Probabilistic Abstractions with Arbitrary Domains

Andreas Gaiser and Javier Esparza

Fakultät für Informatik, Technische Universität München, Germany
{gaiser,esparza}@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. In these approaches, programs are abstracted into Markov Decision Processes (MDPs). Analysis of the MDPs allows to compute lower and upper bounds for the probability of reaching an error state. The bounds can be improved by refining the abstraction. The approaches of Hermanns et al. and Kattenbelt et al. 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 several 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 [5]. Abstract domains are used to compute safe overapproximations of the program behaviour.

Recent work has extended abstraction techniques to probabilistic systems [12, 8]. The key idea is to abstract the Markov chain defining the semantics of the program into a Markov Decision Process (MDP) (or stochastic games). The construction ensures that the probability of reaching a goal state in the Markov chain lies between the probabilities of reaching it in the MDP using an optimal resp. a pessimal strategy to resolve the nondeterminism.

The abstraction technique of [12, 8] 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. If the upper and lower bounds obtained using a set of predicates are not close enough, the abstraction can be refined by adding new predicates by means 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, like potentially expensive equality or 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 [7]. Widening operations are combined with interpolation methods, leading to more efficient abstraction algorithms.

In this paper we show that the ideas of Gulavani *et al.* can also be applied in the probabilistic area, resulting in an extension of the approaches of [12, 8] to arbitrary abstract domains. Given a probabilistic program, an abstract domain and a widening for this domain, we show how to construct a MDP, and how to refine the abstraction using standard techniques of abstract interpretation, like delaying widenings [3]. We also show how this can be done incrementally by "refining" special nodes in the MDP, which can be seen as an analogon to the refinement techniques in probabilistic CEGAR. Finally, we explain how to derive the predicate abstraction approach as a special case.

The rest of the paper is organized as follows. In the rest of the introduction we informally present the key ideas of our approach by means of some examples. Section 2 contains preliminaries. Section 3 formally introduces the abstraction technique, and proves that the probabilities of the optimal resp. pessimal startegies are an upper resp. a lower bound of the exact probability of reaching a state. In section 3.4 we deal with the special case of predicate abstraction. Section 5 discusses the interest of the generalization with the help of some experiments.

Related work. Besides the work of [9] and [12], Monniaux [13, 14] has studied how to abstract probability distributions over program states (instead of the states themselves). Wachter et al. discuss in [17] an abstract-interpretation-based framework for analyzing stochastic games, but still rely on predicate abstraction.

1.1 An example

Consider the C-like program of Fig. 1. The goal of the program is to receive 100 packages. When trying to receive a new package, with probability 0.01 the network connection can be interrupted, i.e. `receive_package()` will return false. Then no packages can be received anymore. The probability of receiving no package at all (equal to the probability of reaching location 6) is obviously 0.01 (only the first iteration of the loop is relevant). However, a brute force automatic technique will construct and analyze the standard semantics of the program, a Markov chain with over 400 states, part of which is shown in Fig. 1. We use this simple example to informally introduce our abstraction technique, which also "sees" that only the first iteration counts.

A (concrete) state of the program is a pair $\langle \ell, v \rangle$, where ℓ is the current program point and v the current value of `nrp`. We use the abstract interpretation framework [5], and choose the *integer interval domain*: an abstract state is a pair $\langle \ell, [a, b] \rangle$, where $[a, b]$ is an interval of values of `npr`. Starting with $\langle 1, [0, 0] \rangle$ we compute successor abstract states in the usual way using depth-first search (DFS): We compute all direct successors of an abstract state q using the standard recipe of abstract interpretation. If one of the successors d has already been generated before, we add a transition from q to d. Also, if the successors of q are the outcome of a probabilistic choice (like at line 2), we label the transitions with

```
   int nrp = 0;

1: while (nrp <= 100)
2:   if (receive_package()) then
3:     nrp = nrp+1
4:   else break
5: if (nrp < 1) then
6:   fail
7: end.
```

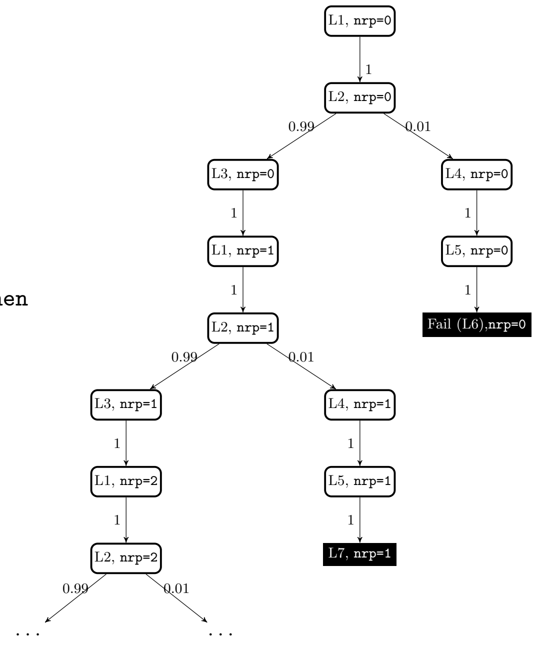

Fig. 1. Example program and corresponding transition system.

probabilities. To keep the number of explored abstract states small, we apply a *widening operator* ∇: If we generate an abstract state $\langle \ell, [a,b] \rangle$ such that ℓ is the head of a while loop (in our example the only head is $\ell = 1$), and $\langle \ell, [a,b] \rangle$ has an ancestor $\langle \ell, [a',b'] \rangle$ in the spanning tree generated by the DFS, then we overapproximate $\langle \ell, [a,b] \rangle$ by $\langle \ell, s \rangle$, with $s = [a',b'] \nabla [\min(a,a'), \max(b,b')]$. We use the classical widening operator defined in [5] for this example. To get more precision, the use of widenings can be delayed for the first k loop unfoldings. In our example we use $k = 1$, i.e. we apply widening after the second unrolling of the loop. Moreover, we have to take care of the fact that abstraction introduces *nondeterminism*, since an abstract state can represent several concrete states. In our case this happens at state $\langle 1, [1, \infty) \rangle$: The guard `x <= 100` is satisfied by $[1, 100]$, but not satisfied by $[101, \infty)$. In this case we add two successor states, labeled with "?".

