i, s^{\mathsf{init}}, Steps_i\rangle$ for $i \in \{1,2\}$. By construction the controlled variables, external variables, actions and initial valuations of $\alpha(\mathcal{C}_1, \Phi) \,|[A]|\, \alpha(\mathcal{C}_2, \Phi)$ and $\alpha(\mathcal{C}_1 \,|[A]|\, \mathcal{C}_2, \Phi)$ are equal. To complete the proof it therefore remains to show that the transition functions of the two ACMDPs are the same. To ease notation let $\overline{Steps}_{\alpha\|\alpha}$ denote the transition function of $\alpha(\mathcal{C}_1, \Phi) \,|[A]|\, \alpha(\mathcal{C}_2, \Phi)$, $\overline{Steps}_{\alpha(\|)}$ the transition function of $\alpha(\mathcal{C}_1 \,|[A]|\, \mathcal{C}_2, \Phi)$ and $\alpha(\mathcal{C}_i, \Phi) = \langle \overline{V}, \overline{V_i^{\mathsf{ctrl}}}, \overline{V_i^{\mathsf{ext}}}, \overline{Act_i}, \overline{s^{\mathsf{init}}}, \overline{Steps}_i\rangle$ for $i \in \{1,2\}$. We split the proof into two parts by showing that for any $\overline{s} \in \mathsf{val}(\overline{V})$:

1. if $\overline{\mathsf{step}} \in \overline{Steps}_{\alpha\|\alpha}(\overline{s})$, then $\overline{\mathsf{step}} \in \overline{Steps}_{\alpha(\|)}(\overline{s})$;
2. if $\overline{\mathsf{step}} \in \overline{Steps}_{\alpha(\|)}(\overline{s})$, then $\overline{\mathsf{step}} \in \overline{Steps}_{\alpha\|\alpha}(\overline{s})$.

Therefore consider any $\overline{s} \in \mathsf{val}(\overline{V})$.

1. If $\overline{\mathsf{step}} \in \overline{Steps}_{\alpha\|\alpha}(\overline{s})$, then from Definition 8 there exists $\overline{\mathsf{step}}_1 \in \overline{Steps}_1(\overline{s})$ and $\overline{\mathsf{step}}_2 \in \overline{Steps}_2(\overline{s})$ such that $\langle \overline{s}, \overline{\mathsf{step}}\rangle = \langle \overline{s}, \overline{\mathsf{step}}_1\rangle \,|[A]|\, \langle \overline{s}, \overline{\mathsf{step}}_2\rangle$. Now, since $\overline{\mathsf{step}}_1 \in \overline{Steps}_1(\overline{s})$ and $\overline{\mathsf{step}}_2 \in \overline{Steps}_2(\overline{s})$, it follows from Definition 11 there exist valuations $s_1, s_2 \in \mathsf{val}(V)$ such that $\alpha(s_1, \Phi) = \alpha(s_2, \Phi) = \overline{s}$ and $\alpha(\langle s_1, Steps_1(s_1)\rangle, \Phi) = \langle s_1, \overline{\mathsf{step}}_1\rangle$ and $\alpha(\langle s_2, Steps_2(s_2), \Phi) = \langle s_2, \overline{\mathsf{step}}_2\rangle$. Letting $s = s_1\lceil_{(V\setminus V_2^{\mathsf{ctrl}})} \| s_2\lceil_{V_2^{\mathsf{ctrl}}}$, we have $s\lceil_{V_1^{\mathsf{ctrl}}} = s_1\lceil_{V_1^{\mathsf{ctrl}}}$, $s\lceil_{V_2^{\mathsf{ctrl}}} = s_2\lceil_{V_2^{\mathsf{ctrl}}}$ and $\alpha(s, \Phi) = \overline{s}$, since $\mathcal{C}_1$ and $\mathcal{C}_2$ are abstraction preserving, we have:

