Variable Probabilistic Abstraction Refinement

Luis María Ferrer Fioriti[1], Ernst Moritz Hahn[1], Holger Hermanns[1], and Björn Wachter[2]

[1] Saarland University, Germany
[2] University of Oxford, UK

Abstract. Predicate abstraction has proven powerful in the analysis of very large probabilistic systems, but has thus far been limited to the analysis of systems with a fixed number of distinct transition probabilities. This excludes a large variety of potential analysis cases, ranging from sensor networks to biochemical systems. In these systems, transition probabilities are often given as a function of state variables—leading to an arbitrary number of different probabilities.
This paper overcomes this shortcoming. It extends existing abstraction techniques to handle such *variable probabilities*. We first identify the most precise abstraction in this setting, the best transformer. For practicality purposes, we then devise another type of abstraction, mapping on extensions of constraint or interval Markov chains, which is less precise but better applicable in practice. Refinement techniques are employed in case a given abstraction yields too imprecise results. We demonstrate the practical applicability of our method on two case studies.

1 Introduction

Many systems, including network protocols, manufacturing systems and biological systems are characterised by *random* phenomena which can be modeled using probabilities. Since these system are distributed, they are inherently *concurrent*. Markov decision processes (MDPs) are often used as a semantic foundation in this context, because they account for both nondeterminism (used to model concurrency) and probabilism. Typically, one is interested in computing (maximal or minimal) reachability probabilities, e.g., the chance of having delivered three messages after ten transmission attempts, under best-case and worst-case assumptions concerning the environment. For finite MDPs, these probabilities can be computed by linear optimisation or using numerical techniques such as value iteration [6]. The latter is done e.g., in the popular probabilistic model checker Prism [13]. However, this approach suffers from the state explosion problem, even more than in non-probabilistic model checking, due to expensive numerical computations.

Abstraction-refinement methods have gained popularity as approaches to alleviate this problem. Early approaches (e.g. [1,3]) were restricted to finite models, since they unfold the state space of the model under analysis. More recently, symbolic abstractions that operate at the source-code level were introduced to support infinite or very large models. Predicate abstraction and counterexample-guided abstraction refinement (CEGAR) have been proposed [8] and implemented in the verification tool Pass [7] for concurrent probabilistic programs. Building on seminal work on game-based abstraction [14], abstraction refinement has also been applied to sequential probabilistic programs [11], probabilistic timed automata [15] and finite-state concurrent probabilistic programs [12].

There is one disturbing limitation to the otherwise promising techniques based on symbolic abstraction, which is common to all works in this field [8,14,11,15,12]. Namely, so far, the analysis is limited to models with a fixed set of transition probabilities, and effectively cannot handle probabilities expressed in terms of arithmetic expressions that involve program variables. For instance, it is not possible to have transition probabilities $\frac{1}{n}$ and $1 - \frac{1}{n}$ where n is an integer variable. Such *variable probabilities* induce a potentially unbounded number of transition probabilities corresponding to the underlying variable domain. Previous symbolic-abstraction techniques [14,19] produce intractably large, or even infinite abstract models in presence of variable probabilities, while existing interval abstractions of transition probabilities [9] only apply to finite explicit-state models [4,10].

However, variable probabilities naturally arise in most biological systems, in wireless sensor networks, manufacturing systems and many other application contexts. For instance, protocol standards such as IETF RFC 3927 (Zeroconf), or IEEE 802.11e (QoS for WLAN) make extensive use of random selections which are based on the value of certain state variables. With the further advancement of self-stabilising, self-healing, self-supportive or energy-harvesting systems, the use of situation-specific random sampling is likely to be the rule, not the exception. Guarantees for such systems are probabilistic in nature, but are only possible if such randomisation mechanisms are supported by analysis tools. This is what this paper aims to achieve for probabilistic abstraction refinement. We develop theory and tools for the automatic analysis of concurrent probabilistic systems with variable probabilistic selections. We introduce novel abstractions and refinement procedures for probabilistic programs based on probabilistic games.

To handle variable probabilities, our new abstractions explicitly summarise transition probabilities, which is more challenging than both (interval) abstraction of explicit-state models and non-variable symbolic abstraction, because computing abstract transition probabilites requires reasoning over the arithmetic theory of the program. We solve this problem by using optimisation techniques and explore the trade-off between precision versus performance by applying optimisation at different stages. Our symbolic abstraction retains the symbolic representation of transition probabilities, as in the program. While this is very precise, the analysis involves a costly fixed-point iteration with nested linear optimisation problems. We therefore develop an alternative interval abstraction technique, computing a probabilistic game with transition probabilities given as intervals, which may be less precise but requires to only solve an optimisation problem once. A prototype of the interval abstraction with abstraction refinement has been implemented in Pass and has been successfully applied to a number of non-trivial case studies.

Outline. In Section 2, we discuss probabilistic programs and their semantics. In Section 3, we introduce an abstract game model to safely overapproximate properties of the semantics. Computing abstractions in practice is treated in Section 4. Automatic refinement is discussed in Section 5. Experiments follow in Section 6 and Section 7 concludes the paper. An extended version containing the proofs and covering also abstractions for models in which the size of the probability distributions can be variable can be found at [5]

2 Preliminaries

Let S be a set. A *probability distribution* over S is a function $\mu\colon S \to [0,1]$ with $\sum_{s\in S} \mu(s) = 1$. The *support* of μ is given by $Supp(\mu) = \{s \in S \mid \mu(s) > 0\}$. We denote the set of all probability distributions over a given set S by $Distr(S)$. The domain of a partial function $f\colon A \rightharpoonup B$ is $Dom(f) \subseteq A$.

Definition 1. *A* probabilistic automaton *is a tuple* (S, I, Act, δ) *where*

- S *is a set of* states *of which* $I \subseteq S$ *with* $I \neq \emptyset$ *is the set of* initial states*,*
- Act *is a set of* actions*, and*
- $\delta\colon (S \times Act) \rightharpoonup Distr(S)$ *is the* transition function*.*

We denote by $en(s) \subseteq Act$ the actions that are enabled in s, i.e. $\alpha \in en(s)$ iff $\delta(s,\alpha)$ is defined. We require that $en(s) \neq \emptyset$ for all $s \in S$. Any automaton that does not satisfy this condition can be modified by introducing self-loops in the deadlock states. Intuitively, in each state s, we choose an action $\alpha \in en(s)$. In turn, the probability distribution $\mu = \delta(s,\alpha)$ provides a probabilistic choice over the states reached in the next step.

Definition 2. *A* probabilistic program *is a tuple* $\mathcal{P} = (\mathtt{X}, \mathtt{Dom}, \mathtt{I}, \mathtt{C})$ *where*

- $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$ *is a set of* program variables*; for each variable* $\mathtt{x}_i \in \mathtt{X}$*, we denote by* $\mathtt{Dom}(\mathtt{x}_i)$ *the (possibly infinite)* variable domain *so that the state space of the program is the set* $\mathtt{Dom}(\mathtt{X}) \stackrel{\text{def}}{=} \mathtt{Dom}(\mathtt{x}_1) \times \cdots \times \mathtt{Dom}(\mathtt{x}_n)$*,*
- $\mathtt{I} \subseteq \mathtt{Dom}(\mathtt{X})$ *with* $\mathtt{I} \neq \emptyset$ *is an expression characterising the set of* initial states*,*
- $\mathtt{C}$ *is a set of* guarded commands *of the form* $\mathtt{c} = (\mathtt{g} \to \mathtt{p}_1 : \mathtt{u}_1 + \ldots + \mathtt{p}_m : \mathtt{u}_m)$ *with a guard expression* $\mathtt{g} \subseteq \mathtt{Dom}(\mathtt{X})$ *and probabilistic choices* $1 \leq i \leq m$*; each choice* i *comes with a* weight function $\mathtt{p}_i$ *and an* update function $\mathtt{u}_i$*:*
  - *update function* $\mathtt{u}_i\colon \mathtt{Dom}(\mathtt{X}) \to \mathtt{Dom}(\mathtt{X})$ *assigns a successor to each state,*
  - *weight function* $\mathtt{p}_i\colon \mathtt{Dom}(\mathtt{X}) \to [0,1]$ *assigns a weight to each state* $\vec{\mathtt{x}}$*, such that the weights sum to one:* $\sum_{1\leq i\leq m} \mathtt{p}_i(\vec{\mathtt{x}}) = 1$*.*

