Probabilistic Abstractions with Arbitrary Domains

Javier Esparza and Andreas Gaiser

Fakultät für Informatik, Technische Universität München, Germany
{esparza,gaiser}@model.in.tum.de

Abstract. Recent work by Hermanns et al. and Kattenbelt et al. has extended counterexample-guided abstraction refinement (CEGAR) to probabilistic programs. These approaches are limited to predicate abstraction. We present a novel technique, based on the abstract reachability tree recently introduced by Gulavani et al., that can use arbitrary abstract domains and widening operators (in the sense of Abstract Interpretation). We show how suitable widening operators can deduce loop invariants difficult to find for predicate abstraction, and propose refinement techniques.

1 Introduction

Abstraction techniques are crucial for the automatic verification of systems with a finite but very large or infinite state space. The Abstract Interpretation framework provides the mathematical basis of abstraction [8]. Recent work has extended abstraction techniques to probabilistic systems using games [13,14,16,22,23]. The systems (e.g. probabilistic programs) are given semantics in terms of Markov Decision Processes (MDPs), which can model nondeterminism and (using interleaving semantics) concurrency. The key idea is to abstract the MDP into a stochastic 2-Player game, distinguishing between nondeterminism inherent to the system (modeled by the choices of Player 1) and nondeterminism introduced by the abstraction (modeled by Player 2). The construction ensures that the probability of reaching a goal state in the MDP using an optimal strategy is bounded from above and from below by the supremum and infimum of the probabilities of reaching the goal in the 2-Player game when Player 1 plays according to an optimal strategy (and Player 2 is free to play in any way)[1]. An analogous result holds for the probability of reaching a goal state in the MDP using a pessimal strategy.

The abstraction technique of [16,23] and the related [13,14,22] relies on predicate abstraction: an abstract state is an equivalence class of concrete states, where two concrete states are equivalent if they satisfy the same subset of a given set of predicates. The concretization of two distinct abstract states is always disjoint (the *disjointness property*). If the upper and lower bounds obtained using a set of predicates are not close enough, the abstraction is refined by adding new

[1] In [16], the roles of the players are interchanged.

E. Yahav (Ed.): SAS 2011, LNCS 6887, pp. 334–350, 2011.

predicates with the help of interpolation, analogously to the well-known CEGAR approach for non-probabilistic systems.

While predicate abstraction has proved very successful, it is known to have a number of shortcomings: potentially expensive equality and inclusion checks for abstract states, and "predicate explosion". In the non-probabilistic case, the work of Gulavani *et al.* has extended the CEGAR approach to a broader range of abstract domains [12], in which widening operations can be combined with interpolation methods, leading to more efficient abstraction algorithms. We show that the ideas of Gulavani *et al.* can also be applied to probabilistic systems, which extends the approaches of [16,17,23] to arbitrary abstract domains. Given a probabilistic program, an abstract domain and a widening for this domain, we show how to construct an abstract stochastic 2-Player reachability game. The disjointness property is not required. We prove that bounds on the probability of reaching a goal state in the MDP can be computed as in [16,23]. The proofs of [23] use the disjointness property to easily define a Galois connection between the sets of functions assigning values to the abstract and the concrete states. Since there seems to be no easy way to adapt them or the ones from [16,17] to our construction, we show the soundness of our approach by a new proof that uses different techniques.

We also propose an abstraction refinement technique that adapts the idea of delaying the application of widenings [5] to the probabilistic case. The technique delays widenings at the nodes which are likely to have a larger impact in improving the bounds. We present experimental results on several examples.

The paper is organized as follows. In the rest of the introduction we discuss related work and informally present the key ideas of our approach by means of examples. Section 2 contains preliminaries. Section 3 formally introduces the abstraction technique, and proves that games we are considering indeed give us upper resp. lower bound of the exact minimal and maximal reachability probabilities of reaching a set of states. Section 4 shows methods of refining our abstractions and discusses some experiments.

Related work. Besides [13,14,16,22,23], Monniaux has studied in [18] how to abstract probability distributions over program states (instead of the states themselves), but only considers upper bounds for probabilities, as already pointed out in [23]. In [19], Monniaux analyses different quantitative properties of Markov Decision processes, again using abstractions of probability distributions. In contrast, our approach constructs an abstraction using "non-probabilistic" domains and widenings and then performs the computation of strategies and strategy values, which might be used for a refinement of the abstraction. Finally, in [20] Hankin, Di Pierro, and Wiklicky develop a framework for probabilistic Abstract Interpretation which, loosely speaking, replaces abstract domains by linear spaces and Galois connections by special linear maps, and aims at computing expected values of random variables. In contrast, we stick to the standard framework, since in particular we wish to apply existing tools, and aim for upper and lower bounds of probabilities.

1.1 An Example

Consider the following program, written in pseudo code:

```
   int nrp = 0;
1: while (nrp < 100)
2:    if (rec_pack()) then nrp = nrp+1 else break
3: if (nrp < 1) then (fail or goto 1) else exit
```

where the choice between `fail` and `goto 1` is decided by the environment. The goal of the program is to receive up to 100 packets through a network connection. Whenever `rec_pack()` is executed, the connection can break down with probability 0.01, in which case `rec_pack()` returns false and no further packets can be received. If at least one packet is received (line 3) the program terminates[2]; otherwise, the program tries to repair the connection, which may fail or succeed, in which case the program is started again. The choice between success and failure is nondeterministic.

We formalize the pseudo code as a *Nondeterministic Probabilistic Program*, abbreviated NPP[3] (see Fig. 1). A NPP is a collection of guarded commands. A guarded command consists of a name (e.g. `A1`), followed by a guard (e.g. `(ctr = 1) & (nrp < 100)`) and a sequence of pairs of probabilities and update commands (e.g. `0.99: (nrp' = nrp+1)`), separated by '+'. A `reach`-line at the end of the NPP describes the set of states for which we want to compute the reachability probability. We call them the *final* states. In our example, reaching `fail` corresponds to satisfying `(ctr = 3) && (nrp < 1)`. A program execution starts with the initial configuration given by the variable declarations. The program chooses a guarded command whose guard is enabled (i.e., satisfied) by the current state of the program, and selects one of its update commands at random, according to the annotated probabilities. After performing the update, the process is repeated until the program reaches a final state.

The probability of reaching `fail` depends on the behaviour of the environment. The smallest (largest) probability clearly corresponds to the environment always choosing `goto 1` (`fail`), and its value is 0 (0.01). However, a brute force automatic technique will construct the standard semantics of the program, a Markov decision process (MDP) with over 400 states, part of which is shown in Fig. 1. We informally introduce our abstraction technique, which does better and is able to infer tight intervals for both the smallest and the largest probability. It is based on the parallel or menu-based predicate abstraction of [13,22], which we adapt to arbitrary abstract domains.

1.2 Constructing a Valid Abstraction

Given an abstract domain and a widening operator, we abstract the MDP of the program into four different *stochastic 2-Player games* sharing the same arena and

[2] It would be more realistic to set another bound, like 20 packets, but with one packet the probabilities are easy to compute.