The result of the construction is shown in Fig. 2. As in [12, 8], and because of nondeterminism, the result is in general a *Markov Decision Process*. Given a *strategy* to resolve the nondeterminism, we obtain a Markov chain, for which we can compute the probability of reaching the fail location. The first theorem of our paper is the counterpart of the result of [12, 8]: the probability of reaching the fail location in the real program lies between the probabilities of reaching it under the optimal and pessimal strategies for the MDP. Since these probabilities can be computed using well-known algorithms, we obtain a lower and an upper bound for the real probability. In our example we obtain a probability of failure of

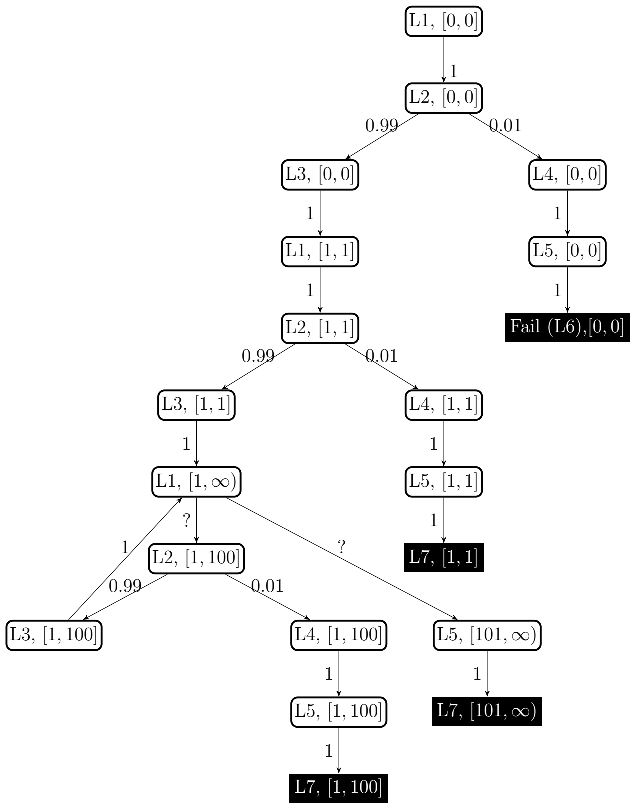

Fig. 2. Abstract transition system of the program in Fig. 1.

0.01, independently of the strategy at the only nondeterministic state $\langle 1, [1,\infty]\rangle$, and so we compute the exact value.

Consider now the program on the left of Fig. 3, a variant of the program above. Here `choice`(p) models a random number generator call, 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. However, when this program is analyzed with PASS [8, 9], a leading tool on probabilistic abstraction refinement on the basis of predicate abstraction, the while loop is (implicitly) unrolled 100 times because the tool fails to "catch" the invariant, independently of the options chosen to refine the abstraction. 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 [3], easily 'catches" the invariant (see Section 5 for a detailed analysis). The same happens for the program on the right of the figure, which exhibits a more interesting probabilistic behaviour: we obtain good upper and lower bounds for the probability of failure using the standard interval domain.

```
   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) fail
```

```
   int c = 0, i = 0;
1: while(i <= 100)
2:   if choice(0.99) then i = (i+1);
3:   c = c-i+2;
4: if (c >= i) then fail
```

Fig. 3. Example programs 2 and 3.

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 MDP-based abstraction approach of [12, 8] can be easily extended to arbitrary abstract domains, making it more flexible and efficient.

2 Preliminaries

2.1 Probabilistic systems

This section introduces Markov Decision Processes and Markov chains, and lists certain properties of those systems that we need later on. For a more thorough introduction into the subject and proofs for the theorems see e.g. [16, 11].

Definition 1. *A* Markov Decision Process (MDP) $\mathcal{M}$ *is a tuple* (A, P, T_A, T_P, q_0), *where*

- A *is a finite or countable set of* action states*,*
- P *is a finite or countable set of* probabilistic choice states*,*
- $q_0 \in A \cup P$ *is the* initial state*,*
- $T_A \subseteq A \times (A \cup P)$ *are the* action transitions *of* $\mathcal{M}$*,*
- $T_P \subseteq P \times (0, 1] \times (A \cup P)$ *are the* probabilistic transitions *of* $\mathcal{M}$*,*
- *for every state* $s \in A$ *there exists at least one* $s' \in (A \cup P)$ *such that* $(s, s') \in T_A$*,*
- *for every state* $s \in P$, $\sum_{(s,p,s') \in T_P} p = 1$*, and*
- *for every* $s \in P$ *and* $s' \in A \cup P$, $|\{(s, p, s') \mid p \in [0, 1]\} \cap T_P| \leq 1$.

A MDP $\mathcal{M} = (A, T_A, P, T_P, q_0)$ *is a* Markov chain (MC) *if for every* $s \in A$ *there exists exactly one* $s' \in A \cup P$ *such that* $(s, s') \in T_A$.

Note 1. Our definition of Markov Decision Process and Markov chains differ slightly from the usual ones used in the literature; especially we do not assign any action names since we do not need them further.

For the rest of the section we fix a MDP $\mathcal{M} = (A, P, T_A, T_P, q_0)$ and let $S = A \cup P$. We define the relation $\rightarrow \subseteq S \times S$ by setting $s \rightarrow s'$ iff $(s, s') \in T_A$ or if there is a $p \in (0, 1]$ such that $(s, p, s') \in T_P$. We define $Succs(s) := \{s' \in S \mid s \rightarrow s'\}$ for every $s \in S$. A path r of $\mathcal{M}$ is a finite sequence $s_0, s_1, \ldots s_n$

of states such that $s_i \to s_{i+1}$ for every $i \in \{0, \ldots, n-1\}$. We also write r as $s_0 \to s_1 \to \ldots \to s_n$. A path r is *initial* if $s_0 = q_0$.

The nondeterminism of action states is resolved with the help of strategies:

Definition 2. *A* strategy ϕ *maps every initial path* $r = s_0 \to s_1 \to \ldots \to s_n$ *with* $s_n \in A$ *to a distribution* $\phi(r) : Succs(s_n) \to [0,1]$. *A strategy* ϕ *is* non-randomized *if for every* r*, there exists an* $s \in S$ *such that* $\phi(r)(s) = 1$*, and then we write* $\phi(r) = s$ *as an abbreviation. A strategy* ϕ *is* memoryless *iff* ϕ*'s choice only depends on the last state of the path, i.e., if for all initial paths* $x_1, \ldots, x_k$ *and* $y_1, \ldots, y_l$ *with* $x_k = y_l$ *and* $x_k \in A$, $\phi(x_1 \to \ldots \to x_k) = \phi(y_1 \to \ldots \to y_l)$ *holds. A memoryless and non-randomized strategy can be seen as a map from* S *to* $S \cup A$. *A path* r *is* realizable *in* $\mathcal{M}$ *obeying strategy* ϕ *if* r *is an initial path of* $\mathcal{M}$ *and for every prefix* $s_0 \to \ldots \to s_n$ *of* r *with* $s_{n-1} \in A$, $\phi(s_0 \to \ldots \to s_{n-1})(s_n) > 0$ *holds. The* weight $w_\phi(r)$ *of a realizable path obeying strategy* ϕ *is the product of the probabilities of its edges. If* $\mathcal{M}$ *is a Markov chain, there is only one possible strategy* ϕ*. In this case we just write* w *instead of* w_ϕ.

Definition 3. *Let* $\mathrm{Cyl}(\mathcal{M}[\phi], \sigma)$ *denote the set of initial runs (or* cylinders*) realizable in* $\mathcal{M}$ *obeying* ϕ *which end in* σ *and have not visited* σ *before. (If* $\mathcal{M}$ *is a Markov chain, there is only one possible strategy and we write* $\mathrm{Cyl}(\mathcal{M}, \sigma)$*). The* probability of reaching a state σ *in* $\mathcal{M}$ *using* ϕ *is given by*

$$P_{\mathcal{M}[\phi]}(\sigma) = \sum_{r \in Cyl(\mathcal{M}[\phi], \sigma)} w_\phi(r).$$

If $\mathcal{M}$ *is a Markov chain, the probability is* $P_{\mathcal{M}}(\sigma) = \sum_{r \in Cyl(\mathcal{M}, \sigma)} w(r)$.