  We require that $\bigcup_{\mathtt{c}\in\mathtt{C}} \mathtt{g}_\mathtt{c} = \mathtt{Dom}(\mathtt{X})$*.*

Definition 3. *The semantics* $sem(\mathcal{P})$ *of a probabilistic program* $\mathcal{P} = (\mathtt{X}, \mathtt{Dom}, \mathtt{I}, \mathtt{C})$ *is the probabilistic automaton* $(S, \mathtt{I}, \mathtt{C}, \delta)$ *with* $S \stackrel{\text{def}}{=} \mathtt{Dom}(\mathtt{X})$*. The transition function is defined on pairs of states* s *and commands* $\mathtt{c}$ *which can be executed in* s*, i.e.,* $Dom(\delta) \stackrel{\text{def}}{=} \{(s,\mathtt{c}) \in S \times \mathtt{C} \mid \mathtt{c} = (\mathtt{g} \to \mathtt{p}_1 : \mathtt{u}_1 + \ldots + \mathtt{p}_m : \mathtt{u}_m) \wedge s \in \mathtt{g}\}$*, and the corresponding distribution is defined by the probabilistic choices, i.e., for each* $s' \in S$ *it is* $\delta(s,\mathtt{c})(s') \stackrel{\text{def}}{=} \sum_{\substack{1\leq i\leq m,\\ \mathtt{u}_i(s)=s'}} \mathtt{p}_i(s)$ *for* $\mathtt{c} = (\mathtt{g} \to \mathtt{p}_1 : \mathtt{u}_1 + \ldots + \mathtt{p}_m : \mathtt{u}_m) \in \mathtt{C}$*.*

The guarded-command language in Definition 2 forms the core language of the probabilistic model checker PRISM [13], for which a vast collection of case studies exists, many of which feature variable probabilities. However, previous predicate abstraction techniques for probabilistic programs did not fully support variable probabilities. The probability associated with a probabilistic choice had to be a *constant*. To a limited degree, variable probabilities could be encoded by creating a copy of the same command for each induced distribution. However, variable probabilities may induce infinitely many distributions, as illustrated by the following Example 1. Even if the number of distributions is finite, the encoding creates extra work for the abstraction procedure with each copy of a command.

Example 1. Consider the program $(\mathtt{X}, \mathtt{Dom}, \mathtt{I}, \mathtt{C})$ in Fig. 1. The program contains two variables, $\mathtt{s}$ and $\mathtt{x}$ where $\mathtt{Dom}(\mathtt{s}) = \{0,1,2,3\}$ and $\mathtt{Dom}(\mathtt{x}) = \mathbb{N}$. We have $\mathtt{I} = \{0\} \times \mathbb{N}$. For command $\mathtt{b} = (\mathtt{g} \to \mathtt{p}_1 : \mathtt{u}_1 + \mathtt{p}_2 : \mathtt{u}_2)$ we have $\mathtt{g} = \{1\} \times \mathbb{N}$, $\mathtt{p}_1(s,x) = \frac{9x-8}{16x}$, $\mathtt{p}_2(s,x) = \frac{7x+8}{16x}$, $\mathtt{u}_1(s,x) = (2,x)$ and $\mathtt{u}_2(s,x) = (3,x)$. The other commands are formalised likewise.

This program could not have been handled by the previous approach, as command $\mathtt{b}$ would have to be encoded by a countably infinite number of guarded commands. Even if the domain of variable x was finite, the number of transition probabilities in the abstraction would have at least the size of the domain $\mathtt{Dom}(x)$.

Probabilistic Reachability. Given the automata semantics $\mathcal{M} = (S, I, Act, \delta)$ of a probabilistic program, we are interested in the probability to reach a set of goal states $F \subseteq S$. This probability depends on the resolution of nondeterminism in the automaton, i.e. choosing an action at a given state. A particular resolution of nondeterminism is a function from paths to a distribution over actions, and induces a probability measure over a set of paths. We are interested in the minimal and maximal probabilities to reach a set of goal states F, particularly starting in an initial state.

```
s : [0..3];  x : [1..inf);
[a] s=0 -> (s'=1) & (x'=1);
[b] s=1 -> ((9x-8)/(16x)):(s'=2)
         + ((7x+8)/(16x)):(s'=3);
[c] s=1 -> 1 : (x'=x+1);
init s=0 endinit
```

Fig. 1. Example for a probabilistic program.

Reachability probabilities can be expressed and computed in terms of functions from states to probabilities $\nu \colon S \to [0,1]$. We call these functions *valuations* and denote by $Asg(S)$ the set of all valuations for S.

The set $Asg(S)$ forms a complete lattice with the pointwise order $\leq$, i.e. for each pair $\nu_1, \nu_2 \in Asg(S)$ we let $\nu_1 \leq \nu_2 \overset{\text{def}}{\iff} \forall s \in S.\ \nu_1(s) \leq \nu_2(s)$. We call a monotone function $f \colon Asg(S) \to Asg(S)$ a *valuation transformer*. As valuations form a complete lattice, each transformer has a least (and a greatest) fixed point $lfp_{\leq}\, f$ ($gfp_{\leq}\, f$).

Definition 4. *Given a probabilistic automaton $\mathcal{M} = (S, I, Act, \delta)$ and a set of* goal states *$F \subseteq S$, the* $+$ valuation transformer *is the function $pre^{+}_{\mathcal{M},F} \colon Asg(S) \to Asg(S)$. For $\nu \in Asg(S)$ and $s \in S$ it is $pre^{+}_{\mathcal{M},F}(\nu)(s) \overset{\text{def}}{=} 1$ for $s \in F$ and otherwise*

$$pre^{+}_{\mathcal{M},F}(\nu)(s) \overset{\text{def}}{=} \sup_{\alpha \in en(s)} \sum_{s' \in S} \delta(s,\alpha)(s') \cdot \nu(s').$$

The $-$ valuation transformer is defined accordingly using the infimum. By $p^{\bullet}_{\mathcal{M},F} \overset{\text{def}}{=} lfp_{\leq}\ pre^{\bullet}_{\mathcal{M},F}$ for $\bullet \in \{+,-\}$ we describe the least fixed points of these operators.

The maximal (minimal) reachability probability equals $p^{+}_{\mathcal{M},F}$ ($p^{-}_{\mathcal{M},F}$). This allows to use an iterative algorithm to compute concrete probabilities in finite models [6]: We let $\nu^0(s) \overset{\text{def}}{=} 0$ and $\nu^{i+1}(s) \overset{\text{def}}{=} pre^{+}_{\mathcal{M},F}(\nu^i)(s)$ for $i \geq 0$. The sequence ν^i converges to the fixed point. We can thus use ν^n as an approximation for the fixed point, where n is chosen according to the precision to be obtained.

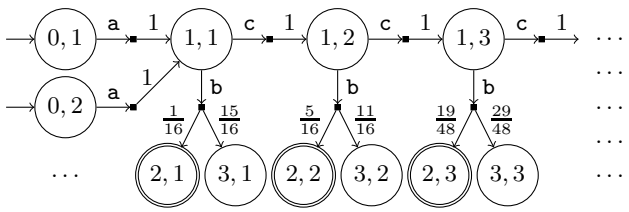

Fig. 2. Probabilistic automata semantics of the probabilistic program in Fig. 1.

