Probabilistic CEGAR*

Holger Hermanns, Björn Wachter, and Lijun Zhang

Universität des Saarlandes, Saarbrücken, Germany
{hermanns,bwachter,zhang}@cs.uni-sb.de

Abstract. Counterexample-guided abstraction refinement (CEGAR) has been *en vogue* for the automatic verification of very large systems in the past years. When trying to apply CEGAR to the verification of probabilistic systems, various foundational questions arise. This paper explores them in the context of predicate abstraction.

1 Introduction

Probabilistic behavioral descriptions are widely used to analyze and verify systems that exhibit "quantified uncertainty", such as embedded, networked, or biological systems. The semantic model for such systems often are Markov chains or Markov decision processes. We here consider homogeneous discrete-time Markov chains (MCs) and Markov decision processes (MDPs). Properties of these systems can be specified by formulas in temporal logics such as PCTL [1], where for instance quantitative probabilistic reachability ("the probability to reach a set of bad states is at most 3%") is expressible. Model checking algorithms for such logics have been devised mainly for finite-state MCs [1] and MDPs [2], and effective tool support is provided by probabilistic model checkers such as Prism [3] or Mrmc [4]. Despite its remarkable versatility, the approach is limited by the state explosion problem, aggravated by the cost of numerical computation compared to Boolean CTL model checking.

Predicate abstraction [5] is a method for creating finite abstract models of non-probabilistic systems where symbolic expressions, so-called predicates, induce a partitioning of its (potentially infinite) state space into a finite number of regions. For automation, it is typically coupled [6,7] with *counterexample-guided abstraction refinement* (CEGAR) [8] where an initially very coarse abstraction is refined using diagnostic information (predicates) derived from abstract counterexamples, until either the property is proved or refuted.

In this paper, we discuss how counterexample-guided abstraction refinement can be developed in a probabilistic setting. Predicate abstraction without abstraction refinement has been presented in [9] for a guarded command language whose concrete semantics maps to MDPs – more precisely, to probabilistic automata [10]. This is the natural basis for our present work. We restrict our

* This work is supported by DFG as part of the Transregional Collaborative Research Center SFB/TR 14 AVACS and by the NWO-DFG bilateral project VOSS.

treatment to probabilistic reachability and aim to determine if the probability to reach a set of bad states exceeds a given threshold.

The core challenge of developing probabilistic CEGAR lies in the notion and analysis of counterexamples. In the traditional setting, an abstract counterexample is a single finite path (to some bad state) and counterexample analysis consists in checking if the concrete model exhibits a corresponding error path. In contrast, a counterexample to a probabilistic reachability property can be viewed as a finite, but generally cyclic, Markov chain [11]. Due to these cycles, probabilistic counterexample analysis is not directly amenable to conventional methods. We circumvent this problem, by preprocessing the abstract counterexample using the strongest evidence idea of [12]: We generate a finite set *afp* of abstract finite paths that together carry enough abstract probability mass, and formulate the problem of computing the realizable probability mass of *afp* in terms of a weighted MAX-SMT problem [13]. The set *afp* is built incrementally in an *on-the-fly* manner, until either enough probability is realizable, or *afp* cannot be enriched with sufficient probability mass to make the probability threshold realizable, in which case the counterexample is spurious.

These ingredients result in a theory for probabilistic CEGAR. We have evaluated the approach on various case studies. Indeed, CEGAR entirely mechanizes the verification process: predicates are added mechanically on demand based on counterexample analysis. To this end, our implementation employs interpolation [14] to generate new predicates from spurious paths.

Related Work. Aljazzar & Hermanns [15] and Katoen & Han [12] introduced concepts and algorithms to deal with probabilistic counterexamples. However, both papers do not consider counterexamples in the context of abstraction.

We are not aware of previous work that combines abstraction refinement with predicate abstraction for probabilistic systems. Up to now different abstraction refinement approaches for finite-state probabilistic models have been proposed: among those [16,17,18,19] do not exploit counterexamples, while Chatterjee et. al [11] apply CEGAR to finite probabilistic two-player game structures. Although finite MDPs are a special case of these game structures, our work differs from [11] since it considers infinite-state models, stays more in the classical predicate abstraction realm, and is supplemented with a running implementation.

Outline. We briefly review our previous work on predicate abstraction for probabilistic programs in Section 2. The contributions of this paper, counterexample analysis and abstraction refinement for probabilistic programs, are presented in Section 3. Experimental results are presented in Section 4.

2 Preliminaries

Probabilistic Programs. We consider probabilistic programs in a guarded command language [9] which is inspired by the PRISM [3] input language, but supports infinite data domains. We fix a finite set of program variables $\mathtt{X}$ and a finite set of actions Act. Variables are typed in a definition such as `i : int`. We denote the set of expressions over the set of variables V by $Expr_V$ and the set of Boolean expression over V by $BExpr_V$. An *assignment* is a total function $E : \mathtt{X} \to Expr_{\mathtt{X}}$ from variables $\mathtt{x} \in \mathtt{X}$ to expressions $E(\mathtt{x})$. Given an expression $\mathtt{e} \in Expr_{\mathtt{X}}$ and an assignment E, we denote by $\mathtt{e}[\mathtt{X}/E(\mathtt{X})]$ the expression obtained from $\mathtt{e}$ by substituting each occurrence of a variable $\mathtt{x}$ with $E(\mathtt{x})$.

A *guarded command* $\mathtt{c}$ consists of an action a, a guard $\mathtt{g} \in BExpr_{\mathtt{X}}$ and assignments $E_{\mathtt{u}_1}, ..., E_{\mathtt{u}_k}$ weighted with probabilities $p_1, ..., p_k$ where $\sum_{i=1}^{k} p_i = 1$. We denote by $\mathtt{X}' = E$ the simultaneous update E of variables $\mathtt{X}$. With the i-th update of $\mathtt{c}$, we associate a unique update label $\mathtt{u}_i \in \mathtt{U}$. Updates are syntactically separated by a "+": $[a]\ \mathtt{g} \to \mathtt{p}_1 : \mathtt{X}'{=}E_{\mathtt{u}_1} + \ldots + \mathtt{p}_k : \mathtt{X}'{=}E_{\mathtt{u}_k}$. If the guard is satisfied, the i-th update will be executed with probability p_i. For $\mathtt{c}$, we write $a_{\mathtt{c}}$ for its action, $\mathtt{g}_{\mathtt{c}}$ for its guard. If $\mathtt{c}$ is clear from the context, we write $a, \mathtt{g}$ and $\mathtt{u}_i$ instead. We define the weakest liberal precondition of an expression with respect to an update as follows: $\mathtt{WP}_{E_{\mathtt{u}}}(\mathtt{e}) = \mathtt{e}[\mathtt{X}/E_{\mathtt{u}}(\mathtt{X})]$.