We define

$$P^+_{\mathcal{M}}(\sigma) := \sup_{\phi \in \phi_{\mathcal{M}}} P_{\mathcal{M}[\phi]}(\sigma) \text{ and } P^-_{\mathcal{M}}(\sigma) := \inf_{\phi \in \phi_{\mathcal{M}}} P_{\mathcal{M}[\phi]}(\sigma)$$

The following theorem is well-known [11]:

Theorem 1. *If* $\mathcal{M}$ *has only finitely many states, there exist memoryless, non-randomized strategies* $\phi^-, \phi^+ \in \mathcal{S}_{\mathcal{M}}$ *such that*

$$P^-_{\mathcal{M}}(\sigma) = P_{\mathcal{M}[\phi^-]}(\sigma) \text{ and } P^+_{\mathcal{M}}(\sigma) = P_{\mathcal{M}[\phi^+]}(\sigma). \tag{1}$$

Moreover, ϕ^- *and* ϕ^+ *are effectively computable (e.g. by value iteration).*

3 Abstractions of probabilistic programs

3.1 Probabilistic Programs

We study imperative programs equipped with an operator $\oplus$ that realizes *probabilistic choice*: The intended meaning of $P_1 \oplus_p P_2$ is that with probability p, P_1 will be executed, and with probability $1-p$, P_2 will be executed. Of course we can generalize $\oplus$ to allow more than 2 outcomes. For the sake of simplicity, we restrict ourselves to program variables with integer values, but this limitation is not necessary for the method itself.

Definition 4. *Let $\mathcal{V}$ be a finite set of variables. By $\Sigma_{\mathcal{V}} := \mathcal{V} \to \mathbb{Z}$ we denote the set of all configurations of $\mathcal{V}$. By* Trans($\mathcal{V}$) *we denote the class of all maps in $2^{\Sigma_{\mathcal{V}}} \to 2^{\Sigma_{\mathcal{V}}}$ that are monotone with respect to $\subseteq$.*

We represent the programs we investigate as *probabilistic control-flow graphs*. We use a notation similar to the one in [6] for non-probabilistic programs:

Definition 5. *A* probabilistic control flow graph *C, PCFG for short, is a tuple $(\mathcal{L}_A, \mathcal{L}_P, l_0, l_e, \mathcal{V}, \sigma_0, \mathcal{E}, \delta)$, where*

- *$\mathcal{L}_A$ and $\mathcal{L}_P$ (with $\mathcal{L} := \mathcal{L}_A \cup \mathcal{L}_P$) are finite and disjunct sets of program locations, l_0 is the* start location*, $l_e \in \mathcal{L}_A$ the* error location*, $\mathcal{V}$ is a finite set of integer variables, $\sigma_0 \in \Sigma_{\mathcal{V}}$ is the* start configuration *of C, $\mathcal{E} \subseteq \mathcal{L} \times \mathcal{L}$ is the set of* control flow edges*, and $\delta : \mathcal{E} \to ((0,1] \cup Trans(\mathcal{V}))$ is the* label function*,*

satisfying the following properties:

1. *$\delta(l,l') \in Trans(\mathcal{V})$ for all $(l,l') \in (\mathcal{L}_A \times \mathcal{L}) \cap \mathcal{E}$, and $\delta(l,l') \in (0,1]$ for all $(l,l') \in (\mathcal{L}_P \times \mathcal{L}) \cap \mathcal{E}$;*
2. *$\sum_{(l,l') \in (\{l\} \times \mathcal{L}) \cap \mathcal{E}} \delta(l,l') = 1$ for all $l \in \mathcal{L}_P$;*
3. *the graph $(\mathcal{L}, \mathcal{E})$ is connected and reducible (see e.g. [15]), and every node has at most two incoming edges.*

A PCFG is deterministic *if for all locations $l, l' \in \mathcal{L}_A$, and for every $\sigma, \sigma_1, \sigma_2 \in \Sigma_{\mathcal{V}}$ the following holds: If $\delta(l,l')(\sigma) = \sigma_1$ and $\delta(l,l')(\sigma) = \sigma_2$, then $\sigma_1 = \sigma_2$.*

For the following sections let $C = (\mathcal{L}_A, \mathcal{L}_P, l_0, l_e, \mathcal{V}, \sigma_0, \mathcal{E}, \delta)$ be a *deterministic* PCFG. The semantics of C is a Markov chain:

Definition 6. *Let $q_e \notin \mathcal{L} \times \Sigma_{\mathcal{V}}$ be a fresh state. The Markov chain $\mathcal{M}_C = (A, P, \Lambda_A, \Lambda_P, q_0)$ associated to C is defined as follows:*

- $A = (\mathcal{L}_A \times \Sigma_{\mathcal{V}}) \cup \{q_e\}$ *and* $P = \mathcal{L}_P \times \Sigma_{\mathcal{V}}$
- $\Lambda_A = \{(\langle l, \sigma \rangle, \langle l', \sigma' \rangle) \mid (l,l') \in \mathcal{E} \wedge l \in \mathcal{L}_A \wedge \sigma' \in \delta(l,l')(\{\sigma\})\}$
  $\cup (\{l_e\} \times \Sigma_{\mathcal{V}}) \times \{q_e\}$
  $\cup \{(q_e, q_e)\}$
  $\cup \{(\langle l, \sigma \rangle, \langle l, \sigma \rangle) \mid (l,l') \notin \mathcal{E}$ *for all* $l' \in \mathcal{E}\}$
- $\Lambda_P = \{\langle l, \sigma \rangle, p, \langle l', \sigma \rangle) \mid (l,l') \in (\mathcal{E} \cap (\mathcal{L}_P \times \mathcal{L})) \wedge p = \delta(l,l')\}$
- $q_0 = \langle l_0, \sigma_0 \rangle$

The *reachability problem* for the PCFG C is the problem of computing the probability of reaching q_e in $\mathcal{M}_C$.

3.2 Abstract Interpretation

We make use of the Abstract Interpretation framework (see [5]). For abstracting subsets of $\Sigma_{\mathcal{V}}$ we use abstract domains in form of complete lattices $(A^\sharp, \sqsubseteq, \top, \bot, \sqcup, \sqcap)$. For every abstract domain we assume the existence of a monotone abstraction map $\alpha : 2^{\Sigma_{\mathcal{V}}} \to A^\sharp$ and a monotone concretization map $\gamma : A^\sharp \to 2^{\Sigma_{\mathcal{V}}}$, satisfying $\alpha(\gamma(a)) \sqsubseteq a$ for all $a \in A^\sharp$. We also use *widenings*:

Definition 7. *Let* $(A^\sharp, \sqsubseteq, \top, \bot, \sqcup, \sqcap)$ *be an abstract domain. A* widen operator $\nabla : A^\sharp \times A^\sharp \to A^\sharp$ *satisfies the following conditions:*

1. *For all* $a, b \in A^\sharp$*,* $a \nabla b \sqsupseteq a$ *and* $a \nabla b \sqsupseteq b$ *holds,*
2. *For every strictly increasing sequence* $a_0 \sqsubset a_1 \sqsubset \ldots$ *in* $A^\sharp$ *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.*

3.3 Constructing MDPs using Abstract Interpretation

The size of $\mathcal{M}_C$ (which even might have infinitely many states in general) causes us to investigate abstraction techniques for detecting meaningful properties of the corresponding PCFG C. We are especially interested in reachability problems, i.e. the probability of reaching q_e in C. The input to the problem is the deterministic PCFG C, together with a distinguished set of *back edges.* These are the back edges obtained by an arbitrary depth-first search traversal of the graph $(\mathcal{L}, \mathcal{E})$ starting at l_0 (recall that an edge (s, s') of a graph is a back edge of a depth-first traversal if s' is an ancestor of s in the depth first search tree [4]).