$$\alpha(\langle s, Steps_i(s)\rangle, \Phi) = \alpha(\langle s_i, Steps_i(s_i)\rangle, \Phi) \;\text{ for } i \in \{1,2\},$$

   and therefore

$$\begin{aligned} \langle \overline{s}, \overline{\mathsf{step}}\rangle &= \alpha(\langle s, Steps_1(s)\rangle, \Phi) \,|[A]|\, \alpha(\langle s, Steps_2(s)\rangle, \Phi) \\ &= \alpha(\langle s, Steps_1(s)\rangle \,|[A]|\, \langle s, Steps_2(s)\rangle, \Phi) \qquad \text{by Lemma A.3.} \end{aligned}$$

From Definition 11 and Definition 4 it follows that $\overline{\mathsf{step}}$ is the element of $\overline{Steps}_{\alpha(\|)}(\overline{s})$, induced by s, as required.

2. If $\overline{\mathsf{step}} \in \overline{Steps}_{\alpha(\|)}(\overline{s})$, then by Definition 11 there exists $s \in \mathsf{val}(V)$ such that $\alpha(s, \Phi) = \overline{s}$ and

$$\begin{aligned} \langle s, \overline{\mathsf{step}} \rangle &= \alpha(\langle s, Steps_1(s) \rangle \,|[A]|\, \langle s, Steps_2(s) \rangle, \Phi) \\ &= \alpha(\langle s, Steps_1(s) \rangle, \Phi) \,|[A]|\, \alpha(\langle s, Steps_2(s) \rangle, \Phi) \qquad \text{by Lemma A.3.} \end{aligned}$$

   It then follows from Definition 8 that $\overline{\mathsf{step}} \in \overline{Steps}_{\alpha\|\alpha}(\overline{s})$ as required.

Thus the transition functions of the two ACMDPs are the same which completes the proof of Proposition 3.4. □

B Proofs of Section 4.2

As in Section 4.2 we assume a fixed PRISM model $\mathbf{P} = \langle \mathsf{var}(\mathbf{P}), \mathsf{sys}, \{M_1, \ldots, M_m\} \rangle$ and a set of predicates Φ which is partitioned into subsets $\Phi^{M_1}, \ldots, \Phi^{M_m}$ over the local variables of the modules $M_1, \ldots, M_m$. Before presenting the proofs we first recall Proposition 4.2 and Theorem 4.3.

Proposition 4.2 *If M is a module of $\mathbf{P}$, then $[\![\beta(M, \Phi)]\!] = \alpha([\![M]\!], \Phi)$.*

Theorem 4.3 *If $[\![M]\!]$ is abstract preserving with respect to Φ for each module M of $\mathbf{P}$, then $[\![\beta(\mathbf{P}, \Phi)]\!] = \alpha([\![\mathbf{P}]\!], \Phi)$.*

Proof of Proposition 4.2. By construction the controlled variables, external variables, actions and initial valuations of $[\![\beta(M, \Phi)]\!]$ and $\alpha([\![M]\!], \Phi)$ are equal. To complete the proof it therefore remains to show that the transition functions of the two ACMDPs are the same. Let $\overline{Steps}_{[\![\beta(M,\Phi)]\!]}$ denote the transition function of $[\![\beta(M, \Phi)]\!]$ and $\overline{Steps}_{\alpha([\![M]\!],\Phi)}$ the transition function of $\alpha([\![M]\!], \Phi)$, to complete the proof we will show that $\overline{Steps}_{[\![\beta(M,\Phi)]\!]}(\overline{s}) = \overline{Steps}_{\alpha([\![M]\!],\Phi)}(\overline{s})$ for all $\overline{s} \in \mathsf{val}(\mathsf{bool}(\Phi))$. Therefore consider any $\overline{s} \in \mathsf{val}(\mathsf{bool}(\Phi))$.