A *program* $\mathtt{P} = (\mathtt{X}, \mathtt{I}, \mathtt{C})$ consists of a Boolean expression $\mathtt{I} \in BExpr_{\mathtt{X}}$ that defines the set of initial states and a set of guarded commands $\mathtt{C}$. The program has distinctly labeled guarded commands – two different commands have distinct actions and update labels.

Probabilistic Automata. The semantics of a probabilistic program is a probabilistic automaton. To enable reconstruction of commands and updates from the semantics, automata are decorated with labels from two alphabets: an action alphabet Act for commands, and an update alphabet $\mathtt{U}$ for probabilistic choices. A *simple distribution* π over Σ is a function $\pi : \Sigma \to [0,1]$ such that $\sum_{s \in \Sigma} \pi(s) = 1$. Let $Distr_{\Sigma}$ denote the set of all simple distributions over Σ.

We decorate the distribution with the alphabet $\mathtt{U}$ as follows: an *(update-labeled) distribution* π over $\mathtt{U} \times \Sigma$ is a distribution $\pi : \mathtt{U} \times \Sigma \to [0,1]$ such that (i) the update alphabet is right-unique, i.e., $\pi(\mathtt{u}, s) > 0$ and $\pi(\mathtt{u}, s')$ implies that $s = s'$ and (ii) $\sum_{\mathtt{u} \in \mathtt{U}} \sum_{s \in \Sigma} \pi(\mathtt{u}, s) = 1$. Let $Distr_{(\mathtt{U},\Sigma)}$ denote the set of update-labeled distributions over $\mathtt{U} \times \Sigma$. For $\pi \in Distr_{(\mathtt{U},\Sigma)}$, we call the set of states $Supp(\pi) = \{(\mathtt{u}, s) \mid \pi(\mathtt{u}, s) > 0\}$ the support of π.

A *probabilistic update automaton* $\mathcal{M}$ is a tuple $(\Sigma, I, Act, \mathtt{U}, R)$ where Σ is a set of states, $I \subseteq \Sigma$ is a set of initial states, Act is the action alphabet, $\mathtt{U}$ is the update alphabet, and $R \subseteq \Sigma \times Act \times Distr_{(\mathtt{U},\Sigma)}$ the probabilistic transition relation. $\mathcal{M}$ is called finite if Σ is finite. A finite path is a finite sequence $(s_0, a_0, \mathtt{u}_0, \pi_0), (s_1, a_1, \mathtt{u}_1, \pi_1), \ldots s_n$ such that $s_0 \in I$, $(s_i, a_i, \pi_i) \in R$, and $(\mathtt{u}_i, s_{i+1}) \in Supp(\pi_i)$ for all $i = 0, \ldots, n-1$. Let $Path_{fin}(\mathcal{M})$ denote the set of all finite paths over $\mathcal{M}$. We write $\sigma \leq \sigma'$, if the finite path σ is a prefix of σ'. A finite path σ is maximal if $\sigma \leq \sigma'$ implies that $\sigma = \sigma'$. An infinite path σ is an infinite sequence $(s_0, a_0, \mathtt{u}_0, \pi_0), (s_1, a_1, \mathtt{u}_1, \pi_1), \ldots$ starting with an initial state $s_0 \in I$, $(s_i, a_i, \pi_i) \in R$, and $(\mathtt{u}_i, s_{i+1}) \in Supp(\pi_i)$ for all $i = 0, 1, \ldots$. Let $Path(\mathcal{M})$ denote the set of all infinite or maximal paths over $\mathcal{M}$. For $\sigma \in Path_{fin}$, let $C(\sigma) = \{\sigma' \in Path(\mathcal{M}) \mid \sigma \leq \sigma'\}$ denote the *cylinder set* for σ. For $\sigma \in Path$, let $\sigma[i] = s_i$ denote the $i+1$-th state of σ.

Our definition of probabilistic update automata adds labels at probabilistic choices to the probabilistic automata of [10]. This does not give additional modeling power, it rather allows us to develop the CEGAR approach succinctly. Dropping the labels of the distributions, $\mathcal{M}$ induces a probabilistic automaton $ind(\mathcal{M})$ in the style of [10] as follows: replace every update-labeled distribution π by its induced distribution $ind(\pi)$, defined by $ind(\pi)(s) = \sum_{\mathtt{u} \in \mathtt{U}} \pi(\mathtt{u}, s)$. If the context is clear, we use $\mathcal{M}$ and $ind(\mathcal{M})$ interchangeably.

An adversary is a resolution of non-determinism. In general, an adversary A of an automaton $\mathcal{M}$ is a function from paths to pairs of actions and distributions. We let $\mathcal{D}_\delta$ denote the Dirac distribution defined by: $\mathcal{D}_\delta(\delta) = 1$ where δ is a special symbol for termination. An adversary A is called *simple* if it only looks at the last state in a path, i.e. if it is a function $A : \Sigma \to (Act \times Distr_\Sigma) \cup \{\mathcal{D}_\delta\}$. Note that if $A(s) = \mathcal{D}_\delta$, the adversary A decides to stop at state s. For a given state $s \in \Sigma$ and an adversary A, let P_s^A denote the corresponding probability measure [20] over $Path(\mathcal{M})$. Given a probabilistic automaton $\mathcal{M}$, a simple adversary A induces an MC $\mathcal{M}_A = (\Sigma, I, Act, R_A)$ where $R_A = \{(s, a, \pi) \in R \mid A(s) = (a, \pi)\}$. Note s has no outgoing transitions if $A(s) = \mathcal{D}_\delta$.