[3] NPPs roughly correspond to a subset of the input language of the model checker PRISM [1].

```
  int nrp = 0, ctr = 1;
A1: (ctr = 1) & (nrp < 100)
     -> 0.99:(nrp' = nrp+1)
      + 0.01:(ctr' = 2);
A2: (ctr = 1) & (nrp >= 100)
     -> 1:(ctr' = 3);
A3: (ctr = 2) & (nrp < 1)
     -> 1:(ctr' = 1);
A4: (ctr = 2)
     -> 1:(ctr' = 3);
A5: (ctr = 3) & (nrp >= 1)
     -> 1:(ctr' = 3);
reach: (ctr = 3) & (nrp < 1)
```

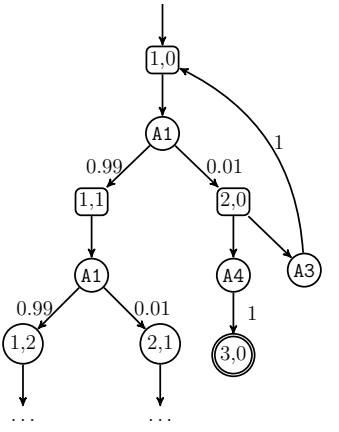

Fig. 1. Example program and prefix of the corresponding Markov Decision Process. Actions are drawn as circles, program states $\langle ctr, nrp \rangle$ as rectangles. $\langle 3, 0 \rangle$ is a final state.

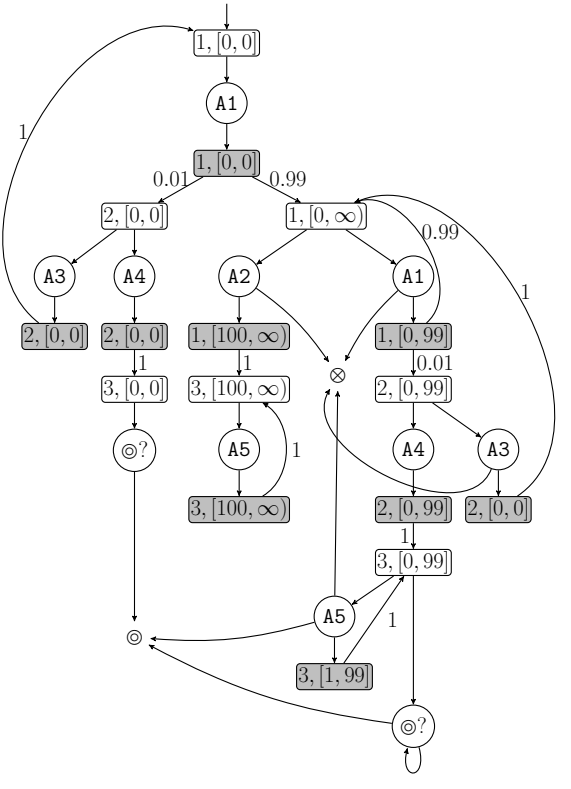

Fig. 2. Abstraction of the program from Fig. 1

the same rules (i.e., the games differ only on the winning conditions). A round of the game starts at an abstract state n (we think of n as a set of concrete states) with Player 1 to move. Let n_i be the set of concrete states of n that enable command A_i. Player 1 proposes an A_i such that $n_i \neq \emptyset$, modeled by a move from n to a node $\langle n, A_i \rangle$. If n contains some final state, then Player 1 can also propose to end the play (modeled by a move to $\langle n, \circledcirc \rangle$). Then it is Player 2's turn. If Player 1 proposes A_i, then Player 2 can accept the proposal (modeled by a move to a node determined below), or reject it and end the play (modeled by a move to another distinguished node $\otimes$), but only if $n_i \neq n$. If Player 2 accepts A_i, she moves to some node $\langle n, A_i, n' \rangle$ such that $n_i \subseteq n'$ (i.e., Player 2 can "pick" a subset n' of n_i out of the subsets offered by the game arena). The next node of the play is determined probabilistically: one of the updates of A_i is selected randomly according to the probabilities, and the play moves to the abstract state obtained by applying the update and (in certain situations) the widening operator to n'. If Player 1 proposes $\circledcirc$ by choosing $\langle n, \circledcirc \rangle$, then Player 2 can accept the proposal, (modeled by a move to $\circledcirc$) or, if not all concrete states of n are final, reject it (modeled by a move $\langle n, \circledcirc \rangle \rightarrow \langle n, \circledcirc \rangle$).

Fig. 2 shows an arena for the program of Fig. 1 with the abstract domain and widening operator described in the following. Nodes owned by Player 1 are drawn as white rectangles, nodes owned by Player 2 as circles, and probabilistic nodes as grey rectangles. In the figure we label a node $\langle n, A_i \rangle$ belonging to Player 2 with A_i and a probabilistic node $\langle n, A_i, n' \rangle$ with n' (n resp. n and A_i can easily be reconstructed by inspecting the direct predecessors). Nodes of the form $\langle n, \circledcirc \rangle$ are labeled with '$\circledcirc$?'.

A (concrete) state of the example program is a pair $\langle ctr, nrp \rangle$, and an abstract state is a pair $\langle ctr, [a, b] \rangle$, where $[a, b]$ is an interval of values of `nrp` (i.e., `ctr` is not abstracted in the example). The widening operator ∇ works as follows: if the abstract state $\langle ctr, [a, b] \rangle$ has an ancestor $\langle ctr, [a', b'] \rangle$ along the path between it and the initial state given by the construction, then we overapproximate $\langle ctr, [a, b] \rangle$ by $\langle ctr, s \rangle$, with $s = [a', b'] \nabla [\min(a, a'), \max(b, b')]$. For instance the node $n = \langle 1, [0, \infty) \rangle$ in Fig. 2 enables `A1` and `A2`, and so it has two successors. Since n contains concrete states that do not enable `A1` and concrete states that do not enable `A2`, both of them have $\otimes$ as successor. The node $\langle n, \texttt{A1}, \langle 1, [0, 99] \rangle \rangle$ (whose label is abbreviated by $\langle 1, [0, 99] \rangle$ in the figure) is probabilistic. Without widening, its successors would be $\langle 2, [0, 99] \rangle$ and $\langle 1, [1, 100] \rangle$ with probabilities 0.01 and 0.99. However, $\langle 1, [1, 100] \rangle$ has $\langle 1, [0, \infty) \rangle$ as (direct) predecessor, which has the same *ctr*-value. Therefore the widening overapproximates $\langle 1, [1, 100] \rangle$ to $\langle 1, [0, \infty) \nabla [0, \infty) \rangle = \langle 1, [0, \infty) \rangle$, and hence we insert an edge from $\langle n, \texttt{A1}, \langle 1, [0, 99] \rangle \rangle$ (back) to $\langle 1, [0, \infty) \rangle$, labeled by 0.99.

After building the arena, we compute lower and upper bounds for the minimal and maximal reachability probabilities as the *values* of four different games, defined as the winning probability of Player 1 for optimal play. The winning conditions of the games are as follows:

(a) Lower bound for the maximal probability: Player 1 wins if the play ends by reaching $\circledcirc$, otherwise Player 2 wins.

```
   int c = 0, i = 0;
1: if choice(0.5) then
2:   while (i <= 100)
3:     i = i+1;
4:     c = c-i+2
5: if (c >= i) then fail
```

```
   int c = 0, i = 0;
1: while(i <= 100)
2:   if choice(0.5) then i = (i+1);
3:   c = c-i+2;
4: if (c >= i) then fail
```

Fig. 3. Example programs 2 and 3

(b) Upper bound for the maximal probability: Players 1 and 2 both win if the play ends by reaching ◎, and both lose otherwise.
(c) Lower bound for the minimal probability: Players 1 and 2 both lose if the play ends by reaching ◎ or ⊗, and both win otherwise.
(d) Upper bound for the minimal probability: Player 1 loses if the play ends by reaching ◎ or ⊗, otherwise Player 2 loses.