Example 2. Fig. 2 shows a fraction of the infinite semantics of the program of Fig. 1, where we mark states of the goal set $F = \{(s, x) \in \mathtt{Dom(X)} \mid s = 2\}$. The supremum probability to reach F is $\frac{9}{16}$: After an initial transition to state $(1, 1)$ by command $\mathtt{a}$, in each state $(1, x)$ one can decide to either move to state $(1, x)$ with certainty or to move to F with probability $\frac{9x-8}{16x}$. It is $\lim_{x\to\infty} \frac{9x-8}{16x} = \frac{9}{16}$, which means that by first executing a sufficiently long sequence of command $\mathtt{c}$, one can reach F with a probability arbitrary close to $\frac{9}{16}$.

Abstract Interpretation. Variable probabilities induce models with unbounded numbers of transition probabilities. This asks for abstraction techniques that extend the existing frameworks [14,19]. In this section, we revisit the formal framework [19] to reason about abstractions for probabilistic programs building on the theory of abstract interpretation. We begin by introducing an abstract state space which partitions[3] the concrete state space.

Definition 5. *Given a probabilistic automaton* (S, I, Act, δ)*, an* abstract state space *is a finite partition* $\mathcal{A} = \{\mathtt{z}_1, \ldots, \mathtt{z}_n\}$ *of* S*, i.e., for all* $1 \leq i \leq n$ *it is* $\mathtt{z}_i \subseteq S$*, for all* $1 \leq j \leq n$ *with* $i \neq j$ *we have* $\mathtt{z}_i \cap \mathtt{z}_j = \emptyset$*, and* $S = \bigcup_{i\in\{1,\ldots,n\}} \mathtt{z}_i$*. For* $\mathtt{z} \in \mathcal{A}$ *and* $s_1, s_2 \in \mathtt{z}$ *we need to have* $en(s_1) = en(s_2)$*. For the analysis with a goal set* F*, we require that for all* $\mathtt{z} \in \mathcal{A}$ *either* $\mathtt{z} \cap F = \emptyset$ *or* $\mathtt{z} \subseteq F$*.*

When we go from the domain $Asg(S)$ to $Asg(\mathcal{A})$, we summarise states *and* probabilities. Assume we already have a concrete valuation and want to compute an abstraction in $Asg(\mathcal{A})$ that represents a lower bound. The valuation we choose maps a given abstract state $\mathtt{z}$ to the lower bound (infimum) of the probabilities over all states $s \in \mathtt{z}$ contained in $\mathtt{z}$. This is captured by the function α^l:

$$\alpha^l \colon\ Asg(S) \to Asg(\mathcal{A}), \quad \nu \mapsto \nu^\sharp \text{ where } \nu^\sharp(\mathtt{z}) \stackrel{\text{def}}{=} \inf_{s\in\mathtt{z}} \nu(s) \text{ for all } \mathtt{z} \in \mathcal{A}.$$

The *upper-bound abstraction* function α^u is defined analogously with the difference that ν is mapped to $\nu^\sharp(\mathtt{z}) \stackrel{\text{def}}{=} \sup_{s\in\mathtt{z}} \nu(s)$ for all $\mathtt{z} \in \mathcal{A}$.

The abstraction functions have a counterpart, the *concretisation* function γ. It maps an abstract valuation back to a concrete one in the obvious way:

$$\gamma \colon\ Asg(\mathcal{A}) \to Asg(S), \quad \nu^\sharp \mapsto \nu \text{ where } \nu(s) \stackrel{\text{def}}{=} \nu^\sharp(\mathtt{z}) \text{ for all } s \in S \text{ with } s \in \mathtt{z}.$$

[3] In the definition, we require that for a given abstract state all contained concrete states are able to perform the same set of commands. This assumption is not strictly needed but used to simplify notations later on, and because it is fulfilled automatically by our implementation of the abstraction methods.

Intuitively, our choice of pairs (α^l, γ) and (α^u, γ) defines a way to map a valuation to "a lower resolution" and back—preserving either lower or upper bounds. The following theorem establishes that this mapping is canonical yielding the most precise mapping that still guarantees correct bounds. Precisely this notion is captured by the well-established concept of a Galois connection:

Proposition 1 (Galois Connections [18]). *Let α^l, α^u, and γ be as defined above. The pair (α^l, γ) is a Galois connection between the domains $(Asg(S), \geq)$ and $(Asg(\mathcal{A}), \geq)$, and (α^u, γ) is a Galois connection between the two domains $(Asg(S), \leq)$ and $(Asg(\mathcal{A}), \leq)$.*

Notably we have two Galois connections depending on whether we are aiming for lower bounds or for upper bounds. The order on the domains is an approximation order in the following sense: Lower bounds are expressed in the domain $(Asg(S), \geq)$. Because a lower bound is more precise than another one if it is greater, the order for lower bounds is the point-wise ordering $\geq$. Conversely, smaller upper bounds are more precise, therefore the order for upper bounds is $\leq$.

To obtain a working analysis on the abstract domain, we additionally require an abstract transformer with type $Asg(\mathcal{A}) \to Asg(\mathcal{A})$, so that the concrete fixed-point computation of transformer $pre^{\bullet}_{\mathcal{M},F} \colon Asg(S) \to Asg(S)$ can be replaced by an abstract fixed-point computation. Due to the complex interplay between nondeterminism and probability, it is not immediately clear what is the best choice. Thankfully abstract interpretation provides canonical and most precise abstract transformers defined in terms of function composition: $\alpha^l \circ pre^{\bullet}_{\mathcal{M},F} \circ \gamma$ for the lower bound and $\alpha^u \circ pre^{\bullet}_{\mathcal{M},F} \circ \gamma$ for the upper bound [19].

While abstract interpretation specifies desirable abstract transformers, probabilistic games provide a representation for them, which stands out from other abstract models in that it admits computing effective lower and upper bounds on reachability probabilities [14]. In the next section, we discuss such games.

3 Probabilistic Games

In this section, we discuss several variants of turn-based probabilistic games, which are needed to obtain abstractions of probabilistic programs with variable probabilities. We begin by introducing probabilistic games which extend probabilistic automata by introducing a second instance of nondeterministic choice. The two kinds of nondeterminism are resolved by two *players*. The basic definition of a probabilistic game is as follows:

Definition 6. *A* probabilistic two-player game *is a tuple $(S, I, Act_1, Act_2, \delta)$ where*

- *S is a set of* states *of which $I \subseteq S$ with $I \neq \emptyset$ is the set of* initial states*,*
- *for $i = 1, 2$, it is Act_i the set of player-i actions, and*
- *$\delta \colon (S \times Act_1 \times Act_2) \rightharpoonup Distr(S)$ provides the* transition function*.*

Similarly to probabilistic automata, we introduce the notion of enabled actions. We have $\alpha_1 \in en_1(s) \subseteq Act_1$ if there exist $\alpha_2 \in Act_2$ such that $\delta(s, \alpha_1, \alpha_2)$ is defined and we have $\alpha_2 \in en_2(s, \alpha_1) \subseteq Act_2$ if $\delta(s, \alpha_1, \alpha_2)$ is defined. We also

require $en_1(s) \neq \emptyset$ for all $s \in S$ and $en_2(s, \alpha_1) \neq \emptyset$ if $\alpha_1 \in en_1(s)$. In state s of a probabilistic game $(S, I, Act_1, Act_2, \delta)$, the first player chooses an action $\alpha_1 \in en_1(s)$. Afterwards, the second player chooses an action $\alpha_2 \in en_2(s, \alpha_1)$. From this, we obtain a distribution $\mu = \delta(s, \alpha_1, \alpha_2)$ which probabilistically determines the next state of the game.

We can define valuation transformers for probabilistic games, similar to those of probabilistic automata. As there are now two types of nondeterminism to be resolved, there are more possible resolutions, and thus valuation transformers.