We fix an abstract domain $(A^\sharp, \sqsubseteq, \top, \bot, \sqcup, \sqcap)$ and a widening $\nabla : A^\sharp \times A^\sharp \to A^\sharp$. We abstract $\mathcal{M}_C$ into a MDP $\mathcal{A}$ that shares certain properties with $\mathcal{M}_C$. The method is heavily based on approaches like the ones in [2, 6] for constructing abstract reachability trees. States of $\mathcal{A}$ are tuples consisting of a program location and an abstract element $s \in A^\sharp$.

Algorithm 1 constructs the initial state $\langle l_0, \alpha(\{s_0\})\rangle$ and generates transitions and successor states in a depth-first search fashion by calling the procedure `dfs`.

The procedure constructs the MDP guided by the semantics of C. Hereby it replaces the maps $\delta(l, l') : 2^{\Sigma_\mathcal{V}} \to 2^{\Sigma_\mathcal{V}}$ from $\mathrm{Trans}(\mathcal{V})$ by abstract counterparts $\delta(l, l')^\sharp : A^\sharp \to 2^{A^\sharp}$ satisfying

$$\delta(l, l')(\gamma(a)) \subseteq \bigcup_{b \in \delta(l, l')^\sharp(a)} \gamma(b) \text{ for all } a \in A^\sharp. \tag{2}$$

Note that we allow for $\delta(l, l')^\sharp$ to return more than one element from $A^\sharp$; this can help increasing the accuracy of the constructed MDP. Hence we implicitly make use of abstract powerset domains (see [6]).

Probabilistic choices in C are translated into nodes in Θ_P, non-probabilistic nodes are translated into nodes in Θ_A. During the construction, we store for every state q the state $\mathrm{pred}(q)$ whose call to `dfs` causes the construction of q. Hence pred gives us the direct predecessor relation of a depth first traversal of $\mathcal{A}$, and hence a depth-first tree, with $\langle l_0, \alpha(\{s_0\})\rangle$ as its root. We will refer to this tree as the *dfs-tree* of $\mathcal{A}$ from now on.

Note that we call `dfs` with two arguments: an abstract state and its predecessor. We use the predecessor and the back edges of C to apply widenings: Suppose we are exploring a state $\langle l, s\rangle$ and $(l, l') \in \mathcal{E}$ is a back edge in the PCFG C. Let us assume we compute a potential successor $\langle l', s'\rangle$ of $\langle l, s\rangle$ with $s' \in \delta(l, l')^\sharp(s)$ and then call `dfs`$(\langle l', s'\rangle, \langle l, s\rangle)$. We compute the element $\hat{s} \in A^\sharp$ such that $\langle l', \hat{s}\rangle$

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

```
    Input: PCFG $C = (\mathcal{L}_A, \mathcal{L}_P, l_0, l_e, \mathcal{V}, \sigma_0, \mathcal{E}, \delta)$, abstract domain
           $(A^\sharp, \sqsubseteq, \top, \bot, \sqcup, \sqcap)$, widening $\nabla : A^\sharp \times A^\sharp \to A^\sharp$.
    Output: MDP $\mathcal{A} = (A, \Theta_A, P, \Theta_P, \langle l_0, s_0 \rangle)$.

    $A \leftarrow \{q_e^\sharp\}; P \leftarrow \emptyset; s_0 \leftarrow \alpha(\{\sigma_0\})$
    $\Theta_A \leftarrow \{(q_e^\sharp, q_e^\sharp)\}; \Theta_P \leftarrow \emptyset$
    pred$(\langle l_0, s_0 \rangle) \leftarrow \langle l_0, s_0 \rangle; d(\langle l_0, s_0 \rangle) \leftarrow 0$
    dfs$(\langle l_0, s_0 \rangle, \langle l_0, s_0 \rangle)$
    return $(A, \Theta_A, P, \Theta_P, \langle l_0, s_0 \rangle)$

    Proc dfs($\langle l, s \rangle$,$\langle l_p, s_p \rangle$)
    $x \leftarrow s$
1   if $(l_p, l)$ is a back edge then
        Compute $\hat{s}$ such that $\langle l, \hat{s} \rangle$ is the nearest predecessor of $\langle l_p, s_p \rangle$ with location
        $l$ in the dfs-tree.
        $x \leftarrow \hat{s} \nabla (s \sqcup \hat{s})$
2   if $\langle l, x \rangle \in A \cup P$ then return $\langle l, x \rangle$
3   $d(\langle l, x \rangle) \leftarrow 0$
    pred$(\langle l, x \rangle) \leftarrow \langle l_p, s_p \rangle$
    if $l \in \mathcal{L}_P$ then                          /* $l$ denotes a probabilistic choice */
        $P \leftarrow P \cup \{\langle l, x \rangle\}$
        forall the $l' : (l, l') \in \mathcal{E}$ do $\Theta_P \leftarrow \Theta_P \cup \{(\langle l, x \rangle, p, \text{dfs}(\langle l', x \rangle, \langle l, x \rangle))\}$
    else if $l = l_e$ then $\Theta_A \leftarrow \Theta_A \cup \{(\langle l, x \rangle, q_e^\sharp)\}$
    else                                                  /* $l$ is an action label */
        $A \leftarrow A \cup \{\langle l, x \rangle\}$
        forall the $l' : (l, l') \in \mathcal{E}$ do
            forall the $s' \in \delta(l, l')^\sharp(x)$ do $\Theta_A \leftarrow \Theta_A \cup \{(\langle l, x \rangle, \text{dfs}(\langle l', s' \rangle, \langle l, x \rangle))\}$
    return $\langle l, x \rangle$
```

is the nearest predecessor of $\langle l, s \rangle$ with location l' in the dfs-tree and use it for the widening operation in line 1 of the procedure `dfs`.

This is a rather simple strategy for ensuring termination. More sophisticated widening strategies can be found in [6].

We ignore the use of the $d(q)$-values (line 3) for now; they are necessary for refining the abstractions as described in section 4.

The following theorem is proved in the appendix.

Theorem 2.

1. *Algorithm 1 terminates.*
2. *$\mathcal{A} = (A, \Theta_A, P, \Theta_P, \langle l_0, \alpha(\{\sigma_0\}) \rangle)$ has the following properties:*
   (a) *If $\langle l, s \rangle \in A$ and $\sigma \in \gamma(s)$, $(l, l') \in \mathcal{E}$ and $\sigma' \in \delta(l, l')(\{\sigma\})$, then there exists at least one transition $(\langle l, s \rangle, \langle l', s' \rangle) \in \Theta_A$ with $\sigma' \in \gamma(s')$.*
   (b) *If $(l, s) \in P$ and $(l, l') \in \mathcal{E}$, $p = \delta(l, l')$, then there exists exactly one transition of the form $(\langle l, s \rangle, p, \langle l', s' \rangle) \in \Theta_A$, and $s' = s$.*

We can show that the probabilities $P^-_{\mathcal{A}}(q^\sharp_e)$ and $P^+_{\mathcal{A}}(q^\sharp_e)$ of reaching the abstract state $q^\sharp_e$ in the abstract MDP $\mathcal{A}$, are a lower and an upper bound for the probability $P_{\mathcal{M}_C}(q_e)$ of reaching the error state q_e in the Markov chain $\mathcal{M}_C$.