MCs and MDPs are special cases of probabilistic automata. An MC is a deterministic probabilistic automaton, i.e. an automaton where for every state s there is at most one transition $(s, a, \pi) \in R$. An MDP is an action-deterministic probabilistic automaton, i.e. an automaton where for every pair $s \in \Sigma$ and $a \in Act$, there exists at most one π with $(s, a, \pi) \in R$.

Program Semantics. A *state* over variables $\mathtt{X}$ is a type-consistent total function from variables in $\mathtt{X}$ to their semantic domains. We denote the set of states by $\Sigma(\mathtt{X})$ or Σ for short and a single state by s. For an expression $\mathtt{e} \in Expr_{\mathtt{X}}$, we denote by $[\![\mathtt{e}]\!]_s$ its valuation in state s. The valuation of a Boolean expression $\mathtt{e}$ is a value $[\![\mathtt{e}]\!]_s \in \{0, 1\}$ (0 for "false", 1 for "true"). For a Boolean expression $\mathtt{e}$ and a state s, we write $s \vDash \mathtt{e}$ iff $[\![\mathtt{e}]\!]_s = 1$. Semantic brackets around a Boolean expression $\mathtt{e}$ without a subscript denote the set of states fulfilling $\mathtt{e}$, i.e. $[\![\mathtt{e}]\!] = \{s \in \Sigma \mid s \vDash \mathtt{e}\}$.

The semantics of a program $\mathtt{P} = (\mathtt{X}, \mathtt{I}, \mathtt{C})$ is the probabilistic update automaton $\mathcal{M} = (\Sigma, I, Act, R)$ with set of states $\Sigma = \Sigma(\mathtt{X})$, set of initial states $I = [\![\mathtt{I}]\!]$, set of actions $Act = \{a_{\mathtt{c}} \mid \mathtt{c} \in \mathtt{C}\}$, and transitions induced by the guarded commands $R = \bigcup_{\mathtt{c} \in \mathtt{C}} [\![\mathtt{c}]\!]$ where $(s, a, \pi) \in [\![\mathtt{c}]\!]$ if $s \vDash \mathtt{g}$ and π such that $\pi(\mathtt{u}_i, s') = p_i$ if $s'(\mathtt{x}) = [\![E_{\mathtt{u}_i}(\mathtt{x})]\!]_s$ for all $\mathtt{x} \in \mathtt{X}$.

Properties. In this paper, we consider probabilistic reachability properties which we write as $Reach_{\leq p}(\mathtt{e})$ where $p \in [0, 1]$ is a probability value, and the Boolean expression $\mathtt{e} \in BExpr_{\mathtt{X}}$ describes the states to be reached. For a state s and an adversary A, let $p_s^A(\leadsto \mathtt{e}) = P_s^A(\{\sigma \in Path(\mathcal{M}) \mid \exists i \in \mathbb{N}\ \sigma[i] \vDash \mathtt{e}\})$ be the probability of set of paths reaching an $\mathtt{e}$-state. Then, $Reach_{\leq p}(\mathtt{e})$ is satisfied by s if $p_s^A(\leadsto \mathtt{e}) \leq p$ for all adversary A, and it is satisfied by the model if it is satisfied by all initial states. Algorithmically, it is sufficient to only consider simple adversaries [2], as extremal probabilities are already attained among them.

Predicate Abstraction. Predicates are Boolean expressions over the program variables. A predicate φ stands for the set of states satisfying it, namely $[\![\varphi]\!]$. We fix a set of predicates $\mathcal{P} = \{\varphi_1, ..., \varphi_n\}$. The set $\mathcal{P}$ partitions the states into disjoint sets characterized by which predicates hold and which not. An equivalence class can therefore be represented by a bit vector of length n. We call such a bit-vector an *abstract state* and denote the set of abstract states by $\Sigma^\sharp$. We define a state-abstraction function by: $h_{\mathcal{P}}(s) = ([\![\varphi_1]\!]_s, ..., [\![\varphi_n]\!]_s)$. For a given $s^\sharp \in \Sigma^\sharp$, we call the corresponding equivalence class the concretization of $s^\sharp$ and denote it by $\gamma(s^\sharp)$. The concretization of $s^\sharp$ is characterized by a Boolean expression $F(s^\sharp)$ such that $\gamma(s^\sharp) = [\![F(s^\sharp)]\!]$. $F(s^\sharp)$ is exactly the conjunction containing satisfied predicates as positive literals and unsatisfied ones as negated literals.

We recall predicate abstraction of probabilistic programs [9]: The state abstraction $h_{\mathcal{P}}$ induces a quotient automaton, denoted by $\mathcal{M}^\sharp = (\Sigma^\sharp, I^\sharp, Act, \mathtt{U}, R^\sharp)$ with the set of initial states $I^\sharp = \{h(s) \mid s \in I\}$, and transitions $R^\sharp = \{(h(s), a, h(\pi)) \mid (s, a, \pi) \in R\}$ where $h(\pi) = \{(\mathtt{u}, h(s)) : p \mid \pi(\mathtt{u}, s) = p\}$. If the reachability property is satisfied by the quotient automaton $\mathcal{M}^\sharp$, we can safely conclude that it holds for the original model $\mathcal{M}$ as well. The soundness follows from the fact that the quotient automaton $\mathcal{M}^\sharp$ simulates $\mathcal{M}$.

3 Refinement

In this section, we present a novel refinement scheme for probabilistic programs based on CEGAR (counterexample-guided abstraction refinement). The plain CEGAR approach is the obvious strategy also to follow in the probabilistic case: start with a coarse abstraction and successively refine it using predicates learned from spurious counterexamples until either a realizable counterexample is found or the abstract model is precise enough to establish the property. However, in order to put refinement to work for probabilistic models, several questions of both principal and practical nature need to be answered. We *(i)* need to identify what an abstract counterexample constitutes, *(ii)* lift it to the concrete system, *(iii)* decide if it is spurious, and *(iv)* identify appropriate predicates to refine the abstract quotient automaton.