Definition 7. *Given a probabilistic two-player game $\mathcal{M} = (S, I, Act_1, Act_2, \delta)$ and a set of goal states $F \subseteq S$, the $+,-$* valuation transformer *is the function $pre^{+,-}_{\mathcal{M},F} \colon Asg(S) \to Asg(S)$. For $\nu \in Asg(S)$ and $s \in S$ it is $pre^{+,-}_{\mathcal{M},F}(\nu)(s) \stackrel{\text{def}}{=} 1$ for $s \in F$ and otherwise*

$$pre^{+,-}_{\mathcal{M},F}(\nu)(s) \stackrel{\text{def}}{=} \sup_{\alpha_1 \in en_1(s)} \inf_{\alpha_2 \in en_2(s,\alpha_1)} \sum_{s' \in S} \delta(s, \alpha_1, \alpha_2)(s') \cdot \nu(s').$$

Other transformers are defined accordingly. By $p^{\bullet_1,\bullet_2}_{\mathcal{M},F} = lfp_{\leq}\ pre^{\bullet_1,\bullet_2}$ we denote the least fixed points of these operators.

Similar solution techniques as for probabilistic automata exist [6].

Definition 8. *Let $\mathcal{A}$ be an abstract state space of S and let $\mu \in Distr(S)$ be a distribution. By $lift_{\mathcal{A}}(\mu) \colon \mathcal{A} \to [0,1]$ we denote the* lifting *of μ where, for $\mathsf{z} \in \mathcal{A}$, we have $lift_{\mathcal{A}}(\mu)(\mathsf{z}) \stackrel{\text{def}}{=} \sum_{s \in \mathsf{z}} \mu(s)$. The lifting $lift_{\mathcal{A}}(g) \colon Act \rightharpoonup Distr(\mathcal{A})$ of a partial function $g \colon Act \rightharpoonup Distr(S)$ is defined such that $lift_{\mathcal{A}}(g)(\alpha) \stackrel{\text{def}}{=} lift_{\mathcal{A}}(g(\alpha))$ for all valid $\alpha \in Act$.*

For a probabilistic automaton (S, I, Act, δ) and an abstract state space $\mathcal{A}$, we define $Valid(\mathsf{z}) \stackrel{\text{def}}{=} \{lift_{\mathcal{A}}(\delta(s, \cdot)) \mid s \in \mathsf{z}\}$ for all $\mathsf{z} \in \mathcal{A}$. The game-based abstraction *is then the probabilistic game $(\mathcal{A}, \mathcal{I}, Act_1, Act_2, \delta')$ where*

- $\mathcal{I} \stackrel{\text{def}}{=} \{\mathsf{z} \in \mathcal{A} \mid I \cap \mathsf{z} \neq \emptyset\}$,
- $Act_1 \stackrel{\text{def}}{=} \{f \colon Act \rightharpoonup Distr(\mathcal{A}) \mid \exists \mathsf{z} \in \mathcal{A}.\ f \in Valid(\mathsf{z})\}$,
- $Act_2 \stackrel{\text{def}}{=} \{\alpha \in Act \mid \exists \mathsf{z} \in \mathcal{A}.\ \exists f \in Valid(\mathsf{z}).\ \alpha \in Dom(f)\}$,
- $\delta'(\mathsf{z}, f, \alpha) \stackrel{\text{def}}{=} \begin{cases} f(\alpha) & ; \quad \mathsf{z} \in \mathcal{A}, f \in Valid(\mathsf{z}), \alpha \in Dom(f) \\ \textit{undefined} & ; \quad \textit{otherwise.} \end{cases}$

Definition 7 defines abstract reachability probabilities of probabilistic games. Previously [19], we had shown how this is possible for the restricted setting of static transition probabilities. Maximisation and minimisation at Act_1-choices exactly capture the lower-bound and upper-bound abstraction function, while Act_2-choices capture nondeterministic choices of the original model.

Here we extend this result to variable probabilities, which is possible as the property relies on characteristics of the Galois connection that have been established in Proposition 1. The following theorem states in more detail how the specific lower and upper bounds can be obtained from the game. Furthermore, it shows that the valuation transformer of Definition 7 is most precise, i.e., among the abstract transformers $Asg(\mathcal{A}) \to Asg(\mathcal{A})$ it gives the most precise abstract transformer that approximates the original probabilistic automaton.

Theorem 1. *Let $\mathcal{M} = (S, I, Act, \delta)$ be a probabilistic automaton and let $\mathcal{M}' = (\mathcal{A}, \mathcal{I}, Act_1, Act_2, \delta')$ be a game-based abstraction of $\mathcal{M}$. Let $F' = \{\mathsf{z} \in \mathcal{A} \mid \mathsf{z} \cap F \neq \emptyset\}$. Then, for $\bullet \in \{+, -\}$, for $\mathsf{z} \in \mathcal{A}$ and $s \in \mathsf{z}$, we have:*

$$pre^{-;\bullet}_{\mathcal{M}',F'} = (\alpha^l \circ pre^{\bullet}_{\mathcal{M},F} \circ \gamma) \qquad \text{and} \qquad pre^{+;\bullet}_{\mathcal{M}',F'} = (\alpha^u \circ pre^{\bullet}_{\mathcal{M},F} \circ \gamma)$$

where $pre^{-;\bullet}_{\mathcal{M}',F'}$ and $pre^{+;\bullet}_{\mathcal{M}',F'}$ are the valuation transformers defined by the game.

Corollary 1. *Let $\mathcal{M} = (S, I, Act, \delta)$ be a probabilistic automaton and let $\mathcal{M}' = (\mathcal{A}, \mathcal{I}, Act_1, Act_2, \delta')$ be a game-based abstraction of $\mathcal{M}$. Let $F' = \{\mathsf{z} \in \mathcal{A} \mid \mathsf{z} \cap F \neq \emptyset\}$. Then, for $\bullet \in \{+, -\}$, for $\mathsf{z} \in \mathcal{A}$ and $s \in \mathsf{z}$, we have*

$$p^{-;\bullet}_{\mathcal{M}',F'}(\mathsf{z}) \leq p^{\bullet}_{\mathcal{M},F}(s) \leq p^{+;\bullet}_{\mathcal{M}',F'}(\mathsf{z}).$$

Example 3. In Fig. 3, we abstract the semantics of the program in Fig. 1 to a probabilistic game with four abstract states. Each of them subsumes the concrete states with the same value of $\mathtt{s}$. Small circles correspond to choices of Act_1, whereas small squares correspond to choices of Act_2. Because of the infinite number of distributions, the abstraction is infinitely large.

As seen in Example 3, abstractions of probabilistic programs might still be infinite. The abstraction is guaranteed to be finite, only if probabilities are constants, i.e., do not depend on the program variables. The infiniteness of the game-based abstraction in Definition 8 lies in the number of player choices Act_1 and Act_2.

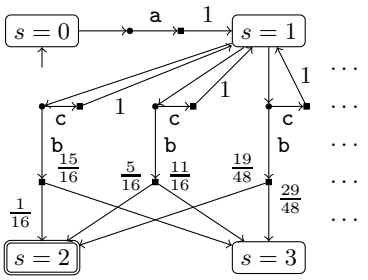

Fig. 3. Game-based abstraction of the semantics in Fig. 2 of the probabilistic program in Fig. 1.

The key idea is to make the sets Act_1 and Act_2 finite, and move the infiniteness to the level of distributions. This is realised by generalising the transition function $\delta\colon (S \times Act_1 \times Act_2) \rightharpoonup Distr(S)$ to return a set of distributions rather than a single distribution. The set of distributions, in turn, can then be represented symbolically, for instance by intervals. The player that controls the abstraction is in charge of picking a distribution, as the different distributions arise from summarising different states. The underlying abstract model is defined as follows:

Definition 9. *A* constraint Markov game *is a tuple $(S, I, Act_1, Act_2, \delta)$ where S, I, Act_1 and Act_2 are as in Definition 6 and the* transition function *is of the form $\delta\colon (S \times Act_1 \times Act_2) \rightharpoonup \mathfrak{C}(\mathbb{N} \times S)$ where $\mathfrak{C}(A) \stackrel{\text{def}}{=} 2^{Distr(A)} \setminus \emptyset$.*

As in probabilistic two-player games, the two players choose their valid actions. However, after the first choice of player 2, there is an additional choice of this player on a distribution function $\mu \in \delta(s, \alpha_1, \alpha_2)$. This distribution is of $Distr(\mathbb{N} \times S)$, where the number $\mathbb{N}$ will be used to later on distinguish between different branches of a guarded command leading to the same abstract state. Constraint Markov games are a special case of probabilistic two-player games—as the choice of both α_2 and the distribution is controlled by player 2.