For the intuition behind these winning conditions, consider first game (a). Since Player 1 models the environment and wins by reaching ◎, the environment's goal is to reach a final state. Imagine first that the abstraction is the trivial one, i.e., abstract and concrete states coincide. Then Player 2 never has a choice, and the optimal strategy for Player 1 determines a set S of action sequences whose total probability is equal to the maximal probability of reaching a final state. Imagine now that the abstraction is coarser. In the arena for the abstract game the sequences of S are still possible, but now Player 2 may be able to prevent them, for instance by moving to ⊗ when an abstract state contains concrete states not enabling the next action in the sequence. Therefore, in the abstract game the probability that Player 1 wins can be at most equal to the maximal probability. In game (b) the team formed by the two players can exploit the spurious paths introduced by the abstraction to find a strategy leading to a better set of paths; in any case, the probability of S is a lower bound for the winning probability of the team. The intuition behind games (c) and (d) is similar.

In our example, optimal strategies for game (b) are: for Player 1, always play the "rightmost" choice, except at $\langle 2, [0, 99]\rangle$, where she should play `A4`; for Player 2, play ◎ if possible, otherwise anything but ⊗. The value of the game is 1. In game (a), the optimal strategy for Player 1 is the same, whereas Player 2 always plays ⊗ (resp. stays in ◎?) whenever possible. The value of the game is 0.01. We get $[0.01, 1]$ as lower and upper bound for the maximal probability. For the minimal probability we get the trivial bounds $[0, 1]$.

To get more precision, we can skip widenings at certain situations during the construction. If we e.g. apply widening only after the second unrolling of the loop, the resulting abstraction allows us to obtain the more precise bounds $[0, 0.01]$ and $[0.01, 0.01]$ for minimal and maximal reachability, respectively.

The main theoretical result of our paper is the counterpart of the results of [16,13]: for arbitrary abstraction domains, the values of the four games described above indeed yield upper and lower bounds of the maximal and minimal probability of reaching the goal nodes.

In order to give a first impression of the advantages of abstraction domains beyond predicate abstraction in the probabilistic case, consider the (deterministic) pseudo code on the left of Fig. 3, a variant of the program above. Here `choice`(p) models a call to a random number generator that returns 1 with probability p and 0 with probability $1 - p$.

It is easy to see that $c \leq 1$ is a global invariant, and so the probability of failure is exactly 0.5. Hence a simple invariant like $c \leq k$ for a $k \leq 100$, together with the postcondition $i > 100$ of the loop would be sufficient to negate the guard of the statement at line 5. However, when this program is analysed with PASS [13,14], a leading tool on probabilistic abstraction refinement on the basis of predicate abstraction, the while loop is unrolled 100 times because the tool fails to "catch" the invariant, independently of the options chosen to refine the abstraction [4].

On the other hand, an analysis of the program with the standard interval domain, the standard widening operator, and the standard technique of delaying widenings [5], easily 'catches" the invariant (see Section 4.1). The same happens for the program on the right of the figure, which exhibits a more interesting probabilistic behaviour, especially a probabilistic choice within a loop: we obtain good upper and lower bounds for the probability of failure using the standard interval domain. Notice that examples exhibiting the opposite behaviour (predicate abstraction succeeds where interval analysis fails) are not difficult to find; our thesis is only that the game-based abstraction approach of [13,16] can be extended to arbitrary abstract domains, making it more flexible and efficient.

2 Stochastic 2-Player Games

This section introduces stochastic 2-Player games. For a more thorough introduction into the subject and proofs for the theorems see e.g. [21,6,7].

Let S be a countable set. We denote by $\mathrm{Dist}(S)$ the set of all distributions $\delta : S \to [0, 1]$ over S with $\delta(x) = 0$ for all but finitely many $x \in S$.

Definition 1. *A* stochastic 2-Player game $\mathcal{G}$ *(short 2-Player game) is a tuple* $((V_1, V_2, V_p), E, \delta, s_0)$*, where*

- V_1, V_2, V_p *are distinct, countable sets of states. We set* $V = V_1 \cup V_2 \cup V_p$*;*
- $E \subseteq (V_1 \cup V_2) \times V$ *is the set of* admissible *player choices;*
- $\delta : V_p \to \mathit{Dist}(V)$ *is a probabilistic transition function;*
- $s_0 \in V_1$ *is the start state.*

Instead of $(q, r) \in E$ *we often write* $q \to r$*. A string* $w \in V^+$ *is a* finite run *(short: run) of* $\mathcal{G}$ *if (a)* $w = s_0$*, or (b)* $w = w's's$ *for some run* $w's' \in V^*(V_1 \cup V_2)$ *and* $s' \to s$*, or (c)* $w = w's's$ *for some run* $w's' \in V^*V_p$ *such that* $\delta(s')(s) > 0$*. We denote the set of all runs of* $\mathcal{G}$ *by* $\mathit{Cyl}(\mathcal{G})$*. A run* $w = x_1 \dots x_k$ *is* accepting

[4] Actually, the input language of PASS does not explicitly include while loops, they have to be simulated. But this does not affect the analysis.

relative to $F \subseteq V_1$ *if* $x_k \in F$ *and* $x_i \notin F$ *for* $1 \leq i < k$. *The set of accepting runs relative to* F *is denoted by* $Cyl(\mathcal{G}, F)$.

A stochastic 2-Player game with $V_2 = \emptyset$ *is called a* Markov Decision Process (MDP)*, and then we write* $\mathcal{G} = ((V_1, V_p), E, \delta, s_0)$.

Fix for the rest of the section a 2-Player game $\mathcal{G} = ((V_1, V_2, V_p), E, \delta, s_0)$. The behaviours of Player 1 and 2 in $\mathcal{G}$ are described with the help of *strategies*:

Definition 2. *A strategy for Player* $i \in \{1, 2\}$ *in* $\mathcal{G}$ *is a partial function* $\phi : Cyl(\mathcal{G}) \to Dist(V)$ *satisfying the following two conditions:*

- $\phi(w)$ *is defined iff* $w = w'v \in V^*V_i$ *and* $v \to x$ *for some* $x \in V$*; and*
- *if* $\phi(w)$ *is defined and* $\phi(w)(x) > 0$ *then* wx *is a run.*

We denote the set of strategies for Player i *by* $S_i(\mathcal{G})$. *A strategy* ϕ *is* memoryless *if* $\phi(w_1) = \phi(w_2)$ *for any two runs* w_1, w_2 *ending in the same node of* V_i*, and* non-randomized *if for every run* w *such that* $\phi(w)$ *is defined there is a node* x *such that* $\phi(w)(x) = 1$. *Given strategies* ϕ_1, ϕ_2 *for Players 1 and 2, the* value $val\,(w)_{\mathcal{G}[\phi_1,\phi_2]}$ *of a run* w *under* ϕ_1, ϕ_2 *is defined as follows:*

- *If* $w = s_0$*, then* $val\,(w)_{\mathcal{G}[\phi_1,\phi_2]} = 1$.
- *If* $w = w's \in V^*V_i$ *for* $i \in \{1, 2\}$ *and* $\phi_i(w')$ *is defined, then* $val\,(w)_{\mathcal{G}[\phi_1,\phi_2]} = val\,(w')_{\mathcal{G}[\phi_1,\phi_2]} \cdot \phi_i(w')(s)$.
- *If* $w = w's's$ *for some run* $w's' \in V^*V_p$ *then* $val\,(w)_{\mathcal{G}[\phi_1,\phi_2]} = val\,(w's')_{\mathcal{G}[\phi_1,\phi_2]} \cdot \delta(s')(s)$.
- *Otherwise* $val\,(w)_{\mathcal{G}[\phi_1,\phi_2]} = 0$.