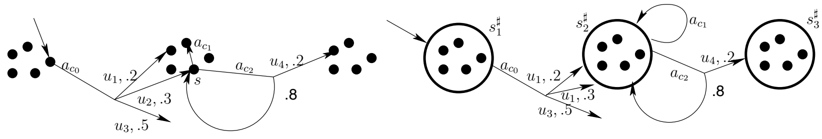
Counterexamples for Quotient Automata. Intuitively, a counterexample is a pair of an initial state and an adversary that violates the property to be checked. This pair induces an MC in the abstract setting. In the sequel, we fix the probabilistic update automaton $\mathcal{M}$ and the reachability property $Reach_{\leq p}(\mathtt{e})$. Let $\mathcal{M}^\sharp$ be the quotient automaton of $\mathcal{M}$.

Definition 1. *A counterexample for $Reach_{\leq p}(\mathtt{e})$ is a pair $(s^\sharp, A^\sharp)$ where $s^\sharp \in I^\sharp$ is an initial state and $A^\sharp$ is an adversary such that $P_{s^\sharp}^{A^\sharp}(\rightsquigarrow \mathtt{e}) > p$. In this case, $A^\sharp$ is called a* counter-adversary*.*

Spurious Counterexamples. In the non-probabilistic setting, a counterexample is a path, which is called spurious if there does not exist a corresponding concrete path. We now introduce the notion of spurious counterexamples for probabilistic automata based on the concretization of an abstract counter-adversary and an abstract counterexample.

Concretization of Counter-adversaries. For a counter-adversary $A^\sharp$ in the quotient $\mathcal{M}^\sharp$, its concretization $\gamma(A^\sharp)$ is an adversary in M defined by: $\gamma(A^\sharp)(s)$ equals (a_c, π) if $A^\sharp(s^\sharp) = (a_\mathsf{c}, \pi^\sharp)$ with $s^\sharp = h(s)$ and $\pi^\sharp = h(\pi)$, otherwise $\mathcal{D}_\delta$.

In case of $\gamma(A^\sharp)(s) = \mathcal{D}_\delta$, the adversary $A^\sharp$ has chosen $(a_\mathsf{c}, \pi^\sharp)$ from $s^\sharp$, however s does not satisfy the guard g_c associated with c. Thus, we let $\gamma(A^\sharp)$ stop at state s, as no corresponding concretization exists. Recall that the program has distinctly labeled guarded commands, thus we can choose at most one corresponding outgoing concrete transition. For illustration, consider the fragment of a probabilistic automaton and its corresponding quotient automaton in the figure below. If the adversary $A^\sharp$ chooses a_{c_2} at state $s_2^\sharp$, the concretization $\gamma(A^\sharp)$ chooses also action a_{c_2} at state s.



Concretization of Counterexamples. Now consider a counterexample $(s^\sharp, A^\sharp)$. Its concretization, denoted $\gamma(s^\sharp, A^\sharp)$, is the set: $\{(s, A) \mid A = \gamma(A^\sharp) \wedge s \in (I \cap \gamma(s^\sharp))\}$. Directly linked to the cardinality of the initial state set, the concretization can contain many (even infinitely many) elements, and thus induce many MCs. The reachability probability $P(\leadsto \mathsf{e})$ may differ from element to element. A counterexample $(s^\sharp, A^\sharp)$ is spurious if its concretization does not contain a pair (s, A) such that the probability threshold is exceeded. In other words, a spurious counterexample does not induce any concrete MC for which the probability measure of reaching concrete e-states exceeds the specified threshold.

Definition 2. *Let $(s^\sharp, A^\sharp)$ be a counterexample for Reach$_{\le p}(\mathsf{e})$ in $\mathcal{M}^\sharp$. Then, $(s^\sharp, A^\sharp)$ is called* realizable *if there exists $(s, A) \in \gamma(s^\sharp, A^\sharp)$ such that $P_s^A(\leadsto \mathsf{e}) > p$. Otherwise we say that the counterexample is* spurious.

Checking Counterexamples. Checking realizability of counterexamples is a key element of the refinement procedure: If a counterexample turns out to be realizable, the property is refuted with A playing the role of a counter-adversary in the concrete model, which can be used for debugging purposes. Otherwise, the abstract model is too coarse and additional predicates will need to be added to eliminate the false negative.

Overall Idea. In the non-probabilistic setting, an abstract counterexample is a single finite abstract path $\sigma^\sharp$ starting in an abstract initial state. Its concretization is a set of corresponding paths in the concrete model each of which starts in some concrete initial state and respects the concrete transition relation. This set might potentially be infinite. If it is empty, the counterexample is spurious. It is common practice to check emptiness of the concretization by expressing the behavior enforced on the concrete program by the abstract path implicitly by a formula and checking the satisfiability of that formula [6,7]. If the formula is satisfied, then the concretization is non-empty, and we have found a concrete counterexample violating the property. In this case, the counterexample is realizable. Otherwise, it is spurious, and additional predicates can be extracted from the path $\sigma^\sharp$ for refinement.

In the probabilistic setting, however, the situation is much more involved. What makes the counterexample $(s^\sharp, A^\sharp)$ realizable is a concrete initial state $s \in (I \cap \gamma(s^\sharp))$ and adversary A such that the probability of reaching an $\mathtt{e}$-state in the thus induced concrete MC exceeds the given threshold p. All candidates (s, A) are contained in $\gamma(s^\sharp, A^\sharp)$ but this set might be infinite. We preprocess the abstract counterexample using the strongest evidence idea of Katoen & Han [12]. They have devised a method that, for a given MC, can be used to construct the smallest set of paths reaching $\mathtt{e}$-states with an accumulated probability measure above p. This fits well to our needs.

As the abstract counterexample $(s^\sharp, A^\sharp)$ induces an abstract MC, we can apply the algorithm from [12] yielding *a finite* set of *finite paths* in the quotient automaton starting from state $s^\sharp$, such that the probability measure exceeds p. To check if the counterexample is spurious, our goal is then to compute how much measure out of this set of paths can be reproduced in $\mathcal{M}$ with respect to *any* $(s, A) \in \gamma(s^\sharp, A^\sharp)$. If that is indeed larger than the threshold p for some (s, A), we have found a realizable counterexample. Otherwise we may be able to conclude that it is spurious, or conclude that more work is needed, as we will explain below.