Definition 10. *Given a constraint Markov game* $\mathcal{M} = (S, I, Act_1, Act_2, \delta)$ *and a set of goal states* $F \subseteq S$*, the* $+,-$ valuation transformer *is defined as the function* $pre^{+,-}_{\mathcal{M},F} \colon Asg(S) \to Asg(S)$*. For* $\nu \in Asg(S)$ *and* $s \in S$ *it is* $pre^{+,-}_{\mathcal{M},F}(\nu)(s) \stackrel{\text{def}}{=} 1$ *for* $s \in F$ *and otherwise*

$$pre^{+,-}_{\mathcal{M},F}(\nu)(s) \stackrel{\text{def}}{=} \sup_{\alpha_1 \in en_1(s)} \inf_{\alpha_2 \in en_2(s,\alpha_1)} \inf_{\mu \in \delta(s,\alpha_1,\alpha_2)} \sum_{i \in \mathbb{N}} \sum_{s' \in S} \mu(i, s') \cdot \nu(s').$$

Other transformers are defined accordingly. By $p^{\bullet_1,\bullet_2}_{\mathcal{M},F} \stackrel{\text{def}}{=} lfp_{\leq}\ pre^{\bullet_1,\bullet_2}$ *we denote the least fixed points of these operators.*

We define a specific form of constraint Markov games.

Definition 11. *An* interval assignment *over a set* A *is a function* $\iota \colon A \to ([0,1] \times [0,1])$*. The set of all interval assignments over* A *is denoted by* $\mathfrak{I}(A)$*. An interval assignment* $\iota \colon A \to ([0,1] \times [0,1])$ *represents the set* $Distr(\iota)$ *of* valid distributions *of* ι*, where* $\mu \in Distr(\iota)$ *iff* $\mu \in Distr(A)$ *and for all* $a \in A$ *if* $\iota(a) = (l, u)$ *then* $\mu(a) \in [l, u]$*. We identify* ι *and* $Distr(\iota)$ *and write* $\mu \in \iota$ *if* $\mu \in Distr(\iota)$

An interval Markov game *is a tuple* $(S, I, Act_1, Act_2, \delta)$ *where* S*,* I*,* Act_1 *and* Act_2 *are as in Definition 6 and the* transition function *is of the form* $\delta \colon (S \times Act_1 \times Act_2) \rightharpoonup \mathfrak{I}(\mathbb{N} \times S)$*.*

Note that interval Markov games are special constraint Markov games and thus the definitions of the valuation transformers $pre^{-,-}_{\mathcal{M},F}, pre^{+,-}_{\mathcal{M},F}, \ldots : Asg(S) \to Asg(S)$ remain as for general constraint Markov games.

We will later on consider both abstractions using interval as well as general constraint Markov games. The interval Markov game abstraction will be coarser, but easier to compute than a constraint Markov game abstraction on the same abstract state space.

4 Computing Abstractions

In this section, we discuss how to obtain abstractions of probabilistic programs.

Let $(\mathtt{X}, \mathtt{Dom}, \mathtt{I}, \mathtt{C})$ be a program. A *predicate* is a subset $\mathsf{pred} \subseteq \mathtt{Dom}(\mathtt{X})$ of the state space, which can be represented as a Boolean expression over program variables. A *predicate set* is a finite set of predicates $\mathsf{Pred} = \{\mathsf{pred}_1, \ldots, \mathsf{pred}_n\}$. Let F be a set of goal states. We require that $\mathtt{I}, F \in \mathsf{Pred}$ and that for each $\mathtt{c} = (\mathsf{g} \to \mathtt{p}_1 : \mathtt{u}_1 + \ldots + \mathtt{p}_m : \mathtt{u}_m) \in \mathtt{C}$ it is $\mathsf{g} \in \mathsf{Pred}$. By $v_1 \cdots v_n$, with $v_i \in \{0, 1\}$, we denote the abstract state $\bigcap_{1 \leq i \leq n} f(v_i, \mathsf{pred}_i)$, where $f(0, A)$ denotes the complement of the set A and $f(1, A)$ denotes A itself.

In the following, we will assume that predicates are described using a linear theory, which ensures that optimisation problems over the predicate theory remain tractable. Assuming that we already have computed a predicate set, the definition below shows how to obtain a constraint Markov game abstraction. As in menu-based abstraction [19] the first player has to choose a command that is enable in one of the concrete states. The same holds for constraint Markov game abstraction. However, while in menu-based abstraction the second player chooses a possible concretisation for this command, player two makes two choices in a constraint Markov game abstraction: first, it picks a valid branching structure

without considering the probabilities (represented by the function f) and, second, it chooses a valid assignment for the probabilities (represented by δ').

Definition 12. *Given a probabilistic program* $(\mathtt{X}, \mathtt{Dom}, \mathtt{I}, \mathtt{C})$ *with a predicate set* $\mathsf{Pred} = \{\mathsf{pred}_1, \ldots, \mathsf{pred}_n\}$*, the* constraint Markov game abstraction *is defined as* $(\mathcal{A}, \mathcal{I}, Act_1, Act_2, \delta')$ *where*

- $\mathcal{A} \stackrel{\text{def}}{=} \{v_1 \cdots v_n \mid (v_1, \ldots, v_n) \in \{0,1\}^n \wedge v_1 \cdots v_n \neq \emptyset\}$,
- $\mathcal{I} \stackrel{\text{def}}{=} \{v_1 \cdots v_n \mid v_1 \cdots v_n \in \mathcal{A} \wedge v_i = 1 \text{ for } \mathsf{pred}_i = \mathtt{I}\}$,
- $Act_1 \stackrel{\text{def}}{=} \mathtt{C}$,
- $Act_2 \stackrel{\text{def}}{=} \{(\mathtt{c}, f) \mid \mathtt{c}{=}(\mathtt{g} \to \mathtt{p}_1 : \mathtt{u}_1 + \ldots + \mathtt{p}_m : \mathtt{u}_m) \in Act_1 \wedge f\colon \{1, \ldots, m\} \to \mathcal{A}\}$,
- $Valid(v_1 \cdots v_n) = \{\mathtt{c} \mid \mathtt{c} \in Act_1 \wedge \exists i \in \{1 \ldots n\}.\ \mathsf{pred}_i = \mathtt{g} \wedge v_i = 1\}$,
- $Valid(\mathtt{z}, \mathtt{c}) \stackrel{\text{def}}{=} \{(\mathtt{c}, f) \mid (\mathtt{c}, f) \in Act_2 \wedge \exists s \in \mathtt{z}.\ \forall i \in \{1 \ldots m\}.\ \mathtt{u}_i(s) \in f(i)\}$,
- *for* $\mathtt{z} \in \mathcal{A}$, $\alpha_1 = \mathtt{c} \in Valid(\mathtt{z})$, $\alpha_2 = (\mathtt{c}, f) \in Valid(\mathtt{z}, \alpha_1)$ *let*

$$\delta'(\mathtt{z}, \alpha_1, \alpha_2) \stackrel{\text{def}}{=} \{\mu \mid \exists s \in wp(\mathtt{z}, \mathtt{c}, f).\ \forall i \in \{1 \ldots m\}.\ \mu(i, f(i)) = \mathtt{p}_i(s)\}$$

- $\delta'(\mathtt{z}, \cdot, \cdot)$ *is undefined if no such* α_1, α_2 *exist.*

Here, $wp(\mathtt{z}, \mathtt{c}, f) \stackrel{\text{def}}{=} \{s \in \mathtt{z} \mid \forall i, 1 \leq i \leq m.\ \mathtt{u}_i(s) \in f(i)\}$ *for* $\mathtt{c} \in Act_1$.