Theorem 3. $P_{\mathcal{M}_C}(q_e) \in [P^-_{\mathcal{A}}(q^\sharp_e), P^+_{\mathcal{A}}(q^\sharp_e)]$.

Using this theorem, and since $\mathcal{A}$ has only finitely many states, we can compute lower and upper bounds for the reachability probability $P_{\mathcal{M}_C}(q_e)$ by computing extremal strategies for $\mathcal{A}$ using e.g. value iteration (see Theorem 1).

3.4 Predicate abstraction as a special case

Predicate abstraction can be embedded in our setting. Let $P = \{P_1, \ldots, P_n\}$ be a set of predicates over $\mathcal{V}$. We use the domain $(A^\sharp = 2^B, \subseteq, B, \emptyset, \cup, \cap)$ with $B = \{0,1\}^n$. For an element $b \in B$, we denote by b_i the i-th component of b. For $b \in B$ and $\sigma \in \Sigma_{\mathcal{V}}$, we say that σ satisfies b if $P_i(\sigma) = b_i$ for all $1 \le i \le n$. We define $\gamma : A^\sharp \to 2^{\Sigma_{\mathcal{V}}}$ by setting

$$\gamma(M) = \{\sigma \in \Sigma_{\mathcal{V}} \mid \exists b \in M : \sigma \text{ satisfies } b\}$$

for all $M \in A^\sharp$ and $\alpha : 2^{\Sigma_{\mathcal{V}}} \to A^\sharp$ by setting

$$\alpha(N) = \{b \in B \mid \exists \sigma \in N : \sigma \text{ satisfies } b\}$$

for all $N \subseteq \Sigma_{\mathcal{V}}$. We construct $\mathcal{A}$ using this domain; hereby only elements $M \in A^\sharp$ with $|M| = 1$ are used for the abstract state space. For constructing transitions in $\mathcal{A}$, we define for $l \in \mathcal{L}_A$ and $a \in B$:

$$\delta(l,l')^\sharp(\{a\}) = \{\{b\} \mid b \in B \wedge \exists \sigma \in \Sigma_{\mathcal{V}} : \sigma \in \delta(l,l')(\gamma(\{a\})) \wedge \sigma \text{ satisfies } b\}$$

and for $l \in \mathcal{L}_P$ we set $\delta(l,l')^\sharp(\{b\}) = p,$ if $\delta(l,l') = p$.

Since $A^\sharp$ is a finite domain, we do not need a widening for ensuring termination of the algorithm in Fig. 1, hence we can skip the if-statement in line 1 of the `dfs` procedure. $\mathcal{A}$ is essentially the quotient automaton of the predicate abstraction approaches (see e.g. [9]).

4 A Method for Refining $\mathcal{A}$

In this section we present techniques that can help refine the results of our method when the abstraction $\mathcal{A}$ is considered too coarse. We start with some definitions:

Definition 8. *We denote by ϕ^-, ϕ^+ for $\mathcal{A}$ two fixed memoryless strategies satisfying*

$$P^-_{\mathcal{M}}(\sigma) = P_{\mathcal{M}[\phi^-]}(\sigma) \text{ and } P^+_{\mathcal{M}}(\sigma) = P_{\mathcal{M}[\phi^+]}(\sigma).$$

Let $q \in A \cup P$. We denote by $\mathrm{val}^-(q)$ *resp.* $\mathrm{val}^+(q)$ *the probability of reaching $q^\sharp_e$ starting at q and obeying strategy ϕ^- resp. ϕ^+. For $q \in A$ we denote by* $\mathrm{choice}^-(q)$ *resp.* $\mathrm{choice}^+(q)$ *the states $\phi^-(q)$ and $\phi^+(q)$.*

Notice that $\phi^-(q)$ and $\phi^+(q)$ can be computed using e.g. value iteration.

4.1 Refinement using selective delay of widening

As in [6, 7], we assume that for every $(l, l') \in \mathcal{E}$, $\delta(l, l')^\sharp$ is *exact*, i.e.

$$\delta(l, l')(\gamma(a)) = \bigcup_{b \in \delta(l,l')^\sharp(a)} \gamma(b) \text{ for all } a \in A^\sharp,$$

and that we are also able to compute exact preimages of $\delta(l, l')$, i.e., we can compute the value $\delta^{-1}(l, l')^\sharp(a) \in 2^{A^\sharp}$ with

$$\delta(l, l')^{-1}(\gamma(a)) = \bigcup_{b \in \delta^{-1}(l,l')^\sharp(a)} \gamma(b) \text{ for all } a \in A^\sharp.$$

We extend $\delta^{-1}(l, l')^\sharp(a)$ to be defined on subsets of $\mathcal{A}$. We also assume that $\sqcap$ is exact. Hence the only source of imprecision is the use of widenings.

We already discussed a simple technique for *delaying widenings*: We avoid widenings during the start of the construction of $\mathcal{A}$. This can be done by skipping line 1 of the `dfs`-procedure for an initial prefix of the dfs-tree. If the Markov chain of the program is finite, by delaying widenings further and further we eventually reach the point in which no widening is applied at all. So this refinment technique is complete, but not efficient. Another applicable technique is widening with threshold values (see [3]), which is also trivially complete, but suffers from the same inefficiency problem. In the following we present a more sophisticated technique involving a more "selective" delaying of widenings. It is heavily based on the work of Gulavani et al. in [6, 7]. We modify parts of the refinement techniques presented there for the probabilistic case.

We prolong the use of widenings in the (re)construction of $\mathcal{A}$ for refining a suitable state q in $\mathcal{A}$. For this we modify the $d(s)$-values we assigned to each state s. We make use of the dfs-tree structure induced by the construction of $\mathcal{A}$, which is represented by the *pred*-function. For states q, q' with q is a predecessor of q' in the spanning tree, we define $\Delta(q, q')$ as the number of edges $\langle l_i, s_i\rangle \to \langle l_{i+1}, s_{i+1}\rangle$ on the tree path from q to q' for which (l_i, l_{i+1}) is a back edge in C.

4.2 Finding a candidate state for refining

Definition 9. *An abstract state $q = \langle l, s\rangle$ is* refinable *if $(val^+(q) - val^-(q)) > 0$ and $choice^+(q) \neq choice^-(q)$.*

I.e., q is refinable if the probabilities of reaching the error state from q using the optimal and pessimal strategies are different (otherwise we have already computed the exact value), and moreover the strategies choose different successors of q. In particular, if q is refinable then $l \in \mathcal{L}_\mathcal{A}$ holds. Given a refinable state q, let choice$^+(q) = \langle l^+, s^+\rangle$ and choice$^-(q) = \langle l^-, s^-\rangle$, with $s^+ \in \delta(l, l^+)^\sharp(s)$ and $s^- \in \delta(l, l^-)^\sharp(s)$. The choice at q is possible because there exist $\sigma \in \gamma(s)$ and $\sigma' \in \gamma(s)$ such that $\delta(l, l^-)(\{\sigma\}) \neq \emptyset$ and $\delta(l, l^+)(\{\sigma'\}) \neq \emptyset$. The procedure `ComputeRefineState()` computes a refinable state q, and the state q' in the dfs-tree of $\mathcal{A}$ that causes σ and σ' to be both contained in $\gamma(s)$. We call q' the