Spuriousness of Abstract Paths. Before coming to adversaries, we first explain how to check if a single abstract path is realizable or spurious. Let $(s^\sharp, A^\sharp)$ be a counterexample and let $\sigma^\sharp = (s_0^\sharp, a_0, \mathtt{u}_0, \pi_0^\sharp)\ (s_1^\sharp, a_1, \mathtt{u}_1, \pi_1^\sharp) \dots s_k^\sharp$ be a path in $\mathcal{M}^\sharp_{A^\sharp}$ where $s_0^\sharp = s^\sharp$ and $s_k^\sharp$ satisfies $\mathtt{e}$. The concretization $\gamma(\sigma^\sharp)$ of an abstract path $\sigma^\sharp$ is a set of finite paths in $\mathcal{M}$ with consistent states, and the same update and action labels, i.e. $\gamma(\sigma^\sharp) = \{(s_0, a_0, \mathtt{u}_0, \pi_0) \dots s_k \mid (s_0, \dots, s_k) \models \mathit{TF}(\sigma^\sharp)\}$ where $\mathit{TF}(\sigma^\sharp)$ is the trace formula which is defined by:

$$\mathit{TF}(\sigma^\sharp) = \mathtt{I}(\mathtt{X}_0) \wedge \bigwedge_{i=0}^{k} F(s_i^\sharp)(X_i) \wedge \bigwedge_{i=0}^{k-1} \big(\mathtt{g}_{\mathtt{c}_i}(\mathtt{X}_i) \wedge \mathtt{X}_{i+1} = E_{u_i}(\mathtt{X}_i)\big) \wedge \mathtt{e}(\mathtt{X}_k) \ .$$

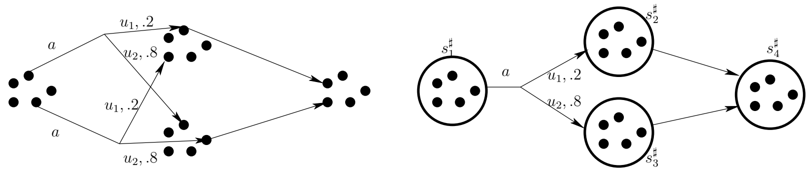
The measure of $\sigma^\sharp$ under $(s^\sharp, A^\sharp)$ is $\prod_{i=0}^{k-1} \pi_i^\sharp(\mathtt{u}_i, s_{i+1}^\sharp)$. Note that the paths in the concretization of $\sigma^\sharp$ share the same measure. The path $\sigma^\sharp$ is called *realizable* if its concretization is non-empty, $\gamma(\sigma^\sharp) \neq \emptyset$, otherwise it is called *spurious*. As for the non-probabilistic setting [7,6], an abstract path is realizable if its trace formula is satisfiable or, equivalently, its weakest precondition. The weakest precondition of an abstract path is formalized in Figure 1 as the repeated application of the standard *syntactic weakest precondition* $\mathtt{WP}_E(e)$ where $\mathtt{WP}_E(e) := e[\mathtt{X}/E(\mathtt{X})]$ for an expression e and an update $\mathtt{X'}$=E.

```
WP($\sigma^\sharp = (s_0^\sharp, a_{\mathtt{c}_0}, \mathtt{u}_0, \pi_0^\sharp) \dots s_k^\sharp$)
1: $exp_{\sigma^\sharp} \leftarrow F(s_k^\sharp) \wedge \mathtt{e}$
2: for ($j = k$ .. $0$) do
3:     $exp_{\sigma^\sharp} \leftarrow \mathtt{g}_{\mathtt{c}_j} \wedge F(s_j^\sharp) \wedge \mathtt{WP}_{\mathtt{u}_j}(exp_{\sigma^\sharp})$
4: end for
5: return $exp_{\sigma^\sharp} \wedge I$
```

Fig. 1. WP of an abstract path $\sigma^\sharp$

Lemma 1. *For an abstract path $\sigma^\sharp$, the following statements are equivalent: (i) The weakest precondition* $\mathrm{WP}(\sigma^\sharp)$ *of path $\sigma^\sharp$ is satisfiable. (ii) The trace formula $TF(\sigma^\sharp)$ of $\sigma^\sharp$ is satisfiable. (iii) The path $\sigma^\sharp$ is realizable, i.e. $\gamma(\sigma^\sharp) \neq \emptyset$.*

Checking Spuriousness. The counterexample $(s^\sharp, A^\sharp)$ is guaranteed to be realizable if it has a concretization with sufficiently high measure. We assume a nonempty set *afp* of abstract paths respecting $(s^\sharp, A^\sharp)$. Note that corresponding concrete paths may start in different initial states, so that the probability in the concrete model is possibly lower. Let us consider an abstract path $\sigma^\sharp$. For all $\sigma \in \gamma(\sigma^\sharp)$ with $\sigma = (s_0, a_0, \mathtt{u}_0, \pi_0) \ldots s_k$, the measure of the cylinder set $C(\sigma)$ under $(s, A) \in \gamma(s^\sharp, A^\sharp)$ is given by $\prod_{i=0}^{k-1} \pi_i(\mathtt{u}_i, s_{i+1})$ if $s = s_0$, which is the same as $\prod_{i=0}^{k-1} \pi_i^\sharp(\mathtt{u}_i, s_{i+1}^\sharp)$. For a set *afp* of abstract path we let $\gamma(\mathit{afp}) = \bigcup_{\sigma^\sharp \in \mathit{afp}} \gamma(\sigma^\sharp)$ denote the union of the concretizations. Now an interesting issue arises: what is the maximal probability measure of the set $\gamma(\mathit{afp})$ under some concretization of $\gamma(s^\sharp, A^\sharp)$. For illustration, consider the figure below where *afp* consists of two disjoint abstract paths $\sigma_1^\sharp, \sigma_2^\sharp$, but the intersection is empty: $\mathit{exp}_{\sigma_1^\sharp} \wedge \mathit{exp}_{\sigma_2^\sharp} = \emptyset$, hence only the maximum of both can be achieved.



We resolve this problem by using weakest preconditions of abstract paths. Given an abstract path $\sigma^\sharp$, the backwards algorithm in Figure 1 computes its weakest precondition, i.e. those initial states in which a path from the concretization of $\sigma^\sharp$ starts. We use these weakest preconditions to obtain subsets of the given set of abstract paths sharing a common concrete initial state. The subset with maximal probability gives us the actual measure in the concrete model.