The set $A = wp(\mathtt{z}, \mathtt{c}, f)$ can be efficiently computed and represented [19]. To be able to apply value iteration, we need to handle the sets $\delta'(\mathtt{z}, \alpha_1, \alpha_2)$ symbolically. For $\mathtt{z} \in \mathcal{A}$, $\alpha_1 = \mathtt{c} = (\mathtt{g} \to \mathtt{p}_1 : \mathtt{u}_1 + \ldots + \mathtt{p}_m : \mathtt{u}_m) \in Valid(\mathtt{z})$ and $\alpha_2 = (\mathtt{c}, f) \in Valid(\mathtt{z}, \alpha_1)$, the part $\inf_{\mu \in \delta(s, \alpha_2, \alpha_2)} \sum_{i \in \mathbb{N}} \sum_{s' \in S} \mu(i, s') \cdot \nu(s')$ of Definition 10 can be rewritten as $\inf_{x \in A} B(x)$ with $B(x) \stackrel{\text{def}}{=} \sum_{\mathtt{z}' \in \mathcal{A}} \sum_{\substack{1 \leq i \leq m, \\ \mathtt{z}' = f(i)}} \mathtt{p}_i(x) \cdot \nu(\mathtt{z}')$. In case all $\mathtt{p}_i$ are fractions of linear functions over the program variables, i.e. there are $a_i, b_i \in \mathbb{R}$ such that $\mathtt{p}_i(\mathtt{x}_1, \ldots, \mathtt{x}_n) = \frac{a_0 + a_1 \mathtt{x}_1 + \ldots + a_n \mathtt{x}_n}{b_0 + b_1 \mathtt{x}_1 + \ldots + b_n \mathtt{x}_n}$ and the denominator is always positive, $B(x)$ can be written as $\frac{a}{b}$ where a and b are linear in the variables of $\mathtt{X}$. In turn, this optimisation problem can be handled by mixed integer quadratic programming [16]. We can ignore the restriction that the program variables are integers to obtain a quadratic program in which all variables are reals. The correctness property stated later will then still hold, but we will obtain a coarser overapproximation. Another way to compute the bounds is to specify some probability bound to hold and then to prove or to disprove it using a constraint solver. The bounds can then be refined as far as necessary by decreasing (increasing) the upper (lower) bound, in a similar way as a binary search. Notice however that a value iteration using this model will still be quite costly, because we have to solve optimisation problems for each state in each step of the value iteration algorithm.

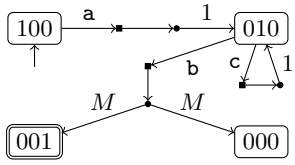

Fig. 4. Constraint Markov game abstraction of the probabilistic program of Fig. 1.

Example 4. In Fig. 4 we show the constraint Markov game abstraction of the semantics of the program in Fig. 1. We used the predicates $\mathsf{pred}_1 = (\mathtt{s} = 0)$,

$\mathsf{pred}_2 = (\mathtt{s} = 1)$, $\mathsf{pred}_3 = (\mathtt{s} = 2)$, which induce the same state space as for the game-based abstraction of Fig. 3. We have $M = \{\mu \in \mathit{Distr}(\mathcal{A}) \mid \exists x \in \mathbb{N} \cap [1, \infty).\ \mu(001) = \frac{9x-8}{16x} \wedge \mu(000) = \frac{7x+8}{16x}\}$. We can obtain the bound $[\frac{1}{16}, \frac{9}{16}]$ for the maximal reachability probability.

We define another abstraction, based on interval Markov games. This new abstraction reduces the information contained in each abstract state in order to reduce the computation and memory complexity. The reduction of information comes at the cost of introducing distributions that are not present in the original model. Roughly, the abstraction consist in maintaining the branching structure of distributions as in the previous abstraction except for the probabilities. Only upper and lower bounds are maintained.

Definition 13. *Given a probabilistic program* $(\mathtt{X}, \mathtt{Dom}, \mathtt{I}, \mathtt{C})$ *with a predicate set* $\mathsf{Pred} = \{\mathsf{pred}_1, \ldots, \mathsf{pred}_n\}$*, the* interval Markov game abstraction *is the tuple* $(\mathcal{A}, \mathcal{I}, \mathit{Act}_1, \mathit{Act}_2, \delta')$ *where* $\mathcal{A}$*,* $\mathcal{I}$*,* Act_1*,* Act_2 *and* $\mathit{Valid}(\cdot)$ *are as in Definition 12 and for* $\mathtt{z} \in \mathcal{A}$*,* $\alpha_1 = \mathtt{c}$*,* $\alpha_2 = (\mathtt{c}, f) \in \mathit{Valid}(\mathtt{z}, \alpha_1)$ *and* $\mathtt{z}' \in \mathcal{A}$*, with*

$$M(\mathtt{z}, \alpha_1, \alpha_2) \stackrel{\text{def}}{=} \{\mu \mid \exists s \in \mathit{wp}(\mathtt{z}, \mathtt{c}, f).\ \forall i \in \{1 \ldots m\}.\ \mu(i, f(i)) = \mathtt{p}_i(s)\}$$

$$\text{we let } \delta'(\mathtt{z}, \alpha_1, \alpha_2)(i, \mathtt{z}') \stackrel{\text{def}}{=} \left(\inf_{\mu \in M(\mathtt{z}, \alpha_1, \alpha_2)} \mu(i, \mathtt{z}'), \sup_{\mu \in M(\mathtt{z}, \alpha_1, \alpha_2)} \mu(i, \mathtt{z}') \right),$$

and $\delta'(\mathtt{z}, \cdot, \cdot)$ *is undefined if no such* α_1, α_2 *exist.*

All ingredients of an interval Markov game abstraction are finite. The lower bound of $\delta'(\mathtt{z}, \mathtt{c}, (\mathtt{c}, f))(i, \mathtt{z}')$ can be expressed using state variables as $\inf_{s \in \mathit{wp}(\mathtt{z}, \mathtt{c}, f)} \mathtt{p}_i(s)$, and correspondingly for the upper one. Thus, as for constraint Markov games, we have to solve optimisation problems. In contrast to the previous abstraction, optimisation problems are simpler. In addition, we have to compute the interval bounds only once and thus do not have to solve an optimisation problem in each step of the value iteration. Instead, we use an adaption of an existing algorithm for *interval Markov chains* [10] to compute these values: The optimisation over the interval part is described in the literature [10]. The optimisations over the other part of the player-2 choices and for the player-1 choices is simple, because there are only finitely many choices.

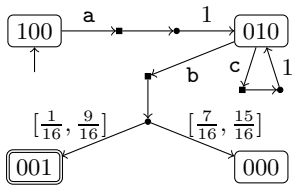

Fig. 5. Interval Markov game abstraction of the probabilistic program of Fig. 1.

Example 5. Fig. 5 shows an interval Markov game abstraction of the program in Fig. 1. We choose the same predicates as in Example 4. In this example, we obtain the same bound $[\frac{1}{16}, \frac{9}{16}]$ for the maximal reachability probability.

Theorem 2. *Consider a probabilistic program* $\mathcal{P} = (\mathtt{X}, \mathtt{Dom}, \mathtt{I}, \mathtt{C})$ *and a constraint or interval Markov game abstraction* $\mathcal{M} = (\mathcal{A}, \mathcal{I}, \mathit{Act}_1, \mathit{Act}_2, \delta')$ *of* $\mathcal{P}$*, and let* $F \subseteq \mathit{Dom}(\mathtt{X})$ *be a set of goal states. Let* $F' = \{\mathtt{z} \in \mathcal{A} \mid \mathtt{z} \cap F \neq \emptyset\}$*. Then for* $\bullet \in \{+, -\}$*, for* $\mathtt{z} \in \mathcal{A}$ *and* $s \in \mathtt{z}$ *it is*

$$p^{\bullet,-}_{\mathcal{M},F'}(\mathtt{z}) \leq p^{\bullet}_{\mathit{sem}(\mathcal{P}),F}(s) \leq p^{\bullet,+}_{\mathcal{M},F'}(\mathtt{z}).$$

5 Abstraction Refinement

In Section 4, we discussed how to compute an abstraction for a probabilistic program, given a fixed set of predicates. However, the probability bounds computed by a given abstraction might be too coarse. This section describes an abstraction refinement technique to obtain tighter probability bounds, which introduces additional predicates, to separate concrete states with a behaviour too different to be subsumed. In the following, we focus on bounds for maximal reachability probability, as the approach is analogous for the minimal case.