Procedure ComputeRefineState

```
**Output**: Returns a refinable state and the start state of its refinement.
Choose $q = \langle l, s\rangle$ in $\mathcal{A}$ such that
    $(\mathrm{val}^+(q) - \mathrm{val}^-(q)) > 0$ and $\mathrm{choice}^+(q) \neq \mathrm{choice}^-(q)$.

$\langle l^+, s^+\rangle \leftarrow \mathrm{choice}^+(q)\,;\ \langle l^-, s^-\rangle \leftarrow \mathrm{choice}^-(q)$
$\mathrm{pre}^+(0) \leftarrow \delta^{-1}(l, l^+)^\sharp(s^+)\,;\ \mathrm{pre}^-(0) \leftarrow \delta^{-1}(l, l^-)^\sharp(s^-)$
$i \leftarrow 0\,;\ \langle l'_0, s'_0\rangle \leftarrow q$
while $pre^+(i) \sqcap s_i \neq \bot \wedge pre^-(i) \sqcap s_i \neq \bot$ do
    $i \leftarrow i + 1\,;\ \langle l'_i, s'_i\rangle \leftarrow \mathrm{pred}(\langle l'_{i-1}, s'_{i-1}\rangle)$
    $\mathrm{pre}^+(i) \leftarrow \delta^{-1}(l'_i, l'_{i-1})^\sharp(\mathrm{pre}^+(i-1))$
    $\mathrm{pre}^-(i) \leftarrow \delta^{-1}(l'_i, l'_{i-1})^\sharp(\mathrm{pre}^-(i-1))$
return $(q, q' = \langle l_i, s_i\rangle)$
```

companion of q. Starting with $\mathrm{pre}_0^+ = s^+$ resp. $\mathrm{pre}_0^- = s^-$, the algorithm computes exact preimages following the path in the dfs-tree from $\langle l_0, s_0\rangle$ to q. This is similar to checking a non-probabilistic reachability counterexample for spuriousness. However, in our case eventually $\mathrm{pre}^+(i) \sqcap s_i \neq \bot$ or $\mathrm{pre}^-(i) \sqcap s_i \neq \bot$ for a $\langle l_i, s_i\rangle$ on the path to the root has to hold. Hence the cause of the different strategy choices possible at q can be found in $q' = \langle l_i, s_i\rangle$. Since widening is the only source of imprecision, we conclude that l_i is the tail of a back edge.

When there are several refinable states, we use a speculative heuristic to select one of them. The heuristic may cause the refinement process loop forever, but is hopefully more efficient in practice. We choose a refinable state q such that the product of $(\mathrm{val}^+(q) - \mathrm{val}^-(q))$ and the weight of the tree path from q_0 to q is maximal. The intuition is that the larger this product, the larger the impact of the refinement in the values of the lower and upper bounds.

4.3 Refinement of $\mathcal{A}$

The recursive procedure `UpdateDelay`() computes the refined successors after computing a refinable state q and its companion q'. For every state q in $\mathcal{A}$, a value $d(q)$, which we call the widen delay, is stored. If not set, then $d(q) = \bot$. During the initial construction of $\mathcal{A}$, $d(q)$ is set to 0 for all states in q. The procedure call updateDelay(q, n) delays the application of widenings on all paths starting at state $q = \langle l, s\rangle$: This implies that every path r in $\mathcal{A}$ starting from $q = \langle l, s\rangle$ and passing at most n backedges is also "realizable“, i.e., there exists a concrete state $\sigma \in \gamma(s)$ and a path in $\mathcal{M}_C$ starting with $\langle l, \sigma\rangle$ having the same location sequence as r. The function call $bedge(\langle l_1, s_1\rangle, \langle l_2, s_2\rangle)$ returns 1 if (l_1, l_2) is a back edge in C, 0 otherwise. For a complete refinement step, we compute the refinable q and its companion q' by `ComputeRefineState()`, then we call `UpdateDelay`$(q', \Delta(q, q'))$.

Procedure UpdateDelay(q,n)

```
Input: $q$, $n \in \mathbb{N}$
if $q = q_\epsilon^\sharp$ then return
if $d(q) = \bot$ then
    $d(q) \leftarrow n$
    if $n = 0$ then dfs$(q, \mathrm{pred}(q))$ else
        Create successor transitions of $q$ and add them to $\Theta_A$ resp. $\Theta_P$
        forall the $q' \in \mathit{Succs}(q)$ do updateDelay$(q', n - \mathit{bedge}(q, q'))$
else if $n = 0 \vee d(q) \geq n$ then return
else if $d(q) > 0$ then
    $d(q) \leftarrow n$
    forall the $q' \in \mathit{Succs}(q)$ do updateDelay$(q', n - \mathit{bedge}(q, q'))$
else if $d(q) = 0$ then
    $d(q) \leftarrow n$
    $B \leftarrow \mathrm{Succs}(q)$
    $\Theta_A \leftarrow \Theta_A \setminus \{q\} \times (A \cup S); \Theta_P \leftarrow \Theta_P \setminus \{q\} \times [0,1] \times (A \cup P)$
    Create successor transitions of $q$ and add them to $\Theta_A$ resp. $\Theta_P$
    For all $q'$ in $B \setminus \mathrm{Succs}(q)$ with pred$(q') = q$, set pred$(q') \leftarrow \hat{q}$ for a $\hat{q} \to q'$. If
    no such $\hat{q}$ exists, delete $q'$ from $A \cup P$.
    forall the $q' \in \mathit{Succs}(q)$ do updateDelay$(q', n - \mathit{bedge}(q, q'))$
```

5 Experiments

We have implemented a preliminary prototype of our approach in Ocaml. It uses the Apron library [10], which provides numerical abstract domains like intervals and polyhedra. We report on some small experiments where the interval domain beats or can compete with predicate abstraction. Notice we *do not claim* interval analysis to be superior to predicate abstraction in general: it is easy to exhibit examples where predicate abstraction beats interval analysis. The experiments only illustrate that intervals outperform predicate abstraction in harmless-looking examples, and so it is important to develop approaches that accomodate both, as well as other abstract domains.

5.1 Examples of Fig.3

We examine the programs from Fig.3. The probability of reaching the fail state in Program 2 is 0.5: an invariant like $c \leq k$ for some $k \leq 100$, together with the postcondition $i > 100$ of the loop, negates the guard of the statement at line 5, and so shows that after executing the loop the program cannot fail. However, PASS fails to generate predicates of this form (we tested all the available refinement strategies). It mostly generates predicates $\mathtt{c} \leq \alpha \cdot \mathtt{i} + \beta$ for some $\alpha > 0, \beta < 0$. Manually adding a predicate (an option of PASS) like $i > 3$, makes the tool compute the solution after only 3 refinements. Without manual help the tool does not terminate after 10 minutes. If we change line 2 to `while` `i <= n` for $n = 25, 50, 75, 100$, we observe that the runtime of PASS seems to increase with n (see Fig. 4). We applied our method without delaying widenings. The implementation terminates with the correct answer in less than a second, independently of the value of n (see again Fig. 4).