For $\mathit{afp} = \{\sigma_1^\sharp, \ldots, \sigma_n^\sharp\}$, let $\mathit{exp}_1, \ldots, \mathit{exp}_n$ denote the weakest preconditions returned by $\mathrm{WP}(\sigma_i^\sharp)$. Moreover, for each of them the probability measure of path $\sigma_i^\sharp$ is given as a weight, denoted by p_i, which corresponds to the probability of the set $\gamma(\sigma_i^\sharp)$ starting from some initial state in exp_i. We now formulate the problem of computing the realizable probability mass of a set of abstract paths in terms of a weighted MAX-SMT [13] problem, which consists in finding an assignment of `X` such that the total weight of the satisfied expression is maximal. Formally, it is defined by: $\textsc{MaxSmt}(\mathit{exp}_1, \ldots, \mathit{exp}_n) = \max\left\{\sum_{i=1}^{n} [\![\mathit{exp}_i]\!]_s \cdot p_i \mid s \in [\![\mathtt{I} \wedge F(s^\sharp)]\!]\right\}$.

Lemma 2. *Let $(s^\sharp, A^\sharp)$ be a counterexample for $\mathit{Reach}_{\leq p}(\mathsf{e})$, and let $\mathit{afp} = \{\sigma_1^\sharp, \ldots, \sigma_n^\sharp\}$ be a set of abstract paths with measure greater than p. It holds:*

(i) $\textsc{MaxSmt}(\mathit{exp}_1, \ldots, \mathit{exp}_n) > p$ *implies that $(s^\sharp, A^\sharp)$ is realizable,*
(ii) $\textsc{MaxSmt}(\mathit{exp}_1, \ldots, \mathit{exp}_n) + P_{s^\sharp}^{A^\sharp}(\rightsquigarrow \mathsf{e}) - P_{s^\sharp}^{A^\sharp}(\mathit{afp}) \leq p$ *implies that the counterexample $(s^\sharp, A^\sharp)$ is spurious.*

Let $\varepsilon = P_{s^\sharp}^{A^\sharp}(\rightsquigarrow \mathsf{e}) - P_{s^\sharp}^{A^\sharp}(afp)$ denote the probability of the set of abstract paths which violate the property $Reach_{\leq p}(\mathsf{e})$, but are not part of the set *afp*. The lemma indicates that the decision algorithm is only partial: if the value $\textsc{MaxSmt}(exp_1, \ldots, exp_n)$ lies in the interval $(p - \epsilon, p]$, we are not sure whether the counterexample $(s^\sharp, A^\sharp)$ is spurious or realizable. By enlarging the set *afp*, the ε can be made arbitrarily small. We will see later how this is exploited for the CEGAR algorithm.

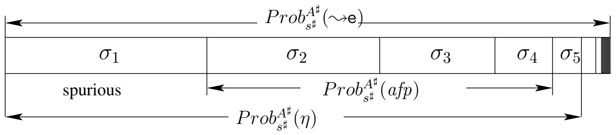
Obtaining Predicates. There are two sources of potential imprecision: spurious abstract paths or a too coarse abstraction of the initial states.

Predicates to Remove Spurious Paths. Let $(s^\sharp, A^\sharp)$ be a counterexample in $\mathcal{M}^\sharp_{A^\sharp}$. Let $\sigma^\sharp = (s_0^\sharp, a_0, \mathsf{u}_0, \pi_0^\sharp)(s_1^\sharp, a_1, \mathsf{u}_1, \pi_1^\sharp) \ldots s_k^\sharp$ be a path such that $s_0^\sharp = s^\sharp$ and $\sigma^\sharp$ satisfies $\rightsquigarrow \mathsf{e}$. Assume that $\sigma^\sharp$ is spurious. Our goal is to find predicates to eliminate the spurious abstract path. The abstract path resolves both nondeterministic choice between different commands, and probabilistic choice between different updates. That enables us to use standard techniques developed in the non-probabilistic setting to find predicates. Here we employ interpolation and apply it to the trace formula of the abstract path along the lines of [21].

Predicates to Separate Initial States. Observe the case where no path in *afp* is spurious but the realizable probability of the paths is lower than the probability threshold p, i.e., $\textsc{MaxSmt}(exp_1, \ldots, exp_n) \leq p$. In this case, the initial state $s^\sharp$ may be too coarse. To this end, we choose the maximal solution obtained from $\textsc{MaxSmt}$. Let ψ^- denote the conjunction of non-satisfied exp_i, and ψ^+ denote the conjunction of satisfied exp_i. Obviously, $\psi^- \wedge \psi^+$ is not satisfiable. Hence, interpolants can be found to refine the abstraction of the initial states. Note that this is a heuristic choice and does not guarantee removal of the abstract counterexample.

CEGAR Algorithm. At the start of each iteration of the CEGAR loop, a quotient automaton $\mathcal{M}^\sharp$ is built using the current set of predicates. We submit the quotient automaton and the property to a probabilistic model checker. Due to soundness of the abstraction, we can safely report success if the property is satisfied in $\mathcal{M}^\sharp$. Otherwise the model checker produces an abstract counterexample $(s^\sharp, A^\sharp)$ which is passed to the counterexample analysis phase.

Counterexample analysis constitutes the next phase: Along the ideas of strongest evidence [12], we maintain a sequence $\eta = \langle \sigma_1, \sigma_2, ..., \sigma_n \rangle$ of abstract paths reaching an e-state in the MC induced by $(s^\sharp, A^\sharp)$, an additional set $afp \subseteq \eta$ contains realizable paths in η. As illustrated in the diagram below, sequence η


$Prob_{s^\sharp}^{A^\sharp}(\rightsquigarrow \mathsf{e})$
σ_1
σ_2
σ_3
σ_4
σ_5
spurious
$Prob_{s^\sharp}^{A^\sharp}(afp)$
$Prob_{s^\sharp}^{A^\sharp}(\eta)$

```
module loop
 N : int;
 invar : N>2;
 bad : bool;
 i : int;
 [a] !bad & i<N ->
  0.9: (i'=i+1) +
  0.1: bad'=(i=N-1)
end module
init !bad & i=0 endinit
```

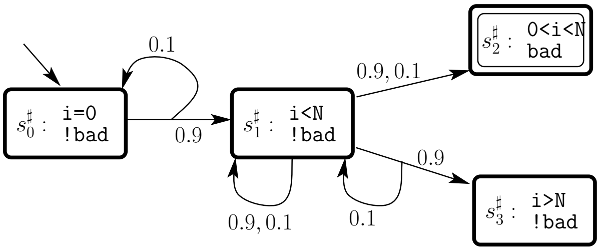


Fig. 2. Cycle program and the quotient automaton with respect to `i=0,bad,i<N`

is ordered by decreasing probability mass – a longer bar means higher probability measure of the path; η is computed incrementally by a variant of best first search [22] in a weighted graph obtained from the MC. Initially η contains only the path with the highest probability, path σ_1.