We are interested in *probabilistic reachability*:

Definition 3. *The* probability $Reach(\mathcal{G}[\phi_1, \phi_2], F)$ of reaching $F \subset V_1$ *in* $\mathcal{G}$ *under strategies* ϕ_1 *and* ϕ_2 *of Players 1 and 2 is*

$$Reach(\mathcal{G}[\phi_1, \phi_2], F) := \sum_{w \in Cyl(\mathcal{G}, F)} val\,(w)_{\mathcal{G}[\phi_1,\phi_2]}.$$

If the context is clear, we often omit the subscript of $\mathrm{val}\,(\cdot)$. We write $\mathrm{Cyl}(\mathcal{G}[\phi_1, \phi_2])$ (resp. $\mathrm{Cyl}(\mathcal{G}[\phi_1, \phi_2], F)$) for the set of all finite runs $r \in \mathrm{Cyl}(\mathcal{G})$ (resp. $r \in \mathrm{Cyl}(\mathcal{G}, F)$) with $\mathrm{val}\,(r)_{\mathcal{G}[\phi_1,\phi_2]} > 0$. In a MDP $\mathcal{M}$ we do not require to have a strategy for the second Player. Here we just write $\mathrm{Reach}(\mathcal{M}[\phi_1], F)$ for a given strategy $\phi_1 \in S_1(\mathcal{M})$.

Definition 4. *Let* $\mathcal{G} = ((V_1, V_2, V_p), E, \delta, s_0)$ *be a 2-Player game, and* $F \subset V_1$. *The* extremal game values $Reach(\mathcal{G}, F)^{++}, Reach(\mathcal{G}, F)^{+-}, Reach(\mathcal{G}, F)^{-+}$ *and* $Reach(\mathcal{G}, F)^{--}$ *are*

$$Reach(\mathcal{G}, F)^{++} := \sup_{\phi_1 \in S_1(\mathcal{G})} \sup_{\phi_2 \in S_2(\mathcal{G})} Reach(\mathcal{G}[\phi_1, \phi_2], F)$$

$$Reach(\mathcal{G}, F)^{+-} := \sup_{\phi_1 \in S_1(\mathcal{G})} \inf_{\phi_2 \in S_2(\mathcal{G})} Reach(\mathcal{G}[\phi_1, \phi_2], F)$$

$$Reach(\mathcal{G}, F)^{-+} := \inf_{\phi_1 \in S_1(\mathcal{G})} \sup_{\phi_2 \in S_2(\mathcal{G})} Reach(\mathcal{G}[\phi_1, \phi_2], F)$$

$$Reach(\mathcal{G}, F)^{--} := \inf_{\phi_1 \in S_1(\mathcal{G})} \inf_{\phi_2 \in S_2(\mathcal{G})} Reach(\mathcal{G}[\phi_1, \phi_2], F)$$

If $\mathcal{G}$ is a MDP, we define $Reach(\mathcal{G},F)^{+} := Reach(\mathcal{G},F)^{++} = Reach(\mathcal{G},F)^{+-}$ and $Reach(\mathcal{G},F)^{-} := Reach(\mathcal{G},F)^{--} = Reach(\mathcal{G},F)^{-+}$.

The following well-known theorem will be crucial for the validity of our abstractions [6]:

Theorem 1. *Let $F \subset V_1$. For each $\kappa \in \{++,+-,-+,--\}$ there exist* non-randomized and memoryless *strategies $\phi_1^\kappa \in S_1(\mathcal{G}), \phi_2^\kappa \in S_2(\mathcal{G})$ such that*

$$Reach(\mathcal{G},F)^{\kappa} = Reach(\mathcal{G}[\phi_1^\kappa,\phi_2^\kappa],F).$$

Extremal game values can be computed e.g. by variants of value iteration [7].

3 Abstractions of Probabilistic Programs

We start by giving a formal definition of NPPs.

Definition 5. *Let $\mathcal{V}$ be a finite set of variables, where $x \in \mathcal{V}$ has a range $rng(x)$. A* configuration *(or state) of $\mathcal{V}$ is a map $\sigma\colon \mathcal{V} \to \bigcup_{x\in\mathcal{V}} rng(x)$ such that $\sigma(x) \in rng(x)$ for all $x \in \mathcal{V}$. The set of all configurations is denoted by $\Sigma_\mathcal{V}$. A* transition *is a map $f \in 2^{\Sigma_\mathcal{V}} \to 2^{\Sigma_\mathcal{V}}$ such that $|f(\{\sigma\})| \leq 1$ for all $\sigma \in \Sigma_\mathcal{V}$ (i.e., a transition maps a single configuration to the empty set or to a singleton again), and*

$$\bigcup_{\sigma\in M} f(\{\sigma\}) = f(M) \text{ for all } M \subseteq \Sigma_\mathcal{V}.$$

A transition g is a guard *if $g(\{\sigma\}) \in \{\{\sigma\},\emptyset\}$ for every configuration σ. We say that σ* enables *g if $g(\{\sigma\}) = \{\sigma\}$. A transition c is an* assignment *if $|c(\{\sigma\})| = 1$ for all $\sigma \in \Sigma_\mathcal{V}$. The semantics of an assignment c is the map $[\![c]\!] : \Sigma_\mathcal{V} \to \Sigma_\mathcal{V}$ given by $[\![c]\!](\sigma) := \sigma'$ if $c(\{\sigma\}) = \{\sigma'\}$. The set of transitions is denoted by $Trans_\mathcal{V}$.*

Definition 6. *Nondeterministic Probabilistic Programs.*
A nondeterministic probabilistic program *(NPP) is a triple $P = (\mathcal{V},\sigma_0,\mathcal{C})$ where $\mathcal{V}$ is a finite set of program* variables*, $\sigma_0 \in \Sigma_\mathcal{V}$ is the* initial configuration*, and $\mathcal{C}$ is a finite set of* guarded commands*. A* guarded command *A has the form $A = g \to p_1 : c_1 + \ldots + p_m : c_m$, where $m \geq 1$, g is a guard, $p_1,\ldots,p_m$ are probabilities adding up to 1, and $c_1,\ldots,c_m$ are assignments. We denote the guard of A by g_A, the* updates *$\{\langle p_1,c_1\rangle,\ldots\langle p_m,c_m\rangle\}$ of A by up_A, and the set $\{up_A \mid A \in \mathcal{C}\}$ by $up_\mathcal{C}$.*