Definition 14. *A* player-1 strategy *for a constraint or interval Markov game* $\mathcal{M} = (S, I, Act_1, Act_2, \delta)$ *is a function* $\sigma_1 \colon S \to Act_1$ *such that* $\sigma_1(s) \in en_1(s)$ *for each state* $s \in S$*. A* player-2 action strategy *is a function* $\sigma_2 \colon (S \times Act_1) \rightharpoonup Act_2$ *that is defined for all* $(s, \alpha_1) \in S \times Act_1$ *such that* $\alpha_1 \in en(s)$*. We require that* $\sigma_2(s, \alpha_1) \in en_2(s, \alpha_1)$*. A* player-2 constraint strategy *is a function* $\sigma_{2'} \colon (S \times Act_1 \times Act_2) \rightharpoonup Distr(\mathbb{N} \times S)$ *which is defined for all* $(s, \alpha_1, \alpha_2) \in S \times Act_1 \times Act_2$ *for which* $\delta(s, \alpha_1, \alpha_2)$ *is defined. We require that* $\sigma_{2'}(s, \alpha_1, \alpha_2) \in \delta(s, \alpha_1, \alpha_2)$*. By* $\Sigma^1_{\mathcal{M}}$ *we denote the set of all player-1 strategies of* $\mathcal{M}$*, by* $\Sigma^2_{\mathcal{M}}$ *all player-2 action strategies and by* $\Sigma^{2'}_{\mathcal{M}}$ *all player-2 constraint strategies.*

Definition 15. *For* $F \subseteq S$ *and* $\nu \in Asg(S)$*, let* $pre^{\sigma_1,\sigma_2,\sigma_{2'}}_{\mathcal{M},F} : Asg(S) \to Asg(S)$ *be defined such that* $pre^{\sigma_1,\sigma_2,\sigma_{2'}}_{\mathcal{M},F}(\nu)(s) \stackrel{\text{def}}{=} 1$ *if* $s \in F$ *and otherwise we have* $pre^{\sigma_1,\sigma_2,\sigma_{2'}}_{\mathcal{M},F}(\nu)(s) \stackrel{\text{def}}{=} \sum_{s' \in S} \sum_{i \in \mathbb{N}} \delta^{\sigma_1,\sigma_2,\sigma_{2'}}(s)(i, s') \cdot \nu(s')$*. There,* $\delta^{\sigma_1,\sigma_2,\sigma_{2'}}$ *is the natural composition of the strategies* σ_1*,* σ_2 *and* $\sigma_{2'}$*. Also, let* $p^{\sigma_1,\sigma_2,\sigma_{2'}}_{\mathcal{M},F}$ *be the least fixed point of* $pre^{\sigma_1,\sigma_2,\sigma_{2'}}_{\mathcal{M},F}$*.*

An optimal $+,-$ *player-1 strategy* $\sigma^{1,+,-}_{\mathcal{M},F}$ *is an element of*

$$\arg \max_{\sigma_1 \in \Sigma^1_{\mathcal{M}}} \min_{\sigma_2 \in \Sigma^2_{\mathcal{M}}} \min_{\sigma_{2'} \in \Sigma^{2'}_{\mathcal{M}}} p^{\sigma_1,\sigma_2,\sigma_{2'}}_{\mathcal{M},F}$$

and likewise the optimal player-2 strategies and the other cases (as the order of min *and* max *does not play a role), where the infimum and supremum are to be understood componentwise.*

As constraint and interval Markov games are special cases of Markov games, optimal values $p^{\bullet_1,\bullet_2}$ can be obtained by corresponding optimal strategies $\sigma^{1,\bullet_1}$, $\sigma^{2,\bullet_2}$,$\sigma^{2',\bullet_2}$ [6]. In the above we assume compactness of the sets $\delta(s, \alpha_1, \alpha_2)$. For interval Markov games, this assumption holds automatically.

When considering pairs of optimal strategies $\sigma^{1,+,-}_{\mathcal{M},F}$, $\sigma^{1,+,+}_{\mathcal{M},F}$ for player 1 against different objectives of player 2 we assume that their decisions agree if possible [18]. We make a corresponding assumption for player 2. Based on it we define predicates called *splitters* that remove the choices that make the strategies differ from the abstract model.

Definition 16. *Let* $(\mathcal{A}, \mathcal{I}, Act_1, Act_2, \delta')$ *be a constraint or interval Markov game abstraction of a probabilistic program* $(\mathtt{X}, \mathtt{Dom}, \mathtt{I}, \mathtt{C})$*,* $\mathtt{z} \in \mathcal{A}$ *and a command* $\mathtt{c} \in \mathtt{C}$ *with* $(\mathtt{c}, f^-) \stackrel{\text{def}}{=} \sigma^{2,+,-}_{\mathcal{M},F}(\mathtt{z}, \mathtt{c}) \neq \sigma^{2,+,+}_{\mathcal{M},F}(\mathtt{z}, \mathtt{c}) \stackrel{\text{def}}{=} (\mathtt{c}, f^+)$*. The* wp-based splitter *of* $(\mathtt{z}, \mathtt{c})$ *is* $\{wp(\mathtt{z}, \mathtt{c}, f^-), wp(\mathtt{z}, \mathtt{c}, f^+)\}$*.*

For models with constant probabilities, wp-based splitting has been described before [19]. It was enough to guarantee progress of the refinement approach since $\delta'(\mathsf{z}, \alpha_1, \alpha_2)$ was a singleton. We introduce a new complementary method to split abstract states when the imprecision is due to the constraint optimisation.

Definition 17. *Let* $(\mathcal{A}, \mathcal{I}, Act_1, Act_2, \delta')$ *be a constraint or interval Markov game abstraction of a probabilistic program* $(\mathtt{X}, \mathtt{Dom}, \mathtt{I}, \mathtt{C})$, $\mathsf{z} \in \mathcal{A}$, *a command* $\mathsf{c} \in \mathtt{C}$ *and* $f\colon \{1, \ldots, m\} \to \mathcal{A}$ *with* $\mu^- \stackrel{\text{def}}{=} \sigma^{2',+,-}_{\mathcal{M},F}(\mathsf{z}, \mathsf{c}, (\mathsf{c}, f)) \neq \sigma^{2',+,+}_{\mathcal{M},F}(\mathsf{z}, \mathsf{c}, (\mathsf{c}, f)) \stackrel{\text{def}}{=} \mu^+$. *The* constraint-based splitter *of* $(\mathsf{z}, \mathsf{c}, f)$ *is* $\{\{s \in S \mid \mathtt{p}_{i_{\max}}(s) \leq \frac{U^+(i_{\max}) + U^-(i_{\max})}{2}\}\}$ *where* $U^\bullet(i) \stackrel{\text{def}}{=} \sum_{\mathsf{z} \in \mathcal{A}} \mu^\bullet(i, \mathsf{z})$ *and* $i_{\max} \stackrel{\text{def}}{=} \arg\max_{i \in \mathbb{N}} |U^+(i) - U^-(i)|$.

To refine, we choose some splitters and add them to the set of predicates. Afterwards, we recompute a refined version of the abstraction with the new predicate set. There are some heuristics [19] to choose splitters that are likely to improve the bounds.