First we check if path σ_1 is realizable using Lemma 1 (in the diagram we assume σ_1 is spurious), and, if so, we add σ_1 to the set of confirmed paths *afp*. If enough "alleged" probability mass has already accumulated in *afp* to exceed the threshold, i.e. $P_{s^\sharp}^{A^\sharp}(afp) > p$, we check how much of that probability is actually realizable using Lemma 2. If the realizable probability mass exceeds the threshold, the property is refuted, since we can report a realizable counterexample. Otherwise we repeat the process with path σ_2 that has the second highest probability: we add it to η, and check if it is realizable. If realizable, we add σ_2 to *afp*. We continue in this way until either we can refute the property or $n - |afp| = C$, in which case we proceed to phase three. C is a verification parameter set by the user, in the diagram we have $C = 2$.

In the third phase predicates are generated from spurious paths or from weakest preconditions. Then the next iteration of the algorithm commences.

Toy example. Consider the program `Cycle` shown in the left part of Figure 2. The right part shows the quotient automaton with respect to predicates `i=0,bad,i<N` where we omitted the actions and updates. Assume we want to check $Reach_{\leq 0.3}(bad)$. In the quotient automaton, the probability of reaching the `bad` state is 1.0. Let $\mathtt{u}_0$ denote the update `i'=i+1`. We start with the abstract path with (highest) probability 0.81 (distributions are omitted): $\sigma^\sharp = (s_0^\sharp, a, \mathtt{u}_0)(s_1^\sharp, a, \mathtt{u}_0)s_2^\sharp$. Obviously, this path is not realizable as witnessed by the unsatisfiability of its trace formula ψ (see Lemma 1). Taking $C = 1$, we apply Lemma 2 from which we conclude that we have a spurious counterexample. To remove the spurious path $\sigma^\sharp$, we apply interpolation to the trace formula ψ, i.e. we compute a simplification of its prefix $\psi_1^- := N > 2 \wedge i_0 = 0 \wedge \neg bad_0 \wedge i_1 = i_0 + 1 \wedge bad_1 = bad_0 \wedge i_1 < N \wedge \neg bad_1$ that is disjoint with its postfix $\psi_1^+ := i_2 = i_1 + 1 \wedge bad_2 = bad_1 \wedge i_2 \geq N \wedge bad_2$. As an interpolant we obtain $i < N - 1$, add it as a fresh predicate and restart. In the ensuing iteration, the property is established.

4 Experimental Results

We have implemented a prototype of probabilistic CEGAR within the predicate abstraction tool PASS [9]. It is is written in C++ and interfaces to the SMT solver Yices [23] which also supports MAX-SMT. PASS uses CEGAR to obtain predicates based on the interpolant-generating theorem prover FOCI [24]. Experiments were run on a Pentium^{TM} IV 2.6 GHz with 1.5 GB RAM.

Table 1. Statistics. Shown on the left are model parameters and properties studied. On the right we display, apart from reachable state numbers, number of transitions (non-zero entries in transition matrix), and computation time, the number of iterations of the CEGAR loop (refs), of predicates generated (preds), and of abstract paths analyzed (paths). The number of states and transitions are given in thousands, i.e. 34K means 34,000.

Case study				Conventional			Abstraction					
(parameters)			Property	states	trans	time	states	trans	refs	preds	paths	time
	5	315	k=3	5,195K	11,377K	93	34K	36K	9	120	604	72
WLAN	6	315	k=3	12,616K	28,137K	302	34K	42K	9	116	604	88
(BOFF T)	6	315	k=6	12,616K	28,137K	2024	771K	113K	9	182	582	306
	6	9500	k=6	–	–	**TO**	771K	113K	9	182	582	311
	3		p1	41K	52K	10	1K	2K	8	58	28	9
CSMA/CD	4		p1	124K	161K	56	6K	9K	14	100	56	38
(BOFF)	3		p2	41K	52K	10	0.5K	0.9K	12	41	28	10
	4		p2	124K	161K	21	0.5K	1.5K	12	41	44	11
	16	3	p1	2K	3K	5.4	2K	3K	9	46	41	9
BRP	32	5	p1	5K	7K	12	5K	7K	9	64	111	21
(N MAX)	64	5	p1	10K	14K	26	10K	14K	8	95	585	91
	>16	3	p4	∞	–	–	0.5K	0.9K	7	26	17	3
	>16	4	p4	∞	–	–	0.6K	1K	7	27	17	4
	>16	5	p4	∞	–	–	0.7K	1K	8	28	18	5
SW			goodput	∞	–	–	5K	11K	3	40	7	87
			timeout	∞	–	–	27K	44K	3	49	6	89

We consider a selection of case studies summarized in Table 1. Time is measured in seconds. The timeout limit was set to two hours (timeout is indicated by **TO**). We have analyzed the finite-state models with PRISM 3.1.1 (column "Conventional") in comparison with PASS (column "Abstraction"). PRISM is the leading finite-state probabilistic model checker. All models except for the Sliding Window protocol are taken from the PRISM web repository and can be scaled via different parameters. Unconstrained parameters yield infinite-state models (denoted by "∞"). Surprisingly, although our method only guarantees *upper bounds* on probabilities in general, probabilities obtained for all case studies are *tight* upper bounds: they agree with those of PRISM for finite models.

IEEE 802.11 Wireless LAN Protocol (WLAN). The protocol is parameterized with an *exponential* back-off counter limit BOFF and a maximal package send time of T μs. We checked the property: "The maximum probability that either station's back-off counter reaches k" for k=3 and k=6. As shown in Table 1, increasing BOFF from 5 to 6 leads to an exponential increase in model size and running time in PRISM, while in PASS the row is identical for BOFF=5 and BOFF=6. This is because states with back-off counter higher than three can reach a goal state (via a reset), however they do not lie on paths with maximal probability. Hence refinement never splits abstract states with respect to back-offs beyond three. Similarly, for fixed value of BOFF, PASS scales much better in comparison to PRISM with respect to different values of T.