Definition 7. *Semantics of NPPs and Reachability Problem.*
The MDP associated to *a NPP $P = (\mathcal{V},\sigma_0,\mathcal{C})$ is $\mathcal{M}_P = ((V_1,V_p),E,\delta,\sigma_0)$, where $V_1 = \Sigma_\mathcal{V}$, $V_p = \Sigma_\mathcal{V} \times \mathcal{C}$, $E \subseteq V_1 \times (V_1 \cup V_p)$, $\delta : (\Sigma_\mathcal{V} \times \mathcal{C}) \to Dist(V_1)$, and for every $A \in \mathcal{C}$, $\sigma,\sigma' \in \Sigma_\mathcal{V}$*

$$\sigma \to \langle\sigma,A\rangle \text{ iff } \sigma \text{ enables } g_A \text{ and } \delta(\langle\sigma,A\rangle)(\sigma') := \sum_{\langle p,c\rangle\in up_A:\ [\![c]\!](\sigma)=\sigma'} p\ .$$

The reachability problem *for P relative to a set $F \subseteq \Sigma_\mathcal{V}$ of states such that $\sigma_0 \notin F$ is the problem of computing $Reach(\mathcal{M}_P,F)^{+}$ and $Reach(\mathcal{M}_P,F)^{-}$. We call F the set of* final states*.*

We assume in the following that for every run $w\sigma \in V^*V_1$ in $\mathcal{M}_P$ either $\sigma \in F$ or σ enables the guard of at least one command (i.e., we do not 'get stuck' during the computation). This can e.g. be achieved by adding a suitable guarded command that simulates a self loop.

3.1 Abstracting NPPs

We abstract NPPs using the Abstract Interpretation framework (see [8]). As usual, an abstract domain is a complete lattice $(D^\sharp, \sqsubseteq, \top, \bot, \sqcup, \sqcap)$ (short $D^\sharp$), and we assume the existence of monotone abstraction and concretization maps $\alpha : 2^{\Sigma_\mathcal{V}} \to D^\sharp$ and $\gamma : D^\sharp \to 2^{\Sigma_\mathcal{V}}$ forming a Galois connection between $D^\sharp$ and $2^{\Sigma_\mathcal{V}}$. A *widen operator* is a mapping $\nabla : D^\sharp \times D^\sharp \to D^\sharp$ satisfying (i) $a \nabla b \sqsupseteq a$ and $a \nabla b \sqsupseteq b$ for all $a, b \in D^\sharp$, and (ii) for every strictly increasing sequence $a_0 \sqsubset a_1 \sqsubset \ldots$ in $D^\sharp$ the sequence $(b_i)^{i \in \mathbb{N}}$ defined by $b_0 = a_0$ and $b_{i+1} = b_i \nabla a_{i+1}$ is stationary.

We abstract sets of configurations by elements of $D^\sharp$. Following ideas from [16,13,14,22], the abstraction of an NPP is a 2-player stochastic game. We formalize which games are *valid abstractions* of a given NPP (compare the definition to the comments in Section 1.2):

Definition 8. *Let $P = (\mathcal{V}, \sigma_0, \mathcal{C})$ be a NPP with a set $F \subseteq \Sigma_\mathcal{V}$ of final states such that $\sigma_0 \notin F$. A 2-player game $\mathcal{G} = ((V_1, V_2, V_p), E, \delta, s_0)$ with finitely many nodes is a* valid abstraction *of P relative to F for $D^\sharp$ if*

- *V_1 contains a subset of $D^\sharp$ plus two distinguished states $\circledcirc, \otimes$;*
- *V_2 is a set of pairs $\langle s, A\rangle$, where $s \in V_1 \setminus \{\circledcirc, \otimes\}$ and either $A = \circledcirc$ or A is a command of $\mathcal{C}$ enabled by some state of $\gamma(s)$;*
- *V_p is a set of fourtuples $\langle s, A, s', d\rangle$, where $s, s' \in V_1 \setminus \{\circledcirc, \otimes\}$ such that $s \sqsupseteq s'$, A is a command enabled by some state of $\gamma(s')$, and d is the mapping that assigns to every update $\langle p, c\rangle \in up_A$ an abstract state $s' \in V_1$ with $\gamma(d(\langle p, c\rangle)) \supseteq c(\gamma(s'))$;*
- *$s_0 = \alpha(\{\sigma_0\})$;*

and the following conditions hold:

1. *For every $s \in V_1 \setminus \{\circledcirc, \otimes\}$ and every $A \in \mathcal{C}$:*
   (a) *If $\gamma(s) \cap F \neq \emptyset$ then $s \to \langle s, \circledcirc\rangle \to \circledcirc$. If moreover $\gamma(s) \subseteq F$, then $\langle s, \circledcirc\rangle$ is the only successor of s; otherwise, also $\langle s, \circledcirc\rangle \to \langle s, \circledcirc\rangle$ holds.*
   (If $\gamma(s)$ contains some final state, then Player 1 can propose $\circledcirc$. If all states of $\gamma(s)$ are final, then Player 2 must accept, otherwise it can accept, or reject by staying in $\langle s, \circledcirc\rangle$. *)*
   (b) *If $g_A(\gamma(s)) \neq \emptyset$ then $\langle s, A\rangle \in V_2$ and $s \to \langle s, A\rangle$.*
   (If some state of $\gamma(s)$ enables A then Player 1 can propose A. *)*
2. *For every pair $\langle s, A\rangle \in V_2$ and every $A \in \mathcal{C}$:*
   (a) *there exist nodes $\{\langle s, A, s_1, d_1\rangle, \ldots, \langle s, A, s_k, d_k\rangle\} \subseteq V_p$ such that $\langle s, A\rangle \to \langle s, A, s_i, d_i\rangle$ for every $i \leq k$ and $g_A(\gamma(s)) \subseteq \bigcup_{j=1}^{k} \gamma(s_j)$.*
   (If Player 2 accepts A, then she can pick any concrete state $\sigma \in \gamma(s)$ enabling A, and choose a successor $\langle s, A, s_i, d_i\rangle$ such that $\sigma \in \gamma(s_i)$. *)*

(b) *If* $\gamma(s) \cap F \neq \emptyset$, *then* $\langle s, A\rangle \to \circledcirc$.
(If* $\gamma(s)$ *contains some final state, then Player 2 can reject and move to* $\circledcirc$. **)*

(c) *If* $g_A(\gamma(s)) \neq \gamma(s)$, *then* $\langle s, A\rangle \to \otimes$.
(If some state of* $\gamma(s)$ *does not enable* A, *then Player 2 can reject* A *and move to* $\otimes$. **)*

3. *For every* $\langle s, A, s', d\rangle \in V_p$ *and every abstract state* $s'' \in V_1$*:*

$$\delta(\langle s, A, s', d\rangle)(s'') := \sum_{\langle p,c\rangle \in up_A\,:\, d(\langle p,c\rangle)=s''} p \ .$$

4. *The states* $\otimes$ *and* $\circledcirc$ *have no outgoing edges.*

We can now state the main theorem of the paper: the extremal game values of the games derived from valid abstractions provide upper and lower bounds on the maximal and minimal reachability probabilities. The complete proof is given in [10].