- If $\overline{\mathsf{step}}_{[\![\beta(M,\Phi)]\!]} \in \overline{Steps}_{[\![\beta(M,\Phi)]\!]}(\overline{s})$, then by Definition 14 there exists a command $\overline{cmd}$ of $\beta(M, \Phi^M)$ such that:

$$\overline{\mathsf{step}}_{[\![\beta(M,\Phi)]\!]} = \left\{ \langle \overline{\mathsf{act}}(\overline{chc}), \overline{\mathsf{dist}}(\overline{chc}, \overline{s}) \rangle \mid \overline{chc} \in \overline{\mathsf{choices}}(\overline{cmd}) \right\}$$

  and $\overline{\mathsf{guard}}(\overline{cmd})(\overline{s})$ holds. By Definition 15, there exists a valuation $s \in \mathsf{val}(\mathsf{var}(\mathbf{P}))$ such that:

$$\overline{\mathsf{step}}_{[\![\beta(M,\Phi)]\!]} = \left\{ \langle \overline{\mathsf{act}}(\overline{cmd}), \overline{\mathsf{dist}}(\overline{cmd}, \overline{s}) \rangle \mid cmd \in \mathsf{com}(M) \wedge \mathsf{guard}(cmd)(s) \right\}$$

  and $\alpha(s, \Phi) = \overline{s}$.

- On the other hand, if $\overline{\mathsf{step}}_{\alpha([\![M]\!],\Phi)} \in \overline{\mathit{Steps}}_{\alpha([\![M]\!],\Phi)}(\overline{s})$, then from Definition 11 it follows that there exists a valuation $s \in \mathsf{val}(\mathsf{var}(\mathbf{P}))$ such that $\langle \overline{s}, \overline{\mathsf{step}}_{\alpha([\![M]\!],\Phi)} \rangle = \alpha(\langle s, \mathsf{step}_{\alpha([\![M]\!],\Phi)} \rangle, \Phi)$ where $\langle s, \mathsf{step}_{\alpha([\![M]\!],\Phi)} \rangle$ is a transition of $[\![M]\!]$. By Definition 6 it follows that $\overline{\mathsf{step}}_{\alpha([\![M]\!],\Phi)}$ equals

$$\begin{aligned}&\alpha\left(\{\langle \mathsf{act}(cmd), \mathsf{dist}(cmd, s)\rangle \mid cmd \in \mathsf{com}(M) \wedge \mathsf{guard}(cmd)(s)\}, \Phi^M\right)\\ &= \left\{\langle \mathsf{act}(cmd), \alpha(\mathsf{dist}(cmd, s), \Phi^M)\rangle \mid cmd \in \mathsf{com}(M) \wedge \mathsf{guard}(cmd)(s)\right\}\end{aligned}$$

by Definition 11.

Combining these results and since $\overline{\mathsf{act}}(\overline{cmd}) = \mathsf{act}(cmd)$, it follows that it is sufficient to show that

$$\alpha(\mathsf{dist}(cmd, s), \Phi^M) = \overline{\mathsf{dist}}(\overline{cmd}, \overline{s}) \ \text{ for all } cmd \in \mathsf{com}(M)\,.$$

Therefore consider any $cmd \in \mathsf{com}(M)$ where $\mathsf{updates}(cmd) = \{\langle \lambda_1, u_1\rangle, \ldots, \langle \lambda_n, u_n\rangle\}$ and $\overline{s}' \in \mathsf{val}(\mathsf{bool}(\Phi^M))$, from Definition 9 we have:

$$\begin{aligned}
\alpha(\mathsf{dist}(cmd, s), \Phi^M)(\overline{s}') &= \sum_{\alpha(s', \Phi^M) = \overline{s}'} \mathsf{dist}(cmd, s)(s') \\
&= \sum_{\alpha(s', \Phi^M) = \overline{s}'} \left(\sum_{1 \leq i \leq n \wedge u_i(s) = s'} \lambda_i \right) && \text{by definition of } \mathsf{dist}(\cdot,\cdot) \\
&= \sum_{\substack{1 \leq i \leq n \wedge \\ \alpha(u_i(s), \Phi^M) = \overline{s}'}} \lambda_i && \text{rearranging} \\
&= \sum_{\substack{1 \leq i \leq n \wedge \\ \overline{u_i}(\overline{s}) = \overline{s}'}} \lambda_i && \text{by Definition 15} \\
&= \sum_{\substack{\overline{u} \in \{\overline{u}_i \,|\, 1 \leq i \leq n\} \\ \wedge \overline{u}(\overline{s}) = \overline{s}'}} \left(\sum_{1 \leq i \leq n \wedge \overline{u_i} = \overline{u}} \lambda_i \right) && \text{rearranging} \\
&= \sum_{\substack{\langle \lambda, \overline{u}\rangle \in \overline{\mathsf{updates}}(\overline{cmd}) \\ \wedge \overline{u}(\overline{s}) = \overline{s}'}} \lambda && \text{by Definition 15} \\
&= \overline{\mathsf{dist}}(\overline{cmd}, \overline{s})(\overline{s}') && \text{by definition of } \overline{\mathsf{dist}}(\cdot,\cdot).
\end{aligned}$$

Therefore, since $\overline{s}' \in \mathsf{val}(\mathsf{bool}(\Phi^M))$ and $cmd \in \mathsf{com}(M)$ were arbitrary, it follows that $\alpha(\mathsf{dist}(cmd, s), \Phi^M) = \overline{\mathsf{dist}}(\overline{cmd}, \overline{s})$ for all $cmd \in \mathsf{com}(M)$ as required. □

Proof of Theorem 4.3. The proof is by induction on the structure of sys. In the base case, $\mathsf{sys} = M$ for some module M and hence $[\![\beta(\mathbf{P}, \Phi)]\!] = [\![\beta(M, \Phi)]\!] = \alpha([\![M]\!], \Phi)$ by Proposition 4.2 as required.

For the inductive step we have that $\mathsf{sys} = \mathsf{sys}_1 \,|[A]|\, \mathsf{sys}_2$ for some process-algebraic expressions sys_1 and sys_2 over the modules of $\mathbf{P}$. First note that, since $[\![M]\!]$ is abstract preserving with respect to Φ for each module M of $\mathbf{P}$, applying Proposition 3.3 it follows that $[\![\mathsf{sys}_1]\!]$ and $[\![\mathsf{sys}_2]\!]$ are abstract preserving, and hence from Proposition 3.4 we have:

$$\alpha([\![\mathsf{sys}_1]\!], \Phi) \,|[A]|\, \alpha([\![\mathsf{sys}_2]\!], \Phi) \;=\; \alpha([\![\mathsf{sys}_1]\!] \,|[A]|\, [\![\mathsf{sys}_2]\!], \Phi)\,. \tag{1}$$

Now by definition of $[\![\cdot]\!]$:

$$\begin{aligned}
[\![\beta(\mathbf{P}, \Phi)]\!] &= [\![\beta(\mathsf{sys}_1), \Phi]\!] \,|[A]|\, [\![\beta(\mathsf{sys}_2), \Phi]\!] & \\
&= \alpha([\![\mathsf{sys}_1]\!], \Phi) \,|[A]|\, \alpha([\![\mathsf{sys}_2]\!], \Phi) & \text{by induction} \\
&= \alpha([\![\mathsf{sys}_1]\!] \,|[A]|\, [\![\mathsf{sys}_2]\!], \Phi) & \text{by (1)} \\
&= \alpha([\![\mathsf{sys}_1 \,|[A]|\, \mathsf{sys}_2]\!], \Phi) & \text{by definition of } [\![\cdot]\!] \\
&= \alpha([\![\mathbf{P}]\!], \Phi) & \text{by definition of } \mathbf{P},
\end{aligned}$$

and hence the theorem holds by induction on the structure of sys. □