Prog	n	Interv.		PASS	
		$\#R$	Time	$\#R$	Time
P2	25	1	0.06	6	0.74
P2	50	1	0.06	7	19.72
P2	75	1	0.06	7	>600.00
P2	100	2	0.06	>7	>600.00
P3	-	4	0.31	52	>600.00

Model	Quest.	del=10	del=15	PASS
game	(1)	0.07	0.10	0.12
game	(2)	0.06	0.06	>600
game	(3)	0.06	0.06	41.18
∞-game	(1)	65.33	21.46	25.00
∞-game	(2)	64.39	21.51	>600.00
∞-game	(3)	66.97	21.58	>600.00
zeroconf(4)	(1)	0.24	0.42	0.51
zeroconf(4)	(2)	0.23	0.43	0.48
zeroconf(6)	(1)	0.34	0.67	0.65
zeroconf(6)	(2)	0.52	0.68	0.60
zeroconf(8)	(1)	0.43	0.99	0.78
zeroconf(8)	(2)	0.42	0.98	0.66

Fig. 4. Left table: results for Programs 2 and 3. $\#R$ = number of refinements. Right table: results for games and zeroconf on several questions; del = widening delay. All times in seconds.

In Program 3 a probabilistic choice is situated inside a loop, and so the program exhibits behaviour more specific to probabilistic programs. Both variants of our approach compute an upper bound of 0.001 for the probabilty of fail in under one second. PASS exhibits similar problems as in Program 2 (again, we tested all refinement strategies and interpolation engines available), and is not able to terminate within 10 minutes.

5.2 Further examples

We report on three further small experiments.

game: The program on the left of Fig. 5.2 repeatedly toss a coin, and depending on the outcome increase the earnings of one of two players, `l` and `r`. The game stops when the sum of the players' earnings is greater than 100. The programs are similar to protocol fragments, where e.g. packages are sent through different channels following some stochastic policy. We compute answers to the following questions with accuracy 0.01: (1) What is the probability that player one has more money than player two at the end of the game (0.5, since the players are symmetrical)? (2) What is the probability that player 1 has twice as much money as player 2 at the end? (3) How probable is it that player one has 50 money units more than player two?

∞-*game*: The program on the right of Fig. 5.2 augments the *game* example with an additional rule: With probability 0.25, we remove a money unit from player two and give it to player two (or vice versa). This causes the state space of ∞-*game* to be infinite. We examine the same questions as in *game*.

zeroconf: This is a simple probabilistic model of the Zeroconf protocol, adapted from [1, 12], where it was analyzed using PRISM and predicate abstraction. It is a good example for the predicate abstraction technique. The example is parameterized by K, the maximal number of probes sent by the protocol implementation. We tested it for $K = 4, 6, 8$ and two different properties.

```
int l=1, r=1;
while (l+r <= 100)
  if choice(0.5) then
     l = 2*l+r+1
  else r = 2*r+l+1;
if (*) then fail
```

```
int l=1, r=1;
while (l+r <= 100)
if choice(0.75) then
  if choice(0.50) then
     l = 2*l+r+1
  else r = 2*r+l+1;
else
  if choice(0.50) then
     l = l-1; r = r+1;
  if choice(0.50) then
     l = l+1; r = r-1;
if (*) then fail
```

Fig. 5. Examples *game* and ∞-game. The three conditions tested at `(*)` were `(l>r)`, `(l>2*r)` and `(l-r>50)`.

We use the delayed widening technique explained in the introduction, with values 10 and 15 (times are similar for both, indicating that for these examples the technique is rather insensitive to the choice of delay). Since the tool described in [12] is to the best of our knowledge not publicly available, we adapted all examples for PASS and again used it for comparisons. We tested both interpolation engines PASS offers.

We point out several interesting aspects. In the *game* example the efficiency of PASS depends highly on the question asked, whereas our approach exhibits always the same behaviour. In ∞-*game* the number of states of our MDPs is quite large (e.g. 3222 states for the first question and a delayed widening of 10), partly caused by multiple refinement steps necessary to improve the result. Again the running time of PASS depends on the asked question. In the *Zeroconf*-protocol PASS behaves very well and generates smaller MDPs than ours, but due to the cheaper successor generation for the interval domain we can still compete.

6 Conclusions

We have shown that the approach of [9, 12] for abstraction of probabilistic systems can be extended to arbitrary domains. For this we have extended the construction of abstract reachability trees presented in [6] to the probabilistic case. The extension no longer yields a tree, but a Markov Decision process that over-approximates the Markov chain semantics of the program.

The new approach allows to refine abstractions using standard techniques like delaying widenings and using widenings with thresholds. We have also presented a more sophisticated technique that selectively delays widenings.

Our work allows probabilistic checkers to profit from well developed libraries for abstract domains like intervals, octagons, and polyhedra [10]. Future work will investigate how to integrate them in existing tools.

References

1. Obtained from http://www.prismmodelchecker.org/files/vmcai09/.
2. Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The software model checker blast. *STTT*, 9(5-6):505–525, 2007.
3. Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. In *The Essence of Computation*, pages 85–108, 2002.
4. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill Book Company, 2001.
5. Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.
6. Bhargav S. Gulavani, Supratik Chakraborty, Aditya V. Nori, and Sriram K. Rajamani. Automatically refining abstract interpretations. In *TACAS*, pages 443–458, 2008.
7. Bhargav S. Gulavani and Sriram K. Rajamani. Counterexample driven refinement for abstract interpretation. In *TACAS*, pages 474–488, 2006.
8. Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. Pass: Abstraction refinement for infinite probabilistic models. In *TACAS*, pages 353–357, 2010.
9. Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic cegar. In *CAV*, pages 162–175, 2008.
10. Bertrand Jeannet and Antoine Miné. Apron: A library of numerical abstract domains for static analysis. In *CAV*, pages 661–667, 2009.
11. M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. A game-based abstraction-refinement framework for Markov decision processes. Technical Report RR-08-06, Oxford University Computing Laboratory, February 2008.
12. Mark Kattenbelt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Abstraction refinement for probabilistic software. In *VMCAI*, pages 182–197, 2009.
13. David Monniaux. Abstract interpretation of probabilistic semantics. In *SAS*, pages 322–339, 2000.
14. David Monniaux. Abstract interpretation of programs as markov decision processes. In *SAS*, pages 237–254, 2003.
15. Steven S. Muchnick. *Advanced Compiler Design and Implementation.* Morgan Kaufmann, 1997.
16. Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley-Interscience, April 1994.
17. Björn Wachter and Lijun Zhang. Best probabilistic transformers. In *VMCAI*, pages 362–379, 2010.

7 Appendix

7.1 Proof of Theorem 2

Theorem 4.