Theorem 2. *Let* P *be a NPP and let* $\mathcal{G}$ *be a valid abstraction of* P *relative to* F *for the abstract domain* $D^\sharp$. *Then*

$$\mathit{Reach}(\mathcal{M}_P, F)^- \in [\mathit{Reach}(\mathcal{G}, \{\circledcirc, \otimes\})^{--}, \mathit{Reach}(\mathcal{G}, \{\circledcirc, \otimes\})^{-+}] \text{ and}$$
$$\mathit{Reach}(\mathcal{M}_P, F)^+ \in [\mathit{Reach}(\mathcal{G}, \{\circledcirc\})^{+-}, \mathit{Reach}(\mathcal{G}, \{\circledcirc\})^{++}].$$

Proof. (*Sketch.*) The result is an easy consequence of the following three assertions:

(1) Given a strategy ϕ of the (single) player in $\mathcal{M}_P$, there exists a strategy $\phi_1 \in S_1(\mathcal{G})$ such that

$$\inf_{\psi \in S_2(\mathcal{G})} \mathrm{Reach}(\mathcal{G}[\phi_1, \psi], \{\circledcirc, \otimes\}) \leq \mathrm{Reach}(\mathcal{M}_P[\phi], F) \text{ and}$$

$$\sup_{\psi \in S_2(\mathcal{G})} \mathrm{Reach}(\mathcal{G}[\phi_1, \psi], \{\circledcirc\}) \geq \mathrm{Reach}(\mathcal{M}_P[\phi], F).$$

(2) Given a strategy $\phi_1 \in S_1(\mathcal{G})$ there exists a strategy $\phi \in S_1(\mathcal{M}_P)$ such that

$$\mathrm{Reach}(\mathcal{M}_P[\phi], F) \leq \sup_{\psi \in S_2(\mathcal{G})} \mathrm{Reach}(\mathcal{G}[\phi_1, \psi], \{\otimes, \circledcirc\}).$$

(3) Given a strategy $\phi_1 \in S_1(\mathcal{G})$ there exists a strategy $\phi \in S_1(\mathcal{M}_P)$ such that

$$\mathrm{Reach}(\mathcal{M}_P[\phi], F) \geq \inf_{\psi \in S_2(\mathcal{G})} \mathrm{Reach}(\mathcal{G}[\phi_1, \psi], \{\circledcirc\}).$$

To prove (1) (the other two assertions are similar), we use ϕ to define a function D that distributes the probabilistic mass of a run $R \in \mathrm{Cyl}(\mathcal{G})$ among all the runs $r \in \mathrm{Cyl}(\mathcal{M})$ (where $\mathcal{M}$ is a normalization of $\mathcal{M}_P$). The strategies ϕ_1 and ϕ_2 are then chosen so that they produce the same distribution, i.e., the mass of all the runs r that follow the strategies and correspond to an abstract run R following ϕ is equal to the mass of R. □

Recall that in predicate abstraction the concretizations of two abstract states are disjoint sets of configurations (disjointness property). This allows to easily define a Galois connection between the sets of functions assigning values to the abstract and the concrete states: Given a concrete valuator f, its abstraction is the function that assigns to a set X the minimal resp. the maximal value assigned by f to the elements of X. Here we have to distribute the value of a concrete state into multiple abstract states (which is what we do in our proof).

3.2 An Algorithm for Constructing Valid Abstractions

Algorithm 1 builds a valid abstraction $\mathcal{G}$ of a NPP P relative to a set F of final states for a given abstract domain $D^\sharp$. It is inspired by the algorithms of [4,11] for constructing abstract reachability trees. It constructs the initial state $s_0 = \alpha(\{\sigma_0\})$ and generates transitions and successor states in a breadth-first fashion using a work list called work. The `GENSUCCS` procedure constructs the successors of a node guided by the rules from Def. 8. It uses abstract transformers $g^\sharp$ and $c^\sharp$ for the guards and commands of the NPP. Hereby a transformer $g^\sharp : D^\sharp \to 2^{D^\sharp}$ abstracting a guard g has to satisfy that for all $a \in D^\sharp$, $g^\sharp(a)$ is finite and $\bigcup_{b \in g^\sharp(a)} \gamma(b) \supseteq g(\gamma(a))$. Allowing $g^\sharp$ to return a set rather than just one element from $D^\sharp$ can help increasing the accuracy of $\mathcal{G}$. Here we implicitly make use of abstract powerset domains. `GENSUCCS` assumes that it can be decided whether $\gamma(s) \cap F = \emptyset$, $\gamma(s) \not\subseteq F$, $g_A(\gamma(s)) \neq \emptyset$ or $g_A(\gamma(s)) \neq \gamma(s)$ hold (lines 2 and 4). The assumptions on F are reasonable, since in most cases the set F has a very simple shape, and could be replaced by conservative tests on the abstract. A conservative decision procedure suffices for the test $g_A(\gamma(s)) \neq \gamma(s)$, with the only requirement that if it returns 0, then $g_A(\gamma(s)) = \gamma(s)$ has to hold. The same holds for the test $g_A(\gamma(s)) \neq \emptyset$. `GENSUCCS` closely follows the definition of a valid abstraction, as specified in Def. 8.

Lines 1 and 2 guarantee that condition (1a) of Def. 8 holds, and, similarly, line 4 guarantees condition (1b). Similarly, lines 3 and 5 are needed to satisfy conditions (2b) and (2c), respectively. The loop at line 6 generates the nodes of the form $\langle s, A, s_i, d\rangle$ required by condition (2a) of our definition, and the loop at line 7 constructs the function d appearing in condition 3.

As usual, termination of the algorithm requires to use widenings. This is the role of the `EXTRAPOLATE` procedure. During the construction, we use the function $\mathrm{pred}(\cdot)$ to store for every node $s \in V_1 \setminus \{s_0, \circledcirc, \otimes\}$ its predecessor in the spanning tree induced by the construction (we call it *the spanning tree* from now on). For a node $s' \in V_1$ that was created as the result of chosing a guarded command A, the procedure finds the nearest predecessor s in the spanning tree with the same property, and uses s to perform a widen operation. Note that in the introductory example, another strategy was used: There we applied widenings only for states with matching control location. The strategy used in `EXTRAPOLATE` does not use additional information like control flow and thus can be used for arbitrary NPPs. We can now prove (see [10]):

Theorem 3. *Algorithm 1 terminates, and its result $\mathcal{G}$ is a valid abstraction.*

Algorithm 1: Computing $\mathcal{G}$.

Input: NPP $P = (\mathcal{V}, \sigma_0, \mathcal{C})$, abstract domain $D^\sharp$, set of final states $F \subseteq \Sigma_\mathcal{V}$, widening ∇.
Output: 2-Player game $\mathcal{G} = ((V_1, V_2, V_p), E, \delta, s_0)$.