IEEE 802.3 CSMA/CD Protocol (CSMA/CD). Similar to the WLAN protocol, CSMA/CD is parameterized with an *exponential* back-off counter limit BOFF. We analyzed the properties: (p1): "The maximum probability that both stations deliver", and (p2): "The message of any station eventually delivered before 1 backoff". For both properties, as shown in the table, the abstract state space is significantly smaller. Consider property p2. Similar to the WLAN protocol, the size of the abstraction does not change with respect to the size of BOFF. However, the number of paths explored increases with BOFF. The reason is that for greater values of BOFF, there is more branching in the probabilistic model, thus in the abstraction there are more abstract paths being explored.

Bounded Retransmission Protocol (BRP). The BRP protocol has two parameters: N denotes the length of the file to be transmitted, and MAX denotes the maximal number of retransmissions. We have studied "Property 1" and "Property 4" (p1 and p4 in the table). On p1, PRISM outperforms PASS. It appears that this is due to little opportunity for abstraction as, seemingly, a lot of model detail is relevant and has to be discovered by refinement. On the other hand, p4 can be analyzed for an infinite parameter range with PASS, since it is an invariant property with respect to the file length N. Thus the constraint $N > 16$ allows us to verify the property for any possible file length greater than 16.

Sliding Window (SW). This is the standard protocol with lossy channels over an unbounded domain of sequence numbers. Thus the model is infinite and hence we have no comparison to PRISM. We checked goodput properties which consider the difference between the number of sent and received packages. We want to know the probability that the number of sent packages exceeds the number of received packages by a particular constant. PASS checked that, at any time, the probability of the difference exceeding three is at most three percent for windows size four. The second property concerns the probability of a protocol timeout.

Initial Predicates. Probabilistic CEGAR leaves the choice of the initial set of predicates as a parameter. Predicates appearing in the property under study are the minimal option and generally a good one (BRP, CSMA). Further, control locations of the program might also be part of the initial abstraction to avoid non-determinism between commands in the abstraction. Thus adding predicates from the initial condition (WLAN) or additionally from guards (SW) can improve running times. PASS features several automatic modes; Table 1 contains data obtained via the respective best mode.

Discussion. To compete with PRISM on finite models, the benefit of state space reduction has to offset the cost of repeating the CEGAR loop. On infinite or very large models only PASS can be used. We observe that abstraction for

WLAN and CSMA/CD, PASS is superior to the conventional approach, due to the significantly smaller abstract state space. Notably the analysis of BRP for $N > 16$ can be considered a parametric analysis: p4 is proven for any such N.

5 Conclusion

This paper explores fundamental questions and pragmatic issues of probabilistic abstraction refinement. The main contribution lies in our treatment of abstract counterexamples which are finite Markov chains, instead of finite paths. Spurious counterexamples are analyzed with interpolation-based predicate inference, leading to a refined model which closes the CEGAR loop. The resulting theory and tool work smoothly, as shown by our experimental evaluation. Our next goal is to enable model checking of full PCTL.

Acknowledgements. Thanks to E. Moritz Hahn for helping us with the implementation, and to the anonymous reviewers for their valuable comments.

References

1. Hansson, H., Jonsson, B.: A Logic for Reasoning about Time and Reliability. Formal Asp. Comput. 6, 512–535 (1994)
2. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995)
3. Hinton, A., Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM: A Tool for Automatic Verification of Probabilistic Systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006 and ETAPS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
4. Katoen, J.P., Khattri, M., Zapreev, I.S.: A Markov reward model checker. In: QEST, pp. 243–244 (2005)
5. Graf, S., Saídi, H.: Construction of Abstract State Graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
6. Ball, T., Rajamani, S.K.: The SLAM project: debugging system software via static analysis. In: POPL, pp. 1–3 (2002)
7. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: POPL, pp. 58–70 (2002)
8. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-Guided Abstraction Refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
9. Wachter, B., Zhang, L., Hermanns, H.: Probabilistic Model Checking Modulo Theories. In: QEST, pp. 129–138 (2007)
10. Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. Nord. J. Comput. 2, 250–273 (1995)
11. Chatterjee, K., Henzinger, T.A., Majumdar, R.: Counterexample-Guided Planning. In: UAI (2005)
12. Han, T., Katoen, J.P.: Counterexamples in probabilistic model checking. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 72–86. Springer, Heidelberg (2007)

13. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. J. Comput. Syst. Sci. 43, 425–440 (1991)
14. McMillan, K.L.: Lazy abstraction with interpolants. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 123–136. Springer, Heidelberg (2006)
15. Aljazzar, H., Hermanns, H., Leue, S.: Counterexamples for timed probabilistic reachability. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 177–195. Springer, Heidelberg (2005)
16. D'Argenio, P.R., Jeannet, B., Jensen, H.E., Larsen, K.G.: Reduction and Refinement Strategies for Probabilistic Analysis. In: Hermanns, H., Segala, R. (eds.) PROBMIV 2002, PAPM-PROBMIV 2002, and PAPM 2002. LNCS, vol. 2399, pp. 57–76. Springer, Heidelberg (2002)
17. de Alfaro, L., Roy, P.: Magnifying-lens abstraction for markov decision processes. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 325–338. Springer, Heidelberg (2007)
18. Kwiatkowska, M., Norman, G., Parker, D.: Game-based Abstraction for Markov Decision Processes. In: QEST, pp. 157–166 (2006)
19. Fecher, H., Leucker, M., Wolf, V.: Don't Know. In: Valmari, A. (ed.) SPIN 2006. LNCS, vol. 3925, pp. 71–88. Springer, Heidelberg (2006)
20. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Chichester (1994)
21. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: POPL, pp. 232–244 (2004)
22. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality af A*. J. ACM 32, 505–536 (1985)
23. Dutertre, B., de Moura, L.M.: A Fast Linear-Arithmetic Solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
24. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. 345, 101–121 (2005)