1. *Algorithm 1 terminates.*
2. *$\mathcal{A} = (A, \Theta_A, P, \Theta_P, \langle l_0, \alpha(\{\sigma_0\})\rangle)$ has the following properties:*
   (a) *If $\langle l, s\rangle \in A$ and $\sigma \in \gamma(s)$, $(l, l') \in \mathcal{E}$ and $\sigma' \in \delta(l, l')(\{\sigma\})$, then there exists at least one transition $(\langle l, s\rangle, \langle l', s'\rangle) \in \Theta_A$ with $\sigma' \in \gamma(s')$.*
   (b) *If $(l, s) \in P$ and $(l, l') \in \mathcal{E}$, $p = \delta(l, l')$, then there exists exactly one transition of the form $(\langle l, s\rangle, p, \langle l', s'\rangle) \in \Theta_A$, and $s' = s$.*

Proof. 1. Assume for the sake of contradiction that Alg. 1 does not terminate. Then $\mathcal{A}$ has infinitely many states. Since every node in the dfs tree has only finitely many sucessors, it must contain a branch with infinite depth, corresponding to a call sequence dfs$(\langle l_0, s_0\rangle, \langle l_0, s_0\rangle)$, dfs$(\langle l_1, s_1\rangle, \langle l_0, s_0\rangle), \ldots$. Let $(\beta_i)_{i\in\mathbb{N}} = \langle l_0, s_0\rangle, \langle l_1, s_1\rangle, \ldots$ with $\beta_i \neq \beta_j$ for $i \neq j$ be the corresponding sequence of generated states. Note that $\beta_i \to \beta_{i+1}$ in $\mathcal{A}$ for all $i \leq 0$. Due to the definition of backedges, there has to be at least one back edge (l, l') in C such that there are infinitely many $s, s' \in A^\sharp$ with $\langle l, s\rangle \to \langle l', s'\rangle$ occuring in β. We fix the infinite subsequence $\beta' = \langle l', s'_0\rangle, \langle l', s'_1\rangle, \ldots$ of β containing only states with program locations l'. Then

$$s'_i \sqsubset s'_i \nabla (s'_i \sqcup x_i) = s'_{i+1} \text{ for } i \leq 0 \text{ and elements } x_i \in A^\sharp$$

Hence we can conclude $s'_0 \sqsubset s'_1 \sqcup x_1 \sqsubset s'_2 \sqcup x_2 \ldots$. We define $a_0 := s'_0$ and $a_i := s'_i \sqcup x_i$ for $i > 0$. Now with an inductive argument we see that

$$s'_{i+1} = s'_i \nabla (s'_i \sqcup x_i) = s'_i \nabla a_i.$$

From this and Def. 7 we conclude that there has to be a k such that $s'_k = s'_{k+1}$. But then two elements in β are equal, which is a contradiction.

2. Follows immediately from the definition of the algorithm.

7.2 Proof of Theorem 3

We start with a simple lemma.

Lemma 1.

1. *An initial path $\langle l_0, \sigma_0\rangle \to \ldots \to \langle l_k, \sigma_k\rangle \to q_e$ in $\mathcal{M}_C$ is completely determined by its location sequence $l_0 \ldots l_k$, i.e., there is no other initial path $\langle l_0, \sigma_0\rangle \to \langle l_1, \sigma'_1\rangle \ldots \to \langle l_k, \sigma'_k\rangle \to q_e$ in $\mathcal{M}_C$.*
2. *Let $r = \langle l_0, \sigma_0\rangle \to \ldots \to \langle l_k, \sigma_k\rangle \to q_e$ be an initial path in $\mathcal{M}_C$ and $r' = \langle l_0, s_0\rangle \to \ldots \to \langle l_k, s_k\rangle \to q_e^\sharp$ be an initial path in $\mathcal{A}$ obeying a non-randomized strategy for $\mathcal{A}$. Then $w(r) = w_\phi(r')$.*

Proof.

1. This follows immediately from the fact that C is deterministic.
2. Let $\langle l,\sigma\rangle \to \langle l',\sigma'\rangle$ in $\mathcal{M}_C$ and $\langle l,s\rangle \to \langle l',s'\rangle$ in $\mathcal{A}$. If $\langle l,\sigma\rangle \in \Lambda_P$, then $\langle l,s\rangle \in \Theta_P$. Definition 6 implies that $\delta(l,l') = p$ for a $p \in [0,1]$. Theorem 2b then gives us $(\langle l,s\rangle, p, \langle l',s'\rangle) \in \Theta_P$. If $\langle l,\sigma\rangle \in \Lambda_A$, then $\langle l,s\rangle \in \Theta_A$, and $(\langle l,\sigma\rangle, \langle l',\sigma'\rangle) \in \Lambda_A$ and $(\langle l,s\rangle, \langle l',s'\rangle) \in \Theta_A$. Since q_e resp. $q_e^\sharp$ is the last state of the path, $l_k = l_e$. Hence $|\mathrm{Succs}(\langle l_k,\sigma_k\rangle)| = |\mathrm{Succs}(\langle l_k,s_k\rangle)| = 1$ by Def. 6 and inspecting Alg. 1. These observations together with the fact that ϕ is non-randomized and the definition of w resp. w_ϕ prove the proposition.

Now we show that there is a strategy for $\mathcal{A}$ that "simulates" the behaviour of $\mathcal{M}_C$.

Lemma 2. *There exists a non-randomized strategy ϕ for $\mathcal{A}$ with the following property:*

$$P_{\mathcal{M}_C}(q_e) = P_{\mathcal{A}_P[\phi]}(q_e^\sharp).$$

Proof. ϕ depends on the sequence of already visited labels. We describe $\phi's$ choices: Let $\langle l_0,s_0\rangle \to \ldots \to \langle l_k,s_k\rangle$ be an initial path of $\mathcal{M}_P^\sharp$, with l_k not corresponding to a probabilistic choice in P.

If the location sequence $l_0,\ldots,l_k$ belongs to a valid initial path $\langle l_0,\sigma_0\rangle \to \ldots \to \langle l_k,\sigma_k\rangle$ in $\mathcal{M}_C$, there cannot be another initial path with the same location sequence $l_0,\ldots,l_k$ and different abstract elements due to Lemma 1. $\langle l_k,\sigma_k\rangle$ then has a unique successor state $\langle l,\sigma\rangle$.

By Theorem 2a we see that $\langle l_k,s_k\rangle$ has at least one successor $\langle l,s\rangle$ with $\sigma \in \gamma(s)$. Hence we set $\phi(\langle l_0,s_0\rangle \to \ldots \to \langle l_k,s_k\rangle) = \langle l,s\rangle$ (it does not matter which possible successor $\langle l,s\rangle$ with this property one chooses).

If $l_0,\ldots,l_k$ does not belong to a valid initial path in $\mathcal{M}_C$, we can fix an arbitrary successor $\langle l,s\rangle$ of $\langle l_k,s_k\rangle$ and set $\phi(\langle l_0,s_0\rangle \to \ldots \to \langle l_k,s_k\rangle) = \langle l,s\rangle$.

For initial paths containing $q_e^\sharp$ the trivial choice for ϕ has to be $q_e^\sharp$.

Now observe that for every initial path $r = \langle l_0,s_0\rangle \to \ldots \to \langle l_k,s_k\rangle \to q_e^\sharp \in \mathrm{Cyl}(\mathcal{A}[\phi], q_e^\sharp)$ there cannot be another initial path with the same location sequence. This can be shown by using the argumentation of Lemma 1 and the definition of ϕ. Furthermore we can define a map $f : \mathrm{Cyl}(\mathcal{A}[\phi], q_e^\sharp) \to \mathrm{Cyl}(\mathcal{M}_C, q_e)$ that maps r to the initial path $f(r') = \langle l_0,\sigma_0\rangle \to \ldots \to \langle l_k,\sigma_k\rangle \to q_e \in \mathrm{Cyl}(\mathcal{M}_C, q_e)$, which is unique (see Lemma 1). It is easy to see that f is a bijection. With Lemma 2 we see that $w_\phi(r) = w(f(r))$.

Now Theorem 3 follows as a simple corollary from the existence of an optimal and a pessimal strategy.