$s_0 = \alpha(\{\sigma_0\})$; $\text{pred}(s_0) \leftarrow$ nil
$V_1 \leftarrow \{s_0, \otimes, \circledcirc\}$; $V_2 \leftarrow \emptyset$; $V_p \leftarrow \emptyset$; work $\leftarrow \{s_0\}$
while *work* $\neq \emptyset$ **do**
  Remove s from the head of work; `GENSUCCS`(s)

Procedure `GENSUCCS`$(s \in V_1)$
fopt $\leftarrow$ *false*
if $\gamma(s) \cap F \neq \emptyset$ **then**
1 $E \leftarrow E \cup \{(s, \langle s, \circledcirc\rangle), (\langle s, \circledcirc\rangle, \circledcirc)\}$
2 **if** $\gamma(s) \not\subseteq F$ **then** { $E \leftarrow E \cup \{(\langle s, \circledcirc\rangle, \langle s, \circledcirc\rangle)\}$; fopt $\leftarrow$ *true* }
  else return
forall the $A \in \mathcal{C}$ **do**
  if $g_A(\gamma(s)) \neq \emptyset$ **then**
3 $V_2 \leftarrow V_2 \cup \{\langle s, A\rangle\}$; $E \leftarrow E \cup \{(s, \langle s, A\rangle)\}$
4 **if** $g_A(\gamma(s)) \neq \gamma(s)$ **then** $E \leftarrow E \cup \{(\langle s, A\rangle, \otimes)\}$
5 **if** *fopt* **then** $E \leftarrow E \cup \{(\langle s, A\rangle, \circledcirc)\}$
6 **forall the** $s' \in g_A^\sharp(s)$ **do**
      Create a fresh array $d : up_\mathcal{C} \to V_1$
7 **forall the** $\langle p, c\rangle \in up_A$ **do**
        $v \leftarrow$ `EXTRAPOLATE`$(c^\sharp(s), s, A)$; $d(\langle p, c\rangle) \leftarrow v$
        if $v \notin V_1$ **then** {
        $V_1 \leftarrow V_1 \cup \{v\}$; $\text{pred}(v) = \langle s, A\rangle$; add v to work }
      $V_p \leftarrow V_p \cup \{\langle s, A, s', d\rangle\}$; $E \leftarrow E \cup \{(\langle s, A\rangle, \langle s, A, s', d\rangle)\}$

Procedure `EXTRAPOLATE`$(v \in D^\sharp, s \in V_1 \setminus \{\circledcirc, \otimes\}, A \in \mathcal{C})$
$\langle s', A'\rangle \leftarrow \text{pred}(s)$
while *pred*$(s') \neq$ *nil* **do**
  if $A' = A$ **then** return $s \nabla (s \sqcup v)$
  else { buffer $\leftarrow s'$; $\langle s', A'\rangle \leftarrow \text{pred}(s')$; $s \leftarrow$ buffer }
return v

4 Refining Abstractions: Quantitative Widening Delay

Algorithm 1 applies the widening operator whenever the current node has a predecessor in the spanning tree that was created by the application applying the same guarded command. This strategy usually leads to too many widenings and poor abstractions. A popular solution in non-probabilistic abstract interpretation is to delay widenings in an initial stage of the analysis [5], in our case until the spanning tree reaches a given depth. We call this approach *depth-based unrolling*.

Note that if $\mathcal{M}_P$ is finite and the application of widenings is the only source of imprecision, this simple refinement method is complete.

A shortcoming of this approach is that it is insensitive to the probabilistic information. We propose to combine it with another heuristic. Given a valid abstraction $\mathcal{G}$, our procedure yields two pairs (ϕ_1^+, ϕ_2^+) resp. (ϕ_1^-, ϕ_2^-) of memoryless and non-probabilistic strategies that satisfy $\text{Reach}(\mathcal{G}[\phi_1^-, \phi_2^-], \{\circledcirc\}) = \text{Reach}(\mathcal{G}, \{\circledcirc\})^{+-}$ resp. $\text{Reach}(\mathcal{G}[\phi_1^+, \phi_2^+], \{\circledcirc\}) = \text{Reach}(\mathcal{G}, \{\circledcirc\})^{++}$. Given a node s for Player 1, let P_s^+ and P_s^- denote the probability of reaching $\circledcirc$ (resp. $\circledcirc$ or $\otimes$ if we are interested in minimal probabilities) starting at s and obeying the strategies (ϕ_1^+, ϕ_2^+) resp. (ϕ_1^-, ϕ_2^-) in $\mathcal{G}$. In order to refine $\mathcal{G}$ we can choose any node $s \in V_1 \cap D^\sharp$ such that $P_s^+ - P_s^- > 0$ (i.e., a node whose probability has not been computed exactly yet), such that at least one of the direct successors of s in the spanning tree has been constructed using a widening. We call these nodes the *candidates* (for delaying widening). The question is which candidates to select. We propose to use the following simple heuristic:

> Sort the candidates s according to the product $w_s \cdot (P_s^+ - P_s^-)$, where w_s denotes the product of the probabilities on the path of the spanning tree of $\mathcal{G}$ leading from s_0 to s. Choose the n candidates with largest product, for a given n.

We call this heuristic the *mass heuristic*. The *mixed heuristic* delays widenings for nodes with depth less than a threshold i, and for n nodes of depth larger than or equal to i with maximal product. In the next section we illustrate depth-based unrolling, the mass heuristic, and the mixed heuristic on some examples.

4.1 Experiments

We have implemented a prototype of our approach on top of the Parma Polyhedra Library [3], which provides several numerical domains [2]. We present some experiments showing how simple domains like intervals can outperform predicate abstraction. Notice that examples exhibiting the opposite behaviour are also easy to find: our experiments are not an argument against predicate abstraction, but an argument for abstraction approaches not limited to it.

If the computed lower and upper bounds differ by more than 0.01, we select refinement candidates using the different heuristics presented before and rebuild the abstraction. We used a Linux machine with 4GB RAM.

Two small programs. Consider the NPPs of Fig. 4. We compute bounds with different domains: intervals, octagons, integer grids, and the product of integer grids and intervals [9]. For the refinement we use the mass (M) depth (D) and mixed (Mix) heuristics. For M and Mix we choose 15 refinement candidates at each iteration. The results are shown in Table 1. For the left program the integer grid domain (and the product) compute precise bounds after one iteration. After 10 minutes, the PASS tool [13] only provides the bounds $[0.5, 0.7]$ for the optimal reachability probability. For the right program only the product of grids and intervals is able to "see" that $x \equiv 0 \pmod 3$ or $y < 30$ holds, and yields

```
int a=0, ctr=0;
A1: (ctr=0)
     -> 0.5:(a'=1)&(ctr'=1)
       +0.5:(a'=0)&(ctr'=1);
A2: (ctr=1)&(a>=-400)&(a<= 400)
     -> 0.5:(a'=a+5)
       +0.5:(a'=a-5);
A3: (ctr=1) -> 1:(ctr'=2);
reach: (a=1)&(ctr=2)
```

```
int x=0, y=0, c=0;
A1: (c=0)&(x<=1000)
     -> 0.25:(x'=3*x+2)&(y'=y-x)
       +0.75:(x'=3*x)&(y'=30);
A2: (c=0)&(x>1000) -> 1:(c'=1);
A3: (c=1)&(x>=3) -> 1:(x'=x-3);
reach: (c=1)&(x=2)&(y>=30)
```

Fig. 4. Two guarded-command programs

precise bounds after 3 refinement steps. After 10 minutes PASS only provides the bounds $[0, 0.75]$. The example illustrates how pure depth-based unrolling, ignoring probabilistic information, leads to poor results: the mass and mixed heuristics perform better. PASS may perform better after integrating appropriate theories, but the example shows that combining domains is powerful and easily realizable by using Abstract Interpretation tools.