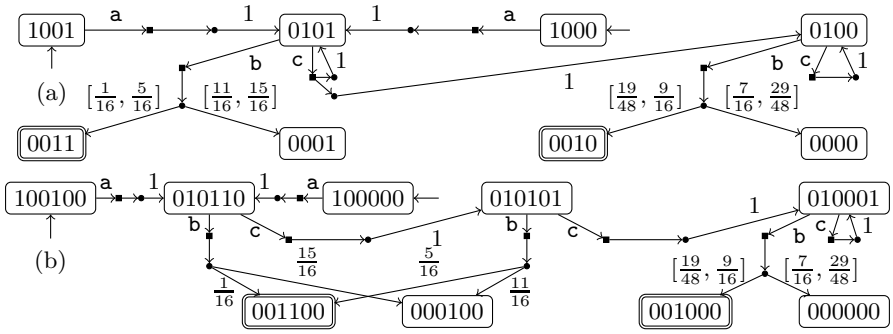

Fig. 6. Refinements of the model of Fig. 5.

Example 6. Consider the interval Markov game abstraction in Fig. 5. There is no splitter based on *wp*. Notice that this implies that if we had constant $\mathtt{p}_i$, we would have already an abstraction yielding exact reachability bounds. We can however obtain the predicate pred_4 equivalent to $(x \leq 2)$ from a splitter based on the imprecision of the interval $[\frac{1}{16}, \frac{9}{16}]$. Adding it leads to the model depicted in Fig. 6 (a). We do not get an improvement of the reachability bounds directly. However, we can now obtain a *wp*-based splitter from state 0101 and command c with predicates $\mathsf{pred}_5 = (s = 1 \wedge x \leq 1)$ and $\mathsf{pred}_6 = (s = 1 \wedge x \not\leq 1)$. The refinement in part (b) yields the improved bound $[\frac{19}{48}, \frac{9}{16}]$.

6 Experiments

We implemented the abstraction and refinement methods described in this paper in our tool Pass [7]. Empirical evaluations of the techniques have been carried out on two different case studies with varying parameters and properties. The first case study corresponds to the von Neumann NAND multiplexer [17] which is used to construct reliable devices from unreliable gates. The second case study is based on the Rubinstein's Alternating Offers Protocol [2]. For both cases, we ran

Model	param	property	total states	abs states	preds	ref (int ref)	time
NAND	20/3	p0	78322	45496	107	21	1m30s
		p4		61644	132	10	1m10s
		p8		70968	129	17	7m49s
		p12		70968	135	23	9m32s
		p20		45496	136	52	16m05s
	20 / 9	p0	308162	45469	112	28	1m59s
		p4		61644	121	13 (5)	5m24s
		p8		70968	122	29	13m31s
		p12		70968	138	15	13m10s
		p20		45469	117	34	8m7s
Alternating Offers	K 1 / 8	pval	504	101	87	6	16s
	K 2 / 8		504	108	90	15 (9)	32s
	Cinc 10 / 100		922	351	85	16 (9)	22s
	Cinc 100 / 10		504	101	94	17 (4)	24s
	T 2000		3848	133	68	4	6s
	B_RP 5000		?	351	90	21 (11)	22s

Table 1. Empirical results

the abstraction refinement loop until precise bounds were achieved. Some characteristic observations for these experiments are summarised in Table 1. Column `total states` contains the numbers of reachable states as reported by PRISM. Column `abs states` lists the numbers of abstract states handled in the final PASS refinement step, i.e. when the precise bound was obtained. Column `preds` refers to the total number of predicates used to construct the final abstraction. Column `ref` reports the total number of refinement steps, and column `int ref` refers to the number of refinement steps in which our new refinement approach for intervals was used. Column `time` reports the total time spent until PASS built a precise abstraction. As can be seen, the abstract state numbers are generally below the total state numbers, and the time consumptions are acceptable, in view of the overall work carried out. In the last model PRISM was not able to build the model due to memory exhaustion. The problem seems to be the large amount of terminal nodes introduced due to variable probabilities in conjunction with the construction of the transition matrix for the possible state space. In the NAND case study most of the time in PASS was spent on internal BDD garbage collection and node creation. This is rooted in the lack of automatic reordering of variables after each refinement step. In our experiments we have observed that the number of abstract states in the final refinement step is in the order of the number of states with non-trivial probability ($\notin \{0, 1\}$) in the original model. Considerably smaller state spaces can be obtained if one does not run the refinement loop until an exact probability results, but only until a safe bound is established.

7 Conclusion

We have introduced new abstraction and refinement methods for probabilistic systems with variable probabilities, which, unlike previous symbolic abstractions, explicitly abstract transition probabilities. While our current experimental results are already very promising, being able to deal with variable probabilities, opens up a whole spectrum of potential applications and case studies, which we would like to study, ranging from sensor network protocols to biochemical reaction cascades. The experiments also provide directions for further performance improvements of the prototype implementation in this direction.

The presented abstractions extend menu-based abstraction [19], which is distinct from the game-based abstraction defined in [14]. We conjecture that our

notion of interval abstraction could be fruitfully combined with the latter. However, the challenge would be to prevent a combinatorial explosion that results from keeping track of the interaction of different concurrent commands, and resulting interval bounds – a problem that does not occur with the present abstraction.

Acknowledgements. This work has been supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS), by the DFG/NWO Bilateral Research Programme ROCKS and by the European Union 7th Framework Programme under grant agreement number 295261 (MEALS).

References

1. de Alfaro, L., Roy, P.: Magnifying-lens abstraction for Markov decision processes. In: CAV. pp. 325–338 (2007)
2. Ballarini, P., Fisher, M., Wooldridge, M.: Automated game analysis via probabilistic model checking: a case study. ENTCS 149(2), 125–137 (2006)
3. D'Argenio, P.R., Jeannet, B., Jensen, H.E., Larsen, K.G.: Reachability analysis of probabilistic systems by successive refinements. In: PAPM-PROBMIV. pp. 39–56 (2001)
4. Fecher, H., Leucker, M., Wolf, V.: *Don't Know* in probabilistic systems. In: SPIN. LNCS, vol. 3925, pp. 71–88. Springer (2006)
5. Ferrer Fioriti, L.M., Hahn, E.M., Hermanns, H., Wachter, B.: Variable probabilistic abstraction refinement. Tech. Rep. 87, SFB/TR 14 AVACS (2012)
6. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer (1996)
7. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PASS: Abstraction Refinement for Infinite Probabilistic Models. In: TACAS. pp. 353–357 (2010)
8. Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: CAV. pp. 162–175 (2008)
9. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: LICS. pp. 266–277. IEEE Computer Society (1991)
10. Katoen, J.P., Klink, D., Leucker, M., Wolf, V.: Three-valued abstraction for probabilistic systems. Journal on Logic and Algebraic Programming pp. 1–55 (2012)
11. Kattenbelt, M., Kwiatkowska, M.Z., Norman, G., Parker, D.: Abstraction refinement for probabilistic software. In: VMCAI. LNCS, vol. 5403, pp. 182–197. Springer (2009)
12. Kattenbelt, M., Kwiatkowska, M.Z., Norman, G., Parker, D.: A game-based abstraction-refinement framework for Markov decision processes. Formal Methods in System Design 36(3), 246–280 (2010)
13. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV'11. LNCS, vol. 6806, pp. 585–591. Springer (2011)
14. Kwiatkowska, M., Norman, G., Parker, D.: Game-based abstraction for Markov decision processes. In: QEST. pp. 157–166 (2006)
15. Kwiatkowska, M.Z., Norman, G., Parker, D.: A framework for verification of software with time and probabilities. In: FORMATS. LNCS, vol. 6246, pp. 25–45. Springer (2010)
16. Lazimy, R.: Mixed-integer quadratic programming. Mathematical Programming 22, 332–349 (1982)
17. Norman, G., Parker, D., Kwiatkowska, M.Z., Shukla, S.K.: Evaluating the reliability of NAND multiplexing with PRISM. TCAD 24(10), 1629–1637 (2005)
18. Wachter, B.: Refined Probabilistic Abstraction. Ph.D. thesis, Saarland Univ. (2010)
19. Wachter, B., Zhang, L.: Best probabilistic transformers. In: VMCAI. pp. 362–379 (2010)