Programs of Fig. 3. For these PASS does not terminate after 10 minutes, while with the interval domain our approach computes the exact value after at most 5 iterations and less than 10 seconds. Most of the predicates added by PASS during the refinement for program 2 have the form $\mathtt{c} \leq \alpha \cdot \mathtt{i} + \beta$ with $\alpha > 0, \beta < 0$: PASS's interpolation engines seem to take the wrong guesses during the generation of new predicates. This effect remains also if we change the refinement strategies of PASS. PASS offers the option of manually adding predicates. Interestingly it suffices to add a predicate as simple as e.g. $i > 3$ to help the tool deriving the solution after 3 refinements for program 2.

Zeroconf. This is a simple probabilistic model of the Zeroconf protocol, adapted from [1,16], where it was analyzed using PRISM and predicate abstraction. It is parameterized by K, the maximal number of probes sent by the protocol. We check it for $K = 4, 6, 8$ and two different properties. *Zeroconf* is a very good example for predicate abstraction, and so it is not surprising that PASS beats the interval domain (see Table 2). The example shows how the mass heuristic by

Table 1. Experimental results for the programs in Fig. 4. Iters is the number of iterations needed. Time is given in seconds. '-' means the analysis did not return a precise enough bound after 10 minutes. Size denotes the maximal number of nodes belonging to Player 1 that occured in one of the constructed games.

Program	Value	Interval			Octagon			Grid			Product		
		M	D	Mix	M	D	Mix	M	D	Mix	M	D	Mix
Left	Iters:	23	81	24	28	81	28	1	1	1	1	1	1
	Time:	25	66.1	27.6	26.6	63.2	26.9	0.39	0.39	0.39	0.6	0.6	0.6
	Size:	793	667	769	681	691	681	17	17	17	61	61	61
Right	Iters:	-	-	-	-	-	-	-	-	-	3	7	3
	Time:	-	-	-	-	-	-	-	-	-	8.3	20.3	8.2
	Size:	-	-	-	-	-	-	-	-	-	495	756	495

Table 2. Experimental results for the Zeroconf protocol. Time in seconds.

Zeroconf protocol (Interval domain)	$K = 4$ P1	$K = 6$ P1	$K = 8$ P1	$K = 4$ P2	$K = 6$ P2	$K = 8$ P2
Time (Mass heuristic):	6.2	16.8	32.2	5.8	18.5	50.6
Time (Depth heuristic):	2.6	6.0	6.6	2.6	6.7	8.1
Time (Mix):	2.6	6.3	6.8	2.6	6.9	8.4
Time PASS:	0.6	0.8	1.1	0.7	0.9	1.2

itself may not provide good results either, with depth-unrolling and the mixed heuristics performing substantially better.

5 Conclusions

We have shown that the approach of [16,23] for abstraction of probabilistic systems can be extended to arbitrary domains, allowing probabilistic checkers to profit from well developed libraries for abstract domains like intervals, octagons, and polyhedra [3,15].

For this we have extended the construction of abstract reachability trees presented in [11] to the probabilistic case. The extension no longer yields a tree, but a stochastic 2-Player game that overapproximates the MDP semantics of the program. The correctness proof requires to use a novel technique.

The new approach allows to refine abstractions using standard techniques like delaying widenings. We have also presented a technique that selectively delays widenings using a heuristics based on quantitative properties of the abstractions.

Acknowledgements. We thank Holger Hermanns, Ernst-Moritz Hahn, and Luis M.F. Fioriti for valuable comments, Björn Wachter for many discussions during the second author's stay at the University of Oxford, made possible by Joel Ouaknine, and five anonymous reviewers for helpful remarks. The second author is supported by the DFG Graduiertenkolleg 1480 (PUMA).

References

1. PRISM homepage: `http://www.prismmodelchecker.org/`
2. Bagnara, R., Dobson, K., Hill, P.M., Mundell, M., Zaffanella, E.: Grids: A domain for analyzing the distribution of numerical values. In: Puebla, G. (ed.) LOPSTR 2006. LNCS, vol. 4407, pp. 219–235. Springer, Heidelberg (2007)
3. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Science of Computer Programming 72(1-2), 3–21 (2008)
4. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker BLAST. Proc. of STTT 9(5-6), 505–525 (2007)
5. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. In: Mogensen, T.Æ., Schmidt, D.A., Sudborough, I.H. (eds.) The Essence of Computation. LNCS, vol. 2566, pp. 85–108. Springer, Heidelberg (2002)

6. Condon, A.: The complexity of stochastic games. Inf. Comput. 96(2), 203–224 (1992)
7. Condon, A.: On algorithms for simple stochastic games. DIMACS Series in Discr. Math. and Theor. Comp. Sci., vol. 13, pp. 51–73. AMS (1993)
8. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of POPL, pp. 238–252 (1977)
9. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: POPL, San Antonio, Texas, pp. 269–282. ACM Press, New York (1979)
10. Esparza, J., Gaiser, A.: Probabilistic abstractions with arbitrary domains. Technical report, Technische Universität München (2011), http://arxiv.org/abs/1106.1364
11. Gulavani, B.S., Chakraborty, S., Nori, A.V., Rajamani, S.K.: Automatically refining abstract interpretations. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 443–458. Springer, Heidelberg (2008)
12. Gulavani, B.S., Rajamani, S.K.: Counterexample driven refinement for abstract interpretation. In: Hermanns, H. (ed.) TACAS 2006. LNCS, vol. 3920, pp. 474–488. Springer, Heidelberg (2006)
13. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PASS: Abstraction refinement for infinite probabilistic models. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 353–357. Springer, Heidelberg (2010)
14. Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 162–175. Springer, Heidelberg (2008)
15. Jeannet, B., Miné, A.: APRON: A library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 661–667. Springer, Heidelberg (2009)
16. Kattenbelt, M., Kwiatkowska, M.Z., Norman, G., Parker, D.: Abstraction refinement for probabilistic software. In: Jones, N.D., Müller-Olm, M. (eds.) VMCAI 2009. LNCS, vol. 5403, pp. 182–197. Springer, Heidelberg (2009)
17. Kattenbelt, M., Kwiatkowska, M.Z., Norman, G., Parker, D.: A game-based abstraction-refinement framework for markov decision processes. Form. Methods Syst. Des. 36, 246–280 (2010)
18. Monniaux, D.: Abstract interpretation of probabilistic semantics. In: SAS 2000. LNCS, vol. 1824, pp. 322–340. Springer, Heidelberg (2000)
19. Monniaux, D.: Abstract interpretation of programs as markov decision processes. In: Proc. of SAS, pp. 237–254 (2003)
20. Di Pierro, A., Hankin, C., Wiklicky, H.: On probabilistic techniques for data flow analysis. Electr. Notes Theor. Comput. Sci. 190(3), 59–77 (2007)
21. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Interscience, Hoboken (1994)
22. Wachter, B.: Refined Probabilistic Abstraction. PhD thesis, Universität des Saarlandes (2011)
23. Wachter, B., Zhang, L.: Best probabilistic transformers. In: Barthe, G., Hermenegildo, M. (eds.) VMCAI 2010. LNCS, vol. 5944, pp. 362–379. Springer, Heidelberg